Yazılım Mühendisliği
Yazılım Gereksinimleri ve Modelleme

Dr. Öğr. Üyesi Emrah ÖZKAYNAK

Dersin Amacı

Amaç: Bu dersin amacı, yazılım gereksinim mühendisliğindeki en güncel süreçleri, yöntemleri ve teknikleri işlemektir. Ayrıca, yazılım geliştirme projelerinde kaliteli yazılım gereksinimleri tanımlamak için gerekli detaylı bilgi ve beceriyi sağlar.

- ✓Planlama raporu içeriği
- ✓Yazılım Yaşam Döngüsü
- ✓Analiz- Gereksinim nedir? Gereksinim türleri
- ✓Gereksinim Verisi Toplama Yöntemleri
- ✓Kullanıcı Arayüz Prototipleme (KAP)
- ✓Sistem Analiz Raporu

Proje Planı(Faaliyet-Zaman-Maliyet Çizelgesi)

Proje Kaynakları-

1. İnsan kaynakları : Proje şamalarında görev alacak personelin nitelikleri ve çalışma zamanları
2. Sistemin geliştirilmesinde ve nihai sistemde kullanılacak donanım kaynaklarının edinilme zaman çizelgesi
3. Sistem geliştirme sunumunda kullanılacak yazılım kaynaklarının edinilme tarihleri

Bu aşamanın en önemli görünür çıktısı projenin çıktılarına ait zaman çizelgesidir.

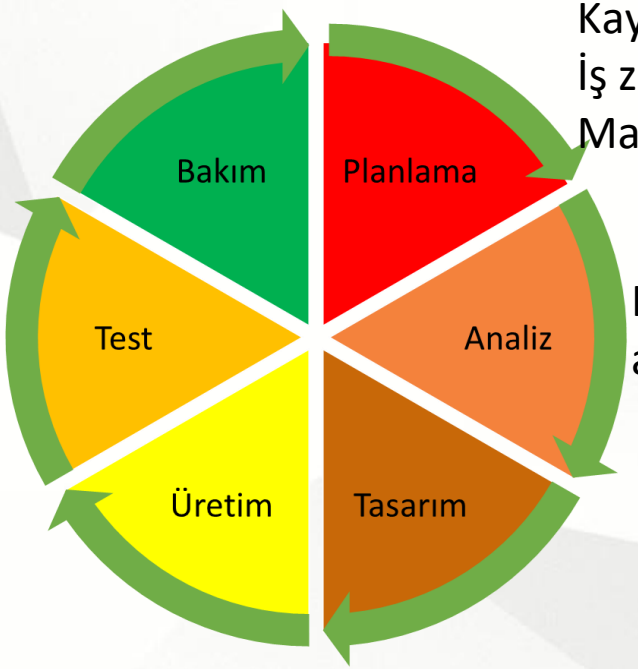
İlk maliyet hesaplama bu aşamada olmasına karşın proje planı raporunda genellikle yer almaz.

Projenin başlangıç tarihi : .../.../ 20...

20... Yılı Fiyatlarıyla

Faaliyet	I. Yıl				II. Yıl				III. Yıl	Maliyet (Bin TL)
	1-3. ay	4-6. ay	7-9. ay	10-12. ay	13-15. ay	16-18. ay	19-21. ay	22-24. ay	...	
1.										
1.1.										
1.2.										
2.										
2.1.										
2.2.										
2.3										
3.										
4.										
5.										
Toplam Tutar										

Yazılım Yaşam Döngüsü



Kaynak: –insan, donanım-yazılım
İş zaman çizelgesi
Maliyet hesabı

Halihazırda işin yapılmakta olan işlerin algoritmalarının belirlenmesi

Analiz (Çözümleme)

✓ Amaç: Sistemin işlevlerini ve kesin gereksinimleri açıklığa kavuşturmak ve sonucunda bunları belirli bir formatta **dokümante** etmektir.

Analiz çalışması; müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi vb. rollerin bir araya geldiği gruplar tarafından yapılabilir. İhtiyaçların net olmadığı durumlarda yazılım mühendisi ve müşteri arasında iletişim ve birlikte çalışmanın çok daha fazla olması gerekir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanıcı dokümanlarının taslakları ile ve test plan dokümanları da oluşturulabilir.

Gereksinim Nedir?

Gereksinim, sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.

✓Gereksinim işlevlerinin nasıl yerine getirileceği ile ilgili değildir. Ne olduğu ile ilgilidir.

İşlevsel Gereksinim

✓İşlevsel gereksinim (Kullanıcı gereksinimi); sistem ile çevresi arasındaki iletişimi belirleyen gereksinimlerdir. Geliştirilecek olan sistemi kullanacak aktörlerin ihtiyaçlarını karşılayacak gereksinimlerdir.

- bordronun ne zaman alınacağı
- hangi verilerin alınacağı
- çıktı formatı

İşlevsel Olmayan Gereksinimler

✓İşlevsel olmayan gereksinimler, kullanıcının sorunundan bağımsız olarak çözülmesi gereken işlemlerdir.

✓Sistem kısıtları olarak ta adlandırılabilir

- kullanılacak bilgisayarın türü
- yazılım geliştirme ortamı
- kullanılacak veri tabanı yönetim sistemi

Gereksinim Türleri

- ✓Fiziksel Çevre
- ✓Arayüzler
- ✓Kullanıcı ve İnsan etmeni
- ✓İşlevsellik
- ✓Belgeleme
- ✓Veri
- ✓Kaynaklar
- ✓Güvenlik
- ✓Kalite Güvencesi

Fiziksel Çevre

- ✓İşlevlerin geliştirileceği, işletileceği aygıtlar nerededir?
- ✓Sistem tek bir yerde mi olacak? Fiziksel olarak ayrı yerler söz konusu mu?
- ✓Sıcaklık nem oranı veya manyetik etkileşim gibi çevresel kısıtlamalar var mı?

Arayüzler

- ✓Girdiler bir mi yoksa birden çok sistemden mi geliyor?
- ✓Çıktılar bir mi yoksa birden çok sisteme mi gidiyor?
- ✓Verilerin nasıl biçimlendirileceğine ilişkin bir yol var mı?
- ✓Verilerin kullanılacağı önerilen bir ortam var mı?

Kullanıcı ve İnsan etmeni

- ✓Sistemi kim kullanacak?
- ✓Farklı tiplerde kullanıcılar olacak mı?
- ✓Her bir kullanıcı tipinin yetenek düzeyi nedir?
- ✓Her kullanıcı tipi için ne tür eğitimler gerekli?
- ✓Bir kullanıcının sistemi kötü amaçlı kullanması ne ölçüde zordur?

İşlevsellik

- ✓Sistem ne yapacak?
- ✓Sistem bunu ne zaman gerçekleştirecek?
- ✓Sistem nasıl ve ne zaman değiştirilebilir ve/veya güçlendirilebilir?
- ✓Çalışma hızı, yanıt süresi ya da çıktı üzerinde kısıtlayıcı etmenler var mı?

Belgeleme

- ✓Ne kadar belgeleme gereklidir?
- ✓Belgeleme hangi kullanıcı kitlesini hedeflemektedir?
- ✓Geliştirilecek sistemin bakım sürecine ait belgeleme gerekli mi?

Veri

- ✓Hem giriş hem çıkış için verinin biçimi ne olmalıdır?
- ✓Bu veri ne sıklıkla alınacak veya gönderilecektir?
- ✓Bu verinin doğruluk ölçüsü ne olmalıdır?
- ✓Hesaplamaların sonuçları hangi hassasiyette olmalıdır?
- ✓Sistemde ne kadar veri akışı olacaktır?
- ✓Verinin depolanma ve işleme süresi ne kadar olacak?

Kaynaklar

- ✓Sistemi kurmak, kullanmak ve bakımını yapmak için ne kadar malzeme, personel ve diğer kaynaklara ihtiyaç var?
- ✓Geliştiriciler hangi yeteneklere sahip olmalı?
- ✓Sistem ne kadar fiziksel yer kaplayacak?
- ✓Güç, ısıtma ve soğutma için kısıtlar nelerdir?
- ✓Geliştirim için tavsiye edilen bir zaman çizelgesi var mı?

Güvenlik

- ✓Sisteme ya da bilgiye erişim denetlenmeli midir?
- ✓Bir kullanıcının verisi diğerinden nasıl ayrılacaktır?
- ✓Kullanıcı programları, diğer program ve işletim sisteminden nasıl ayrı tutulacaktır?
- ✓Sistem hangi sıklıkla yedeklenecektir?
- ✓Yedek kopyaları başka yerde saklanacak mıdır?
- ✓Yangın ve hırsızlığa karşı ne tür önlemler alınacaktır?
- ✓Internet erişimi var mı? Güvenlik kullanılıyor mu?

Kalite Güvencesi

- ✓Güvenirlilik için gereksinimler nelerdir?
- ✓Sistemin özellikleri insanlara nasıl aktarılmalıdır?
- ✓Sistem çökmeleri arasında öngörülen zaman aralığı nedir?
- ✓Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri nelerdir?

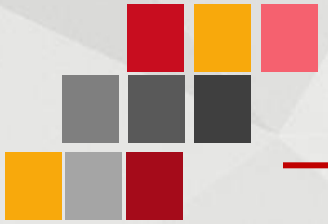
Gereksinim Özellikleri

Gereksinimler üç amaca hizmet eder

- ✓Geliştiricilerin, müşterilerin sistemin nasıl çalışmasını istediklerini anlamalarını sağlar.
- ✓Gereksinimler, sonuç sistemin ne özellikte ve işlevsellikte olacağını söyler.
- ✓Gereksinimler sınama ekibine, kullanıcıyı, sunulan sistemin istenen sistem olduğuna ikna etmek için neler göstermeleri gerektiğini söyler.

Doğrulama Süreci

1. Gereksinimler doğru oluşturulmuş mu?
2. Gereksinimler tutarlı mı?
3. Gereksinimler tam mı? (Dışsal tamlık / İçsel tamlık)
4. Gereksinimler gerçekçi mi?
5. Her gereksinim kullanıcı tarafından istenen bir şeyi mi tanımlamaktadır?
6. Gereksinimler doğrulanabilir mi?
7. Gereksinimler izlenebilir mi?



Yazılım Mühendisliği
Temel Süreçler - PLANLAMA

Dr. Öğr. Üyesi Emrah ÖZKAYNAK

HEDEFLER

- ✓ Proje kaynakları

  İnsan, donanım ve yazılım kaynakları

Planlama

Proje planlama aşamasında yapılan işlemler

- Kaynakların Belirlenmesi
- Maliyetlerin Kestirilmesi
- Proje Ekip Yapısının Oluşturulması
- Ayrıntılı Proje Planının Yapılması
- Projenin İzlenme Yönteminin Belirlenmesi

Çıktı: Proje Planı

Proje planı, tüm proje süresince güncellenerek, kaynak kullanım planlarının doğruluğu gözlemlenir.

Proje Kaynakları

- ✓ İnsan Kaynakları
- ✓ Donanım Kaynakları
- ✓ Yazılım Kaynakları

Planlama; bu kaynakların tanımını yapar ve zaman kullanımı, görev süreleri, edinilme zamanlarını planlar

İnsan Kaynakları

- Planlama; hangi tür elemanların, hangi süre ile ve projenin hangi aşamalarında yer alacağını belirler

Proje Yöneticisi	Donanım Ekip Lideri
Yazılım Ekip Lideri	Donanım Mühendisi
Web Tasarımcısı	Ağ Uzmanı
Sistem Tasarımcısı	Yazılım Destek Elemanı
Programcı	Donanım Destek Elemanı
Sistem Yöneticisi	Eğitmen
Veri Tabanı Yöneticisi	Denetleyici
Kalite Sağlama Yöneticisi	Çağrı Merkezi Elemanı

Donanım Kaynakları

- ✓ Donanım Kaynakları:
  - Ana Bilgisayarlar
  - Sunucular (Web, E-posta, Veri Tabanı)
  - Kullanıcı Bilgisayarları (PC)
  - Yerel Alan Ağı (LAN) Alt Yapısı
  - Geniş Alan Ağı (WAN) Alt Yapısı
- ✓ Yazılımın geliştirileceği ortam, gerçek kullanım ortamı dışında olmalıdır.
- ✓ Öte yandan, geliştirme ve uygulama ortamlarının aynı konfigürasyonda olmaları, ileride kurulum sırasında ortaya çıkabilecek taşıma sorunlarını büyük ölçüde giderecektir.

Yazılım Kaynakları

- ✓ Büyük ölçekte otomatik hale getirilmiş ve bilgisayar destekli olarak kullanılmaktadır.
- ✓ Bilgisayar Destekli Tasarım (CAD) ve Bilgisayar Destekli Mühendislik (CASE) araçları olarak bilinmektedirler

- ✓ Test araçları
  - Yazılımı doğrulama ve geçerleme işlemlerinde kullanılır. Test verisi üreticiler, otomatik test yordamları, ...
- ✓ Prototipleme ve simülasyon araçları
  - Geliştirmenin erken aşamalarında kullanıcıya, sonuç ürünün çalışması ile ilgili fikir veren ve yönlendiren araçlar.
- ✓ Bakım araçları
  - Programın bakımını kolaylaştıran, bir kaynak koddan program şemalarının üretilmesini, veri yapısının ortaya çıkarılmasını sağlayan araçlar.
- ✓ Destek araçları
  - İşletim sistemleri, ağ yazılımları, e-posta ve ortam yönetim araçları.

Proje Maliyetleri

- ✓ Maliyet kestirimi; bir bilgi sistemi ya da yazılım için gerekebilecek iş gücü ve zaman maliyetlerinin üretimden önce belirlenebilmesi için yapılan işlemlerdir.

- ✓ Kullanılan Unsurlar
  - Geçmiş projelere ilişkin bilgiler
  - Proje ekibinin deneyimleri
  - İzlenen geliştirme modeli

birden çok kez uygulanabilir

Maliyet yönetimi sayesinde;

- ✓ Gecikmeler önlenir
- ✓ Bilgi sistemi geliştirme süreci kolaylaştırılır
- ✓ Daha etkin kaynak kullanımı sağlanır
- ✓ İş zaman planı etkin olarak gerçekleştirilir
- ✓ Ürün sağlıklı olarak fiyatlandırılır
- ✓ Ürün zamanında ve hedeflenen bütçe sınırları içerisinde bitirilir

Gözlemlenebilecek değerler

- ✓ Projenin toplam süresi
- ✓ Projenin toplam maliyeti
- ✓ Projede çalışan eleman sayısı, niteliği, çalışma süresi
- ✓ Toplam satır sayısı
- ✓ Bir satırın maliyeti (ortalama)
- ✓ Bir kişi/ay'da gerçekleştirilen satır sayısı
- ✓ Toplam işlev sayısı
- ✓ Bir işlevin maliyeti
- ✓ Bir kişi/ay'da gerçekleştirilen işlev sayısı
- ✓ Bir kişi/ay'da maliyeti

Maliyet Kestirim Yöntemleri

1. Projenin boyut türüne göre
   - Proje büyüklüğünü kestiren yöntemler
   - Proje zaman ve işgücünü kestiren yöntemler
2. Projelerin büyüklüğüne göre
   - Makro yöntemler (büyük boyutlu projeler 30 kişi-yıl)
   - Mikro Yöntemler (orta ve küçük boyutlu projeler)
3. Uygulanış biçimlerine göre
   - Yalın düzeyde
   - Orta düzeyde
   - Ayrıntılı düzeyde

Maliyet Kestirim Yöntemleri

4. Değişik aşamalarda kullanılabilirlik

- Planlama ve analiz aşamasında kullanılabilen
- Tasarım aşamasında kullanılabilen
- Gerçekleştirim aşamasında kullanılabilen yöntemler

5. Yöntemlerin yapılarına göre

- Uzman deneyimine gereksinim duyan
- Önceki projelerdeki bilgileri kullanan yöntemler

İşlev Noktaları Yöntemi

- ✓ İşlev noktaları geliştirmenin erken aşamalarında (analiz aşamasında) saptanan bir değerdir.
- ✓ Sistemin oluşturulduğu ortamdan bağımsız elde edilir.
- ✓ Problem tanımı girdi olarak alınarak üç temel adım izlenir:
  - Problemin bilgi ortamının incelenmesi
  - Problemin teknik karmaşıklığının incelenmesi
  - İşlev noktası hesaplama

Problemin bilgi ortamının incelenmesi

- ✓ Kullanıcı Girdileri: personel sicil bilgileri, personel izin bilgileri gibi
- ✓ Kullanıcı Çıktıları: her türlü mantıksal çıktı; raporlar, ekran çıktıları, hata iletileri,...
- ✓ Kullanıcı Sorguları: personel sicil bilgilerinin sorgulaması, personel maaş bilgilerinin sorgulaması
- ✓ Dosyalar: Her türlü mantıksal bilgi yığını, tablolar, veri tabanları
- ✓ Dışsal arayüzler: Başka programlarla veri iletimi. import/export

Yukarıda verilen durumlara ait sayılar ağırlık faktörleriyle çarpılarak toplam işlemi yapılır. Çıkan değer Ayarlanmamış İşlev Noktası (AİN) olarak adlandırılır.

Problemin teknik karmaşıklığının incelenmesi

1. Uygulama, güvenilir yedekleme ve kurtarma gerektiriyor mu?
2. Veri iletişimi gerektiriyor mu?
3. Dağıtılmış İşlemler var mı?
4. Performans kritik mi?
5. Girdiler, çıktılar, dosyalar ya da sorgular karmaşık mı?
6. İçsel işlemler karmaşık mı?
7. Tasarlanacak kod yeniden kullanılabilir mi?
8. Dönüştürme ve kurulun tasarımda dikkate alınacak mı?

*****

Cevaplar 0 ile 5 arasında puanlandırılır

Bunlar hesaplanıp toplanarak Teknik Karmaşıklık Faktörü (TKF) elde edilir.

İşlev noktası sayısı hesaplama

- ✓ İN=AİN*(0,65*0,01*TKF)

Değişik amaçlarla kullanılabilir

- **Üretkenlik** = İN / Kişi-Ay
- **Kalite** = Hatalar / İN
- **Maliyet** = $ / İN

Satır Sayısı Kestirimi

Assembly	300
Cobol	100
Fortran	100
Pascal	90
C	90
Ada	70
Nesne Kökenli Diller	30
4. Kuşak Dilleri	20
Kod Üreticiler	15

Örneğin,
İN=300 ise ve Nesne Tabanlı bir dilde geliştirme yapılırsa
Satır Sayısı=300*30, kod üreteçleri ile geliştirme yapılırsa
Satır Sayısı=300*15 satır kadarlık kod yazılması gerekir.
Buradaki kod satır sayısını açıklamalar hariç olarak düşünmek gerekir.

Yazılım Mühendisliği
Yazılım Yaşam Döngüsü

Dr. Öğr. Üyesi Emrah ÖZKAYNAK

Yazılım Yaşam Döngüsü (Software Development Life Cycle - SDLC)

- ✓ Bir yazılımın konsept aşamasından başlayarak üretim, dağıtım, bakım ve sonlandırma süreçlerine kadar olan tüm aşamaları kapsayan bir süreçtir.
- ✓ Bu döngü, yazılımın sistematik bir şekilde geliştirilmesi, test edilmesi ve sürdürülmesi için izlenen bir dizi adım içerir.
- ✓ SDLC, yazılım projelerinde kaliteyi artırmak, maliyetleri azaltmak ve teslim sürelerini optimize etmek amacıyla kullanılır.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

1. Gereksinim Analizi (Requirement Analysis)

- ✓ Gereksinim Analizi (Requirement Analysis), yazılım geliştirme sürecinin en kritik aşamalarından biridir.
- ✓ Bu aşamada, yazılımın hangi amaçla geliştirileceği, ne tür sorunları çözeceği ve kullanıcıların ya da müşterilerin hangi ihtiyaçlarını karşılayacağı belirlenir.
- ✓ Gereksinimlerin doğru şekilde belirlenmesi, yazılımın başarıyla tamamlanması ve beklentileri karşılaması açısından hayati bir öneme sahiptir.

Gereksinim Analizinin Aşamaları

a) Gereksinim Toplama (Requirement Elicitation)
b) Gereksinimlerin Sınıflandırılması (Requirement Classification)
c) Gereksinimlerin Analizi ve Önceliklendirilmesi (Requirement Analysis and Prioritization)
d) Gereksinimlerin Belgelendirilmesi (Requirement Documentation)
e) Gereksinimlerin Onaylanması (Requirement Validation)

a) Gereksinim Toplama (Requirement Elicitation)

- ✓ Gereksinim analizi sürecinin ilk adımı, gereksinimlerin toplanmasıdır. Bu aşamada, yazılımın hedef kitlesinden ve ilgili paydaşlardan (müşteri, son kullanıcılar, iş birimleri vb.) ihtiyaçlar ve beklentiler alınır. Bu bilgiler genellikle aşağıdaki yöntemlerle toplanır:
- ✓ **Görüşmeler (Interviews):** Projenin gereksinimlerini belirlemek için paydaşlarla yüz yüze görüşmeler yapılır.
- ✓ **Anketler (Surveys/Questionnaires):** Geniş kitlelerden veri toplamak amacıyla anketler düzenlenir.

a) Gereksinim Toplama (Requirement Elicitation)

- ✓ **Gözlem (Observation):** Kullanıcıların mevcut sistemle nasıl çalıştığı gözlemlenerek yeni yazılımın ne tür işlevler sunması gerektiği anlaşılır.
- ✓ **Mevcut Belgelerin İncelenmesi (Document Analysis):** Mevcut sistemlere ait dokümantasyon incelenir ve önceki sistemlerle ilgili bilgiler toplanır.
- ✓ **Prototipleme (Prototyping):** Prototipler oluşturularak kullanıcıların ihtiyaçlarını daha net anlamak için kullanılabilir.

b) Gereksinimlerin Sınıflandırılması (Requirement Classification)

- ✓ Toplanan gereksinimler, farklı kategorilere ayrılır ve sınıflandırılır. Genellikle bu gereksinimler iki ana başlık altında toplanır:
- ✓ **Fonksiyonel Gereksinimler (Functional Requirements):** Yazılımın ne yapacağını tanımlar. Örneğin, bir e-ticaret platformu için ödeme sistemi, kullanıcı giriş/çıkış işlemleri gibi işlemler fonksiyonel gereksinimlere girer.

- ✓ **Fonksiyonel Olmayan Gereksinimler (Non-Functional Requirements):** Yazılımın nasıl çalışacağına dair beklentileri içerir. Performans, güvenlik, kullanılabilirlik, ölçeklenebilirlik gibi özellikler bu kategoriye girer.

c) Gereksinimlerin Analizi ve Önceliklendirilmesi (Requirement Analysis and Prioritization)

- ✓ Bu aşamada toplanan gereksinimler detaylı bir şekilde analiz edilir. Her gereksinimin ne kadar önemli olduğu, ne derece kritik olduğu ve projenin genel hedefleriyle nasıl uyuştuğu değerlendirilir.

- ✓ Ayrıca gereksinimlerin teknik olarak uygulanabilir olup olmadığı, maliyet ve zaman açısından ne kadar uygun olduğu da göz önünde bulundurulur.

- ✓ **Önceliklendirme (Prioritization):** Yazılımın geliştirme sürecinde hangi gereksinimlerin daha öncelikli olduğu belirlenir. Bu sayede en kritik işlevler öncelikle hayata geçirilir.

d) Gereksinimlerin Belgelendirilmesi (Requirement Documentation)

- ✓ Gereksinimlerin detaylandırılıp yazılı hale getirilmesi aşamasıdır. Bu belgeler, yazılımın geliştirilmesi, test edilmesi ve bakım süreçlerinde referans olarak kullanılır. Genellikle iki temel belge oluşturulur:

- ✓ **Gereksinim Özellik Belgesi (Software Requirements Specification - SRS):** Tüm gereksinimlerin detaylı bir şekilde açıklandığı belgedir. Bu belge, yazılım geliştirme ekibine, müşteri ve paydaşlara yazılımın ne yapması gerektiği konusunda net bir yol haritası sunar.

- ✓ **Kullanıcı Hikayeleri (User Stories):** Özellikle çevik yazılım geliştirme metodolojilerinde kullanılır. Bu hikayeler, kullanıcıların sistemle nasıl etkileşime gireceğini kısa ve anlaşılır şekilde tanımlar.

e) Gereksinimlerin Onaylanması (Requirement Validation)

- ✓ Gereksinimlerin doğru anlaşıldığından ve kullanıcı ya da müşterilerin beklentilerini karşıladığından emin olmak için gereksinimler onaylanır.
- ✓ Bu süreçte paydaşlarla bir araya gelinir ve gereksinimler gözden geçirilir.
- ✓ Herhangi bir eksiklik ya da hata varsa, bu aşamada düzeltilir.

Gereksinim Analizinde Dikkat Edilmesi Gereken Hususlar

- ✓ **Paydaş Katılımı:** Gereksinimlerin doğru belirlenmesi için tüm paydaşların sürece etkin şekilde katılımı sağlanmalıdır.
- ✓ **Değişiklik Yönetimi:** Gereksinimlerin zamanla değişebileceği göz önünde bulundurulmalıdır. Gereksinim değişiklikleri, yazılım geliştirme sürecine dahil edilmeli ve iyi bir değişiklik yönetimi yapılmalıdır.
- ✓ **Net ve Anlaşılır Olma:** Gereksinimler, net, spesifik ve ölçülebilir olmalıdır. Yanlış anlamaları önlemek için dil açık olmalı ve her gereksinim kolayca test edilebilir olmalıdır.
- ✓ **Çatışmaların Çözümü:** Farklı paydaşlar arasında gereksinim çatışmaları ortaya çıkabilir. Bu tür çatışmalar dikkatlice ele alınmalı ve yazılımın genel hedefleri doğrultusunda çözülmelidir.

Gereksinim Analizinin Önemi

- ✓ **Doğru Planlama:** Gereksinim analizi sayesinde yazılımın kapsamı net bir şekilde belirlenir, bu da doğru bir proje planlaması sağlar.
- ✓ **Hataları Azaltma:** İyi bir gereksinim analizi, yazılım geliştirme sürecinde ortaya çıkabilecek hataları ve yanlış anlamaları azaltır. Gereksinimlerin doğru anlaşılması, hataların erken tespit edilmesini sağlar.
- ✓ **Müşteri Memnuniyeti:** Gereksinimlerin net bir şekilde belirlenmesi ve karşılanması, müşteri memnuniyetini artırır. Müşterinin ihtiyaçlarını tam olarak karşılayan bir yazılım geliştirilmiş olur.
- ✓ **Zaman ve Maliyet Tasarrufu:** Gereksinimlerin doğru bir şekilde analiz edilmesi, projede gereksiz revizyonları ve değişiklikleri önleyerek zaman ve maliyet tasarrufu sağlar.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- **1. Gereksinim Toplama (Requirement Elicitation)**
- İlk aşamada, platformu kullanacak tüm paydaşlarla görüşmeler yapılır ve gereksinimler toplanır.

- Bu paydaşlar arasında platformun sahibi (işletme sahibi), müşteriler, satıcılar ve teknik ekip yer alır.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **İşletme Sahibi ile Görüşme:**
- Platformun temel işlevleri: Ürün listeleme, arama, sepete ekleme, ödeme alma.
- Gelir modeli: Satıcılardan komisyon alınması.
- Ürün kategorileri, kampanyalar ve indirim sistemleri.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **Müşterilerle Anket ve Gözlem:**
- Müşterilerin kullanıcı deneyimi talepleri.
- Kolayca ürün arama ve filtreleme ihtiyacı.
- Güvenli ödeme sistemlerinin gerekliliği.
- Kullanıcı dostu, hızlı ve mobil uyumlu bir arayüz beklentisi.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **Satıcılarla Görüşme:**
- Satıcıların ürünlerini kolayca yükleyip yönetebileceği bir panel talebi.
- Satış takibi, stok yönetimi ve raporlama ihtiyaçları.
- İade ve değişim süreçlerinin yönetimi.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **Teknik Ekiple Toplantılar:**

- Teknik gereksinimler, platformun ölçeklenebilirliği, güvenlik gereksinimleri.

- Üçüncü taraf entegrasyonları (ödeme sistemleri, kargo firmaları).

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ 2. **Gereksinimlerin Sınıflandırılması (Requirement Classification)**
- ✓ Toplanan gereksinimler iki ana kategoriye ayrılır: **Fonksiyonel gereksinimler** ve **fonksiyonel olmayan gereksinimler**.
- **Fonksiyonel Gereksinimler:**
  - **Üye Kaydı:** Müşterilerin ve satıcıların siteye üye olabilmesi.
  - **Ürün Arama ve Filtreleme:** Kullanıcılar ürünleri kategori, fiyat, marka gibi kriterlere göre arayıp filtreleyebilmelidir.
  - **Sepete Ekleme ve Satın Alma:** Müşteriler ürünleri sepete ekleyebilmeli, ödeme sürecini tamamlayabilmelidir.
  - **Satıcı Paneli:** Satıcıların ürün listelemesi, stok takibi yapması, siparişleri ve iade süreçlerini yönetmesi.
  - **Sipariş Takibi:** Müşteriler siparişlerinin durumunu görebilmeli ve kargo takip bilgilerini alabilmelidir.
  - **İade ve Değişim:** Müşteriler iade ve değişim işlemlerini sistem üzerinden başlatabilmelidir.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

✓ **Fonksiyonel Olmayan Gereksinimler:**

- **Performans:** Site hızlı olmalı ve yüksek trafik altında bile düzgün çalışmalıdır.
- **Güvenlik:** Kullanıcı verileri güvenli bir şekilde saklanmalı ve ödeme işlemleri güvenli olmalıdır.
- **Kullanılabilirlik:** Mobil cihazlarla uyumlu, kullanıcı dostu bir arayüz sunulmalıdır.
- **Ölçeklenebilirlik:** Platform, ileride artacak kullanıcı sayısına ve ürün hacmine uyum sağlayabilecek kapasitede olmalıdır.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **3. Gereksinimlerin Analizi ve Önceliklendirilmesi (Requirement Analysis and Prioritization)**
- ✓ Toplanan gereksinimler analiz edilir ve önceliklendirilir.
- **Kritik Gereksinimler:** Kullanıcı kaydı, ürün arama ve filtreleme, sepet ve satın alma süreci gibi platformun temel işlevleri, ilk aşamada geliştirilmelidir.
- **Orta Öncelikli Gereksinimler:** Satıcı panelinin gelişmiş özellikleri, detaylı sipariş ve stok yönetimi, kampanya modülü gibi unsurlar daha sonraki aşamalarda devreye alınabilir.
- **Düşük Öncelikli Gereksinimler:** Kullanıcı yorumları ve puanlama sistemi gibi özellikler, temel fonksiyonlar tamamlandıktan sonra eklenebilir.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **4. Gereksinimlerin Belgelendirilmesi (Requirement Documentation)**
- ✓ Belirlenen gereksinimler belgelenir ve hem teknik ekibin hem de diğer paydaşların kullanımına sunulur.
- **Gereksinim Özellik Belgesi (SRS):**
  - Bu belgede, platformun tüm işlevsel ve işlevsel olmayan gereksinimleri detaylandırılır. Örneğin, kullanıcı kaydı nasıl çalışacak, ödeme işlemleri hangi sistemlerle entegre edilecek, veri güvenliği için hangi protokoller kullanılacak gibi detaylar yer alır.
- **Kullanıcı Hikayeleri (User Stories):**
  - "Bir müşteri olarak, sepetime ürün ekleyip hızlıca ödeme yapabilmeliyim." gibi basit ve anlaşılır kullanıcı hikayeleri oluşturulur. Bu hikayeler, geliştirme ekibine rehberlik eder.

Örnek - Proje: E-Ticaret Platformu Gereksinim Analizi

- ✓ **5. Gereksinimlerin Onaylanması (Requirement Validation)**
- ✓ Belgelendirilen gereksinimler paydaşlarla tekrar gözden geçirilir ve onaylanır. Örneğin, işletme sahibi platformun gelir modeli ile ilgili ek talepler sunabilir, ya da müşteriler daha iyi bir kullanıcı deneyimi talep edebilir. Bu geri bildirimler doğrultusunda gereksinimler yeniden değerlendirilir.
- **Prototip Onayı:** Gereksinimler doğrultusunda basit bir prototip geliştirilir ve paydaşlarla paylaşılır. Kullanıcı arayüzü ve temel işlevlerin tasarımı hakkında geri bildirimler alınır ve gerekli düzenlemeler yapılır.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **1. Gereksinim Toplama (Requirement Elicitation)**
- ✓ Bu aşamada, bankanın paydaşlarıyla görüşmeler yapılır, kullanıcı ihtiyaçları analiz edilir ve teknik gereksinimler toplanır.
- ✓ **Banka Yönetimi ile Görüşme:**
- Bankanın mobil bankacılık stratejisi.
- Hangi hizmetlerin mobil uygulamada sunulacağı: Para transferleri, fatura ödemeleri, hesap hareketlerinin izlenmesi, kredi başvuruları gibi.
- Banka güvenlik politikaları ve yasal gereklilikler.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **Müşteriler ile Anket ve Gözlem:**
- Müşterilerin en sık kullandığı bankacılık işlemleri (para transferi, bakiye görüntüleme vb.).
- Mobil uygulamanın nasıl bir deneyim sunması gerektiği: Kolay erişim, hızlı işlem yapma, kullanıcı dostu arayüz.
- Güvenlik endişeleri: Parmak izi veya yüz tanıma gibi biyometrik güvenlik özellikleri talebi.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **Teknik Ekip ile Görüşme:**
- Uygulamanın bankanın mevcut sistemleriyle entegrasyonu.
- Güvenlik gereksinimleri, veri şifreleme ve güvenli oturum yönetimi.
- Uygulamanın hangi platformlarda (iOS, Android) destekleneceği.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **2. Gereksinimlerin Sınıflandırılması (Requirement Classification)**
- ✓ Toplanan gereksinimler, işlevsel ve işlevsel olmayan gereksinimler olarak iki ana gruba ayrılır.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **Fonksiyonel Gereksinimler:**

- **Kullanıcı Kimlik Doğrulama:** Müşteriler, kullanıcı adı ve şifre ile giriş yapabilmeli. Ayrıca, parmak izi, yüz tanıma gibi biyometrik yöntemlerle de giriş yapılabilmeli.
- **Hesap Görüntüleme:** Müşteriler hesap bakiyelerini, hareketlerini ve detaylarını görüntüleyebilmelidir.
- **Para Transferi:** Müşteriler hızlıca başka hesaba para transferi (EFT, havale) yapabilmelidir.
- **Fatura ve Ödeme İşlemleri:** Kullanıcılar fatura ödemelerini ve kredi kartı borcu ödeme işlemlerini yapabilmelidir.
- **Kredi Başvurusu:** Kullanıcılar mobil uygulama üzerinden ihtiyaç kredisi başvurusu yapabilmelidir.
- **Kart Yönetimi:** Kullanıcılar kartlarını yönetebilmeli, kart limitlerini artırabilmeli ya da geçici olarak kartlarını kapatabilmelidir.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **Fonksiyonel Olmayan Gereksinimler:**

- **Güvenlik:** Uygulama, yüksek güvenlik standartlarına sahip olmalı, veri şifreleme ve iki faktörlü kimlik doğrulama kullanmalıdır.
- **Performans:** Uygulama hızlı çalışmalı ve düşük internet bağlantısı altında bile etkin olmalıdır.
- **Kullanıcı Deneyimi (UX):** Arayüz basit ve kullanıcı dostu olmalı; her yaştan insan kolayca kullanabilmelidir.
- **Platform Uyumluluğu:** Uygulama, hem iOS hem de Android cihazlarda sorunsuz çalışmalıdır.
- **Veri Şifreleme:** Hem veri iletiminde hem de depolamada, müşteri bilgileri şifreli bir şekilde korunmalıdır.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **Gereksinimlerin Analizi ve Önceliklendirilmesi (Requirement Analysis and Prioritization)**
- ✓ Gereksinimler analiz edilir ve işlevselliklerine göre önceliklendirilir. Banka için kritik olan işlemler önceliklendirilir ve müşteri memnuniyeti artıracak işlevler sonraki aşamalara bırakılabilir.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **Kritik Gereksinimler:**

- Kullanıcı kimlik doğrulama (giriş işlemleri, biyometrik doğrulama).
- Hesap görüntüleme ve bakiye kontrolü.
- Para transferi (EFT ve havale).
- Güvenlik ve şifreleme özellikleri.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **Orta Öncelikli Gereksinimler:**

- Fatura ödemeleri ve kredi kartı borcu ödeme.
- Kart yönetimi (kart kapama, limit artırma).
- Müşteri hizmetlerine kolay erişim (canlı destek, çağrı merkezi entegrasyonu).

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **Düşük Öncelikli Gereksinimler:**

- Kredi başvurusu.
- Kampanya ve promosyonların görüntülenmesi.
- Yatırım işlemleri (döviz, altın alım/satım).

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

✓ **4. Gereksinimlerin Belgelendirilmesi (Requirement Documentation)**

✓ Toplanan ve analiz edilen gereksinimler yazılı hale getirilir. Bu belgeler, hem geliştirme sürecinde hem de projede yer alan paydaşlar arasında bir referans noktası olacaktır.

- **Gereksinim Özellik Belgesi (SRS):**
  - Kimlik doğrulama işlemlerinden güvenlik standartlarına kadar tüm işlevler detaylı şekilde açıklanır.
  - Örneğin, "Müşteriler, mobil uygulamaya giriş yaptığında iki faktörlü kimlik doğrulama zorunlu olacak, ayrıca şifreleme protokolleri (AES 256-bit) ile veri güvenliği sağlanacak."
- **Kullanıcı Hikayeleri (User Stories):**
  - "Bir müşteri olarak, banka hesaplarımı hızlıca görüntüleyebilmek istiyorum."
  - "Bir kullanıcı olarak, faturalarımı güvenli bir şekilde mobil uygulama üzerinden ödemek istiyorum."

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- ✓ **5. Gereksinimlerin Onaylanması (Requirement Validation)**
- ✓ Bu aşamada, gereksinimler paydaşlar tarafından gözden geçirilir ve onaylanır. Gereksinimlerin doğruluğu ve eksiksizliği bu aşamada netleştirilir. Müşteriler ve banka yönetimi ile yapılan toplantılarda, uygulamanın nasıl çalışacağına dair geri bildirimler alınır.
- **Prototip Onayı:** İlk prototip oluşturulur ve paydaşlara sunulur. Kullanıcı arayüzü ve kritik işlevler hakkında geri bildirimler alınır. Kullanıcıların uygulamayı nasıl kullandığına dair gözlemler yapılır.

Örnek 2 - Proje: Mobil Bankacılık Uygulaması Gereksinim Analizi

- **6. Gereksinim Değişiklikleri ve Geri Bildirimler:**
- Geri bildirimler sonucunda, örneğin bankanın bazı güvenlik politikaları ya da kullanıcı deneyimi iyileştirmeleri ile ilgili eklemeler yapılabilir.

- Örneğin, giriş işlemlerinde biyometrik kimlik doğrulamanın daha kullanışlı olması için "parmak izi" girişine ek olarak "yüz tanıma" özelliği eklenebilir.

LAB Ödevi

- ✓ Üniversite Öğrenci Bilgi Sistemi (ÖBS) Gereksinim Analizi

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

2. Fizibilite Çalışması (Feasibility Study)

- ✓ Fizibilite çalışması, bir projenin ya da girişimin uygulanabilir olup olmadığını, yani başarıya ulaşıp ulaşamayacağını değerlendiren kapsamlı bir analizdir.
- ✓ Bu çalışma, bir projenin maliyetleri, riskleri, zaman çizelgesi, teknik gereksinimleri ve potansiyel getirilerini detaylı bir şekilde inceleyerek, projenin başarılı olup olamayacağını öngörmeye çalışır.
- ✓ Fizibilite çalışması, genellikle proje başlamadan önce yapılır ve karar vericilere projenin devam edip etmeyeceği konusunda rehberlik eder.

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **1. Teknik Fizibilite**
- ✓ Teknik fizibilite, projenin teknik açıdan uygulanabilir olup olmadığını değerlendiren aşamadır. Bu bölümde projenin teknik gereksinimleri incelenir ve mevcut teknolojilerle bu gereksinimlerin karşılanıp karşılanamayacağı araştırılır. Teknik fizibilite, bir projenin başarılı olabilmesi için gerekli olan yazılım, donanım, altyapı ve uzmanlık gibi faktörleri değerlendirir.
- ✓ **Öne Çıkan Unsurlar:**
- **Mevcut Teknolojiler:** Proje için gereken yazılım, donanım veya diğer teknolojik altyapılar mevcut mu? Bu teknolojiler projeye uygun mu?
- **Teknik Uzmanlık:** Projeyi gerçekleştirmek için gerekli teknik bilgiye ve uzmanlığa sahip misiniz? Ekibinizin teknik becerileri yeterli mi?
- **Teknik Zorluklar:** Projede karşılaşılabilecek teknik zorluklar veya engeller neler? Bu zorluklar aşılabilir mi?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **2. Ekonomik Fizibilite (Finansal Fizibilite)**
- ✓ Ekonomik fizibilite, projenin finansal açıdan sürdürülebilir olup olmadığını belirler. Bu aşamada, proje için gerekli maliyetler, beklenen kazançlar ve yatırımın geri dönüş süresi (ROI) hesaplanır. Projenin finansal getirileri, yatırılan sermayeyi ne kadar sürede geri kazandıracak? Proje, uzun vadede kâr sağlayacak mı?
- ✓ **Öne Çıkan Unsurlar:**
- **Proje Maliyetleri:** Projenin başlangıç maliyeti nedir? Geliştirme sürecinde ne kadar para harcanacak? Maliyetler, donanım, yazılım, iş gücü, altyapı ve diğer giderleri kapsar.
- **Yatırım Getirisi (ROI):** Projeden beklenen gelir nedir? Proje, yapılan harcamaları ne kadar sürede karşılayacak?
- **Finansal Riskler:** Projenin karşılaşabileceği finansal riskler nelerdir? Gelir beklenen düzeyde olmazsa ne gibi sonuçlarla karşılaşılabilir?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **3. Operasyonel Fizibilite**
- ✓ Operasyonel fizibilite, projenin günlük operasyonlar açısından uygulanabilirliğini inceleyen aşamadır. Bu aşama, projenin organizasyonel yapıya nasıl entegre edileceğini, iş süreçleri üzerindeki etkisini ve operasyonel riskleri değerlendirir.
- ✓ **Öne Çıkan Unsurlar:**
- **İş Süreçlerine Uyum:** Proje mevcut iş süreçlerine nasıl entegre edilecek? Operasyonel süreçlerde ne gibi değişiklikler gerekecek?
- **Kaynak ve Kapasite:** Mevcut kaynaklar (personel, malzeme, altyapı) projeyi yürütmek için yeterli mi? Yeni kaynaklara ihtiyaç var mı?
- **Organizasyonel Riskler:** Proje, organizasyonel yapıda ne gibi zorluklar çıkarabilir? Mevcut iş gücü ve operasyonel kapasite projeyi kaldırabilecek mi?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **4. Zaman (Zamanlama) Fizibilitesi**
- ✓ Zaman fizibilitesi, projenin belirlenen zaman çizelgesine uygun bir şekilde tamamlanıp tamamlanamayacağını değerlendirir. Projenin zamanında tamamlanabilmesi için gereken aşamalar ve bu aşamaların süresi analiz edilir.
- ✓ **Öne Çıkan Unsurlar:**
- **Zaman Çizelgesi:** Proje için belirlenen takvim gerçekçi mi? Proje belirlenen sürede tamamlanabilir mi?
- **Kritik Yol Analizi:** Proje için gerekli olan tüm faaliyetler ve bu faaliyetlerin birbiriyle olan bağımlılıkları nedir? En kritik faaliyetler hangileridir?
- **Zaman Riskleri:** Projenin gecikmesine neden olabilecek faktörler nelerdir? Bu gecikmeler nasıl yönetilecek?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **5. Yasal ve Düzenleyici Fizibilite**
- ✓ Yasal ve düzenleyici fizibilite, projenin yürütülmesi sırasında karşılaşılabilecek yasal engelleri, düzenleyici gereklilikleri ve mevzuat uyumunu değerlendiren aşamadır. Proje yasal mevzuatlara uygun mu? Gerekli izinler ve lisanslar alınabilecek mi?
- ✓ **Öne Çıkan Unsurlar:**
- **Mevzuat ve Düzenlemeler:** Proje, yerel ve ulusal mevzuatlara uygun mu? Yasal gereklilikler karşılanacak mı?
- **İzinler ve Lisanslar:** Proje için gerekli olan resmi izinler alınabilecek mi?
- **Sözleşme ve Hukuki Riskler:** Proje boyunca karşılaşılabilecek yasal riskler nelerdir? Bu riskler nasıl yönetilecek?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **6. Pazar Fizibilitesi**
- ✓ Pazar fizibilitesi, projenin hedef pazarına yönelik potansiyelini ve başarı şansını değerlendirir. Bu analizde, pazardaki talep, rekabet durumu, müşteri profili ve pazar büyüklüğü gibi unsurlar ele alınır.
- ✓ **Öne Çıkan Unsurlar:**
- **Talep Analizi:** Projeye yönelik pazar talebi nedir? Pazar büyümesi nasıl görünüyor?
- **Rakip Analizi:** Pazarın mevcut rakipleri kimler? Rakipler karşısında projenin avantajları ve dezavantajları neler?
- **Müşteri Profili:** Projenin hedef kitlesi kim? Bu müşteri grubunun projeye ilgisi nasıl olacak?

2. Fizibilite Çalışması (Feasibility Study)

- ✓ **7. Risk Analizi**
- ✓ Risk analizi, projenin başarıya ulaşmasını engelleyebilecek potansiyel riskleri belirler. Bu riskler, finansal, teknik, yasal veya operasyonel olabilir. Risk analizi, bu risklerin nasıl yönetileceği ve minimize edileceği konusunda stratejiler geliştirir.
- ✓ **Öne Çıkan Unsurlar:**
- **Risk Türleri:** Projenin hangi aşamalarında riskler var? Teknik, finansal, operasyonel veya yasal riskler neler?
- **Risk Yönetimi Stratejileri:** Bu riskler nasıl önlenebilir veya minimize edilebilir?

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ Bir girişimci, çevrimiçi satış yapabileceği bir e-ticaret platformu kurmayı planlıyor.
- ✓ Fizibilite çalışması yapılırken teknik, ekonomik, operasyonel, yasal, zaman ve pazar fizibiliteleri detaylıca incelenecektir.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

✓ **1. Teknik Fizibilite**

✓ Bu bölümde, projenin teknik açıdan uygulanabilir olup olmadığı değerlendirilecektir. E-ticaret platformu kurmak, birçok teknik altyapıyı gerektirir. Öne çıkan unsurlar şunlardır:

- **Mevcut Teknoloji ve Altyapı:** Girişimci, platformu oluşturmak için gerekli olan yazılım ve donanımları temin edebilecek mi? Bulut tabanlı altyapı mı kullanılacak, yoksa sunucular şirket içinde mi barındırılacak? Teknik altyapı maliyetleri ve gereklilikleri değerlendirilecek.
- **E-Ticaret Yazılımı:** Platformu oluşturmak için bir hazır e-ticaret yazılımı (Shopify, WooCommerce gibi) mı kullanılacak yoksa sıfırdan bir yazılım mı geliştirilecek? Hazır çözümler hızlıca devreye alınabilirken, özelleştirilmiş çözümler daha esnek ve uzun vadeli olabilir. Bu karar, teknik gereksinimlere göre alınacaktır.
- **Teknik Destek ve Bakım:** Platform devreye alındıktan sonra teknik bakım nasıl yapılacak? Site kesintileri, siber saldırılar veya teknik arızalar durumunda hızlı bir müdahale ekibi olacak mı?
- **Ölçeklenebilirlik:** Platform büyüdükçe daha fazla müşteriye hizmet edebilmek için ölçeklenebilir bir altyapı kullanmak gerekiyor. Teknik analizde, platformun nasıl büyüyeceği ve trafiği nasıl kaldıracağı değerlendirilmeli.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- **2. Ekonomik Fizibilite**
- Ekonomik fizibilite, projenin maliyet ve getirilerini analiz eder. Bu aşamada, başlangıç maliyetleri, işletme giderleri ve potansiyel gelirler değerlendirilir.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **Başlangıç Maliyetleri:**
- **Altyapı ve Teknoloji Yatırımları:** Sunucular, yazılım lisansları, domain ve SSL sertifikası gibi teknoloji maliyetleri. Eğer sıfırdan yazılım geliştirilecekse, yazılım ekibi masrafları.
- **Pazarlama ve Reklam:** Marka bilinirliğini artırmak için SEO, Google Ads, sosyal medya reklamları gibi dijital pazarlama harcamaları.
- **Lojistik ve Depo Maliyetleri:** Ürünlerin depolanması, paketlenmesi ve müşteriye ulaştırılması için gerekli lojistik altyapı maliyetleri.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **İşletme Giderleri:**
- **Personel Giderleri:** Yazılımcı, pazarlama ekibi, müşteri hizmetleri gibi personele ödenecek maaşlar.
- **Teknik Destek:** Web sitesinin bakımı, güvenlik güncellemeleri ve müşteri hizmetleri giderleri.
- ✓ **Yatırım Getirisi (ROI):**
- Yapılan yatırımların ne kadar sürede geri kazanılacağı hesaplanmalıdır. Örneğin, ilk 6 ayda ne kadar satış bekleniyor, bu satışların kârlılığı nedir? Platformun ortalama 1 yıl içinde kâra geçmesi hedefleniyor olabilir.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **3. Operasyonel Fizibilite**
- ✓ Bu aşamada, e-ticaret platformunun günlük operasyonel süreçlere nasıl entegre edileceği ve işletme sürecinin nasıl yönetileceği değerlendirilir.
- **İş Süreçleri:** E-ticaret platformunun mevcut iş süreçlerine entegrasyonu nasıl olacak? Lojistik, depolama, stok yönetimi gibi süreçler nasıl işletilecek? Ürün tedariği, stok yönetimi ve müşteri hizmetleri süreçleri değerlendirilmelidir.
- **Müşteri Hizmetleri:** Müşterilerden gelecek sorulara ve şikayetlere hızlı bir şekilde yanıt verecek bir müşteri hizmetleri ekibi kurmak gerekir. Online siparişlerde yaşanabilecek sorunları çözmek için bir destek sistemi oluşturulmalı.
- **Lojistik ve Dağıtım:** Ürünlerin depolanması, paketlenmesi ve teslimat süreçleri nasıl yönetilecek? Lojistik maliyetleri minimize edilebilir mi? Ürünlerin müşterilere ne kadar sürede ulaştırılacağı gibi operasyonel detaylar incelenmeli.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **4. Yasal ve Düzenleyici Fizibilite**
- ✓ Yasal ve düzenleyici fizibilite, e-ticaret platformunun yasal gereksinimlerini ve düzenlemelere uygunluğunu değerlendirir.
- **Vergilendirme ve Lisanslar:** E-ticaret platformu hangi yasalara tabi olacak? İnternetten satış yaparken vergi yükümlülükleri neler? Şirketin kurulumu ve ürün satışları için gerekli lisanslar temin edilecek mi?
- **Tüketici Hakları:** E-ticaret platformu tüketici hakları yasalarına uygun olmalı. İade ve değişim politikaları nasıl belirlenecek? Müşterilerle olan sözleşmelerde yasal gereksinimler karşılanacak mı?
- **Veri Gizliliği:** Müşteri bilgilerini toplarken GDPR veya KVKK gibi veri koruma yasalarına uyulacak mı? Müşteri verilerinin güvenliği nasıl sağlanacak?

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **5. Zaman Fizibilitesi**
- ✓ Projenin zaman çizelgesi, hedeflenen tarihlerde tamamlanıp tamamlanamayacağını inceler.
- **Zaman Çizelgesi:** E-ticaret platformunun hayata geçirilmesi için gereken adımlar ve süreler belirlenir. Platformun geliştirilmesi, test edilmesi ve canlıya alınması için ne kadar süre gerekecek? Projenin belirlenen zaman diliminde tamamlanması gerçekçi mi?
- **Kritik Yol Analizi:** Projede hangi aşamaların tamamlanması zorunlu ve hangileri ertelenebilir? Örneğin, platformun ön yüzü geliştirildikten sonra ödeme sistemleri entegre edilebilir.

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **6. Pazar Fizibilitesi**
- ✓ Pazar fizibilitesi, platformun hedef kitlesine ve pazar dinamiklerine uygun olup olmadığını inceler.
- **Talep Analizi:** E-ticaret platformunun hedef kitlesi kimlerdir? Hangi ürünlerin satılacağı belirlendi mi? Bu ürünlere pazar talebi var mı? E-ticaret sektörünün büyüme potansiyeli nedir?
- **Rakip Analizi:** Hedef pazarda mevcut rakipler kimlerdir? Rakiplerle rekabet edebilmek için platformun ne gibi özellikler sunması gerekiyor? Fiyatlandırma, ürün çeşitliliği ve müşteri hizmetleri açısından rekabet avantajları değerlendirilmeli.
- **Müşteri Profili:** Hedef kitlenin demografik özellikleri, alışveriş alışkanlıkları ve online harcama eğilimleri nedir? Platform, bu kitlenin ihtiyaçlarına nasıl cevap verecek?

Örnek- Fizibilite Çalışması
Proje: E-Ticaret Platformu Kurulumu

- ✓ **7. Risk Analizi**
- ✓ Risk analizi, projenin başarılı olma şansını etkileyebilecek potansiyel riskleri değerlendirmeye odaklanır.
- **Teknik Riskler:** Platformda teknik aksaklıklar (çökmeler, hatalar, güvenlik açıkları) yaşanabilir mi? Bu riskler nasıl yönetilecek?
- **Finansal Riskler:** Beklenen satış hedeflerine ulaşılmazsa, projenin finansal olarak sürdürülebilirliği nasıl sağlanacak? Yatırımın geri dönüşü gecikirse ne gibi tedbirler alınabilir?
- **Pazar Riskleri:** Pazar koşullarındaki değişiklikler, talebin düşmesi veya yeni rakiplerin ortaya çıkması gibi durumlar, projenin başarısını etkileyebilir mi?

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ Bir enerji firması, çevre dostu ve sürdürülebilir enerji üretimi sağlamak amacıyla bir güneş enerjisi santrali kurmayı planlıyor.

- ✓ Bu projeyi hayata geçirmeden önce teknik, ekonomik, operasyonel, yasal, zaman ve pazar fizibilitesi analiz edilerek projenin uygulanabilirliği değerlendirilecektir.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **1. Teknik Fizibilite**
- ✓ Bu bölümde, santralin kurulması için gerekli teknolojik altyapı, mühendislik çözümleri ve teknik detaylar incelenir.
- **Mevcut Teknoloji:** Güneş panelleri ve enerji dönüşüm teknolojilerinin güncel durumu değerlendirilir. Hangi tür güneş panellerinin kullanılacağı ve bunların verimlilik oranları incelenir. Yüksek verimli paneller mi kullanılacak, yoksa maliyetleri düşürmek için daha uygun fiyatlı seçenekler mi tercih edilecek?
- **Yer Seçimi:** Santralin kurulacağı bölgenin güneşlenme oranı, iklim koşulları ve toprak yapısı analiz edilir. Güneş ışığından maksimum verimi almak için en ideal bölge seçilir. Ayrıca, yer seçiminin enerji iletim hatlarına yakın olması da maliyet avantajı sağlayabilir.
- **Teknik Altyapı:** Güneş enerjisi üretimi için kullanılacak inverterler, trafo merkezleri, enerji depolama çözümleri gibi altyapılar belirlenir. Ayrıca santralin bağlantı noktaları ve elektrik şebekesi ile entegrasyonu değerlendirilir.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **2. Ekonomik Fizibilite**
- ✓ Güneş enerjisi santrali kurulumunun finansal açıdan sürdürülebilir olup olmadığı incelenir.
- ✓ Başlangıç maliyetleri, işletme maliyetleri ve yatırımın geri dönüş süresi (ROI) hesaplanır.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **Başlangıç Maliyetleri:**
- **Güneş Panelleri ve Donanım:** Panellerin satın alım maliyeti, inverterler, enerji iletim hatları ve diğer ekipmanlar için gerekli olan yatırım.
- **İnşaat Maliyetleri:** Santralin kurulacağı alanın hazırlanması, altyapı inşaatı, panellerin kurulumu ve gerekli lojistik maliyetler.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **İşletme Giderleri:**
- **Bakım ve Onarım:** Güneş panellerinin ve diğer ekipmanların düzenli bakımı, temizliği ve olası arızaların giderilmesi.
- **Personel Giderleri:** Santral operasyonlarını yönetecek mühendisler, teknikerler ve güvenlik personeli için ödenmesi gereken maaşlar.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **3. Operasyonel Fizibilite**
- ✓ Güneş enerjisi santralinin günlük operasyonel süreçlerinin yönetimi ve teknik detayları analiz edilir.
- **İş Süreçleri:** Santralin operasyonel süreçleri nasıl yönetilecek? Santralin kurulumu tamamlandıktan sonra enerji üretim süreci nasıl devam edecek? Sistemlerin otomasyonu için gerekli olan yazılım ve kontrol sistemleri var mı?
- **Enerji Depolama:** Santralin ürettiği enerjiyi depolamak için batarya sistemleri kullanılacak mı? Depolama sistemleri, gece ve bulutlu günlerde enerji arzını sürdürülebilir kılmak için kritik öneme sahiptir.
- **Enerji İletim:** Üretilen enerjinin elektrik şebekesine nasıl entegre edileceği ve iletim hattı kapasitesi analiz edilir. Enerji iletim hattının döşenmesi için gerekli inşaat ve altyapı düzenlemeleri gözden geçirilir.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **4. Yasal ve Düzenleyici Fizibilite**
- ✓ Yasal düzenlemeler, izin süreçleri ve çevre mevzuatlarına uygunluk bu aşamada değerlendirilir.
- **Lisanslar ve İzinler:** Güneş enerjisi santrali kurmak için gerekli olan enerji üretim lisansları, çevre düzenlemeleri ve yerel yönetimlerden alınacak izinler incelenir. Örneğin, santral için enerji düzenleme kurulu tarafından verilen lisansların süreci ve maliyeti değerlendirilmelidir.
- **Çevresel Etki Değerlendirmesi (ÇED):** Güneş enerjisi santralinin çevresel etkileri değerlendirilir. Toprağın korunması, bölgedeki ekosistemin etkilenmemesi gibi faktörler dikkate alınır. Çevresel etkileri en aza indirgemek için gerekli önlemler planlanır.
- **Yasal Uyum:** Santralin işletimi sırasında çevre mevzuatlarına, iş güvenliği yasalarına ve enerji sektörüne yönelik düzenlemelere uyum sağlanmalıdır. Ayrıca, enerji üretimi sırasında oluşabilecek yasal riskler değerlendirilir.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **5. Zaman Fizibilitesi**
- ✓ Santralin inşası ve işletime alınması için gereken süre hesaplanır ve zaman çizelgesi oluşturulur.
- **İnşaat Zaman Çizelgesi:** Güneş panellerinin kurulumu, altyapının inşası ve bağlantı hatlarının döşenmesi ne kadar sürecek? Projenin baştan sona tamamlanma süresi 12 ay olarak öngörülüyor, bu süreçte karşılaşılabilecek olası gecikmeler analiz edilir.
- **Proje Yönetimi:** İnşaat sürecinde kritik aşamalar ve bu aşamalarda yaşanabilecek gecikmelerin projeyi nasıl etkileyeceği belirlenir. Örneğin, panellerin teslimatında yaşanacak bir gecikme, tüm süreci uzatabilir.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **6. Pazar Fizibilitesi**
- ✓ Güneş enerjisi santralinin üreteceği enerjinin pazarda talep görüp görmeyeceği ve pazardaki rekabet durumu değerlendirilir.
- **Enerji Talebi:** Güneş enerjisine olan talep analiz edilir. Bölgedeki enerji tüketimi artış oranları, sanayi ve konut sektörlerinin enerji ihtiyaçları gözden geçirilir. Özellikle yenilenebilir enerjiye olan talep ve devlet teşvikleri bu kararı etkileyebilir.
- **Rekabet Analizi:** Aynı bölgede başka güneş enerjisi santralleri var mı? Diğer yenilenebilir enerji kaynaklarıyla rekabet nasıl olacak? Pazar analizi, rekabet avantajları ve dezavantajlarını ortaya koyacaktır.
- **Fiyatlandırma:** Güneş enerjisinin mevcut piyasa fiyatı nedir? Devletin yenilenebilir enerji için sağladığı teşvikler ve alım garantileri gibi faktörler de göz önünde bulundurulmalıdır.

Örnek 2 - Fizibilite Çalışması
Proje: Güneş Enerjisi Santrali Kurulumu

- ✓ **7. Risk Analizi**
- ✓ Güneş enerjisi santralinin başarılı olmasını engelleyebilecek olası riskler değerlendirilir.
- **Finansal Riskler:** Proje finansmanında bir sorun çıkarsa, inşaat maliyetlerinin artması gibi beklenmeyen durumlar oluşursa, projeyi sürdürebilmek için alternatif finansman kaynakları sağlanabilecek mi?
- **Teknik Riskler:** Güneş panellerinin teknik sorunlar yaşaması, üretim hatalarının oluşması veya arızaların giderilmesindeki zorluklar nasıl çözülecek?
- **Çevresel Riskler:** İklim değişiklikleri, bölgedeki güneşlenme oranında düşüşe neden olabilir mi? Bu durumda üretim kapasitesinde düşüş yaşanması riski var mı?

LAB Ödevi

- ✓ Üniversite Öğrenci Bilgi Sistemi (ÖBS) Fizibilite Çalışmasını Değerlendirin

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Sistem Tasarımı

- **Sistem Tasarımı** adımı, yazılım yaşam döngüsünde kritik bir rol oynar ve genel olarak, geliştirilecek yazılımın mimarisini ve yapısını belirleme sürecidir.

- Bu adım, yazılımın nasıl çalışacağını ve hangi bileşenlerden oluşacağını detaylı bir şekilde tanımlayarak projenin doğru ve verimli bir şekilde ilerlemesini sağlar.

Sistem Tasarımı

- ✓ **Sistem Tasarımının Önemi**
- **Doğru planlama**: Sistem tasarımı aşaması, yazılımın nasıl çalışacağına dair doğru planlamalar yapılmasını sağlar ve sonradan ortaya çıkabilecek problemlerin önüne geçer.
- **Maliyet ve zaman yönetimi**: İyi bir tasarım, geliştirme sürecinde hataların azaltılmasına ve bu sayede maliyet ve zaman kayıplarının önlenmesine yardımcı olur.
- **Geleceğe yönelik genişletilebilirlik**: Tasarım aşamasında yapılan doğru seçimler, yazılımın ileride genişletilmesini veya modifiye edilmesini kolaylaştırır.
- ✓ Sonuç olarak, sistem tasarımı, yazılımın başarılı bir şekilde geliştirilmesinin temelini oluşturan bir adım olup, yazılımın kalite, sürdürülebilirlik ve başarısı açısından kritik bir role sahiptir.

Sistem Tasarımı

- Sistem tasarımı iki ana seviyede gerçekleştirilir:
- **1. Yüksek Seviyeli Tasarım (High-Level Design - HLD)**
- Bu aşama, yazılımın genel mimarisini, büyük bileşenlerin nasıl etkileşimde bulunacağını ve sistemin üst düzey yapılarını belirler.

- **2. Detaylı Tasarım (Low-Level Design - LLD)**
- Bu aşama, yüksek seviyeli tasarımın daha ayrıntılı hale getirildiği bölümdür. Her bir modülün iç yapısı, algoritmaları, veri yapıları, sınıfları ve işlevleri tanımlanır.

Sistem Tasarımı

- ✓ **1. Yüksek Seviyeli Tasarım (High-Level Design - HLD)**
- ✓ Bu tasarım aşamasında genel olarak şu unsurlar ele alınır:
  - **Modüler yapı**: Yazılımın hangi ana modüllerden oluşacağı belirlenir.
  - **Sistem mimarisi**: Sistemin hangi donanım ve yazılım bileşenleri üzerinde çalışacağı, kullanılan teknoloji yığınları tanımlanır.
  - **Veri akışı**: Modüller arasındaki veri akışı ve etkileşimler belirlenir.
  - **Dış sistemlerle entegrasyon**: Yazılımın diğer sistemlerle nasıl entegre olacağı tanımlanır.
- ✓ Bu adım, proje ekiplerine genel bir bakış sunarak sistemin karmaşıklığını anlamalarına yardımcı olur. Genelde yazılım mimarları ve kıdemli mühendisler bu aşamada rol oynar.

Sistem Tasarımı

- ✓ **2. Detaylı Tasarım (Low-Level Design - LLD)**
- ✓ Bu aşama, yüksek seviyeli tasarımın daha ayrıntılı hale getirildiği bölümdür. Her bir modülün iç yapısı, algoritmaları, veri yapıları, sınıfları ve işlevleri tanımlanır. Şu detaylar üzerinde durulur:
- **Veritabanı tasarımı**: Tablolar, ilişkiler ve veri yapıları detaylandırılır.
- **Algoritmalar**: Her modülün içindeki işlemlerin nasıl yapılacağı tanımlanır.
- **Arayüzler ve API'ler**: Modüller arası iletişim için kullanılacak arayüzler belirlenir.
- **Hata yönetimi**: Yazılımda olası hata durumları nasıl ele alınacak, belirlenir.

Sistem Tasarımı

- ✓ 1. **Modül Tasarımı (Module Design)**
- ✓ Her modülün iç yapısı detaylandırılır ve hangi sorumluluklara sahip olduğu belirlenir. Modülün işleyişi, diğer modüllerle nasıl etkileşimde bulunacağı ve veri akışını nasıl yöneteceği tanımlanır.
- **Modül Sorumlulukları**: Her bir modülün spesifik bir görevi vardır. Bu modül örneğin bir kullanıcı giriş sistemini, veri yönetimini veya raporlama süreçlerini yönetebilir.
- **Modüller Arası İlişkiler**: Modüller arasındaki bağımlılıklar, veri akışları ve etkileşimler detaylandırılır. Hangi modüllerin birbirleriyle nasıl iletişim kuracakları açıklanır.

Sistem Tasarımı

- ✓ 1. **Modül Tasarımı (Module Design)**
- ✓ Her modülün iç yapısı detaylandırılır ve hangi sorumluluklara sahip olduğu belirlenir. Modülün işleyişi, diğer modüllerle nasıl etkileşimde bulunacağı ve veri akışını nasıl yöneteceği tanımlanır.
- **Modül Sorumlulukları**: Her bir modülün spesifik bir görevi vardır. Bu modül örneğin bir kullanıcı giriş sistemini, veri yönetimini veya raporlama süreçlerini yönetebilir.
- **Modüller Arası İlişkiler**: Modüller arasındaki bağımlılıklar, veri akışları ve etkileşimler detaylandırılır. Hangi modüllerin birbirleriyle nasıl iletişim kuracakları açıklanır.

Sistem Tasarımı

- ✓ **2. Sınıf Diyagramları (Class Diagrams)**
- ✓ Sınıf diyagramları, yazılımın hangi sınıfları ve nesneleri içereceğini, bu sınıfların birbirleriyle nasıl ilişkili olacağını ve sınıfların hangi özellik ve yöntemlere sahip olacağını gösterir. Nesneye dayalı programlama (OOP) metodolojisine dayalı olarak sınıflar şu şekilde detaylandırılır:
- **Sınıf İsimleri**: Her sınıfın adı ve sorumluluğu tanımlanır.
- **Nitelikler ve Metodlar**: Her sınıfın hangi özelliklere (nitelikler/attributes) ve hangi işlemlere (metodlar/methods) sahip olduğu belirtilir.
- **İlişkiler**: Sınıflar arasındaki ilişkiler (miras alma, bileşim, birleşim gibi) ve bu sınıfların birbirleriyle nasıl etkileşimde bulunduğu gösterilir.

Sistem Tasarımı

- ✓ **3. Algoritma ve Akış Diyagramları (Algorithm and Flow Diagrams)**
- ✓ Yazılımın belirli işlemleri nasıl gerçekleştireceği ve adım adım nasıl ilerleyeceği, algoritmalar ve akış diyagramlarıyla gösterilir. Bu aşamada:
- **Algoritmalar**: Modüllerin ya da fonksiyonların belirli bir görevi yerine getirmesi için kullanılacak adımlar, algoritmalar şeklinde tanımlanır. Örneğin, bir sıralama işlemi ya da veri doğrulama işlemi için algoritmalar belirlenir.
- **Akış Diyagramları**: Algoritmaların nasıl işlediğini göstermek için adım adım veri akışını ve işlem adımlarını gösteren diyagramlar kullanılır. Bu diyagramlar, geliştiricilerin süreci daha kolay anlamasını sağlar.

Sistem Tasarımı

- ✓ **4. Veri Yapıları (Data Structures)**
- ✓ Veri yapıları, yazılımda hangi veri türlerinin kullanılacağı ve bu verilerin nasıl organize edileceği ile ilgilidir. Detaylı tasarımda, kullanılan veri yapılarının türleri, ilişkileri ve yapıları tanımlanır:
- **Diziler, listeler, kümeler ve haritalar**: Kullanılacak temel veri yapılarının hangi durumlarda kullanılacağı belirlenir.
- **Veri ilişkileri**: Verilerin birbiriyle nasıl ilişkili olduğu ve hangi yapıların veriyi depolamak için en uygun olduğu açıklanır.

Sistem Tasarımı

- ✓ **5. Veritabanı Tasarımı (Database Design)**
- ✓ Detaylı tasarımın önemli bir kısmı da veritabanı tasarımıdır. Veritabanı yapıları, tablolar ve veri modelleri detaylandırılır:
- **Tablolar**: Hangi verilerin hangi tablolarda saklanacağı, bu tabloların alanları ve alan türleri belirlenir.
- **İlişkiler**: Tablolar arasındaki ilişkiler (birden bire, birden çoğa vb.) detaylı şekilde gösterilir. Bu ilişkiler, normalizasyon işlemleri ve veri tutarlılığı göz önünde bulundurularak düzenlenir.
- **SQL veya NoSQL kullanımı**: İhtiyaca göre hangi veritabanı türünün kullanılacağı, verilerin nasıl sorgulanacağı ve nasıl işleneceği tasarlanır.

Sistem Tasarımı

- ✓ **6. Arayüzler ve API'ler (Interfaces and APIs)**
- ✓ Modüller ve bileşenler arasındaki etkileşimi sağlamak için kullanılacak arayüzler ve API'ler belirlenir. Bu aşamada şunlar planlanır:
- **Fonksiyonlar ve Parametreler**: Her bir arayüzün hangi işlevleri sağlayacağı, hangi parametrelerle çağrılacağı belirlenir.
- **Giriş ve Çıkışlar**: Her bir fonksiyonun ya da API'nin aldığı girdiler ve sağladığı çıktılar detaylandırılır.
- **API Protokolleri**: REST, GraphQL veya SOAP gibi API'lerin kullanılacağı protokoller belirlenir.

Sistem Tasarımı

- ✓ **7. Hata Yönetimi (Error Handling)**
- ✓ Detaylı tasarım aşamasında, yazılımda karşılaşılabilecek hata durumlarında ne yapılacağı ve hataların nasıl ele alınacağı da belirlenir:
- **Hata Kodları**: Yazılımın karşılaşabileceği hata türleri ve bu hatalar için kullanılacak hata kodları (örn. HTTP 404, 500 hataları) belirlenir.
- **Hata Yakalama Mekanizmaları**: Hataların nasıl yakalanacağı, kayıt altına alınacağı ve yönetileceği (örneğin try-catch blokları) tanımlanır.
- **Hata Loglama**: Hataların takip edilmesi ve incelenebilmesi için loglama (kayıt tutma) mekanizmaları oluşturulur.

Sistem Tasarımı

- ✓ **8. Performans Optimizasyonu (Performance Optimization)**
- ✓ Yazılımın performansının optimize edilmesi için hangi stratejilerin uygulanacağı detaylı şekilde planlanır:
- **Önbellekleme (Caching)**: Hangi verilerin, hangi işlemlerin önbelleğe alınacağı ve hangi önbellekleme yöntemlerinin kullanılacağı (örn. Redis, Memcached) belirlenir.
- **Veri Yoğunluklu İşlemler**: Yüksek veri akışına sahip işlemlerde performansın nasıl optimize edileceği, örneğin indeksleme, sorgu optimizasyonu gibi stratejilerle açıklanır.
- **Yük Dengeleme**: Sistem üzerindeki yükün nasıl dengeleneceği ve sistem performansının nasıl sürdürülebileceği planlanır.

Sistem Tasarımı

- ✓ **9. Güvenlik (Security)**
- ✓ Yazılımın güvenliği, detaylı tasarımda teknik seviyede ele alınır. Kullanıcı verilerinin korunması ve sistemin güvenliğinin sağlanması amacıyla şu unsurlar detaylandırılır:
- **Kimlik Doğrulama ve Yetkilendirme**: Kullanıcıların kimlik doğrulama işlemleri (örneğin, JWT, OAuth) ve hangi kaynaklara erişim izni olduğu belirlenir.
- **Şifreleme**: Verilerin saklanması veya transfer edilmesi sırasında hangi şifreleme yöntemlerinin kullanılacağı (örn. AES, RSA, SSL/TLS) açıklanır.
- **Güvenlik Açıkları**: Potansiyel güvenlik açıklarına karşı önlemler (örn. SQL enjeksiyonu, XSS gibi) detaylandırılır.

Sistem Tasarımı

- ✓ **10. Kapsamlı Test Planları (Comprehensive Testing Plans)**
- ✓ Detaylı tasarım, yazılımın nasıl test edileceği ile ilgili ayrıntıları da içerir. Test süreçleri ve stratejileri detaylandırılır:
- **Birim Testleri (Unit Testing)**: Her modül ve fonksiyon için nasıl birim testlerinin yapılacağı ve bu testlerin kapsamı belirlenir.
- **Entegrasyon Testleri (Integration Testing)**: Modüller arası etkileşimlerin nasıl test edileceği ve olası sorunların nasıl önleneceği planlanır.
- **Sistem ve Performans Testleri**: Yazılımın genel performansının ve sistemin genel stabilitesinin nasıl test edileceği açıklanır.

Sistem Tasarımı

- ✓ **11. Kodlama Standartları (Coding Standards)**
- ✓ Yazılımın tutarlı ve sürdürülebilir olması için hangi kodlama standartlarına uyulacağı belirlenir. Bu aşamada:
- **İsimlendirme Kuralları**: Sınıflar, değişkenler, fonksiyonlar ve diğer kod elemanları için isimlendirme standartları belirlenir.
- **Yazım Stili**: Kodun nasıl yazılacağı, hangi format kurallarına uyulacağı (örneğin, girintileme, yorum ekleme) detaylandırılır.
- **Dokümantasyon**: Kodun nasıl dokümante edileceği ve geliştiricilere rehberlik edecek belgelerin nasıl hazırlanacağı planlanır.

Sistem Tasarımı

- ✓ **12. Hata Ayıklama ve İzleme (Debugging and Monitoring)**
- ✓ Sistemin geliştirilmesi ve çalışması sırasında ortaya çıkabilecek sorunları izlemek ve çözmek için kullanılacak hata ayıklama araçları ve izleme yöntemleri belirlenir:
- **Hata Ayıklama Araçları**: Kodun doğru çalışıp çalışmadığını görmek için kullanılacak hata ayıklama araçları belirlenir.
- **Sistem İzleme**: Uygulamanın performansını ve stabilitesini izlemek için kullanılacak izleme araçları (örn. Grafana, Kibana) ve metrikler tanımlanır.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- **1. Yüksek Seviyeli Tasarım (High-Level Design - HLD)**
- Bu aşamada, sistemin genel yapısını ve ana bileşenlerini tasarlıyoruz. Bu, sistemin nasıl yapılandırılacağını ve hangi modüllere ayrılacağını belirlememizi sağlar.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **1. Yüksek Seviyeli Tasarım (High-Level Design - HLD)**
- ✓ Bu aşamada, sistemin genel yapısını ve ana bileşenlerini tasarlıyoruz. Bu, sistemin nasıl yapılandırılacağını ve hangi modüllere ayrılacağını belirlememizi sağlar.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **a) Sistem Mimarisi**
- ✓ Sistem üç ana bileşenden oluşur:

1. **Kullanıcı Arayüzü (UI/UX)**
   1. Web tabanlı olacak ve kullanıcılar bu arayüz üzerinden kitap satın alabilecek, yorum yapabilecek ve hesaplarını yönetebilecekler.
   2. Arayüz modern JavaScript framework'ü (örn. React.js veya Vue.js) ile geliştirilecek.
2. **Sunucu Katmanı (Back-End)**
   1. İş mantığının ve veri işleme süreçlerinin yönetileceği yer olacak.
   2. RESTful API ile kullanıcı talepleri işlenecek. Python Django veya Node.js kullanılabilir.
3. **Veritabanı Katmanı**
   1. Tüm kitap verileri, kullanıcı bilgileri, siparişler ve yorumlar bu katmanda saklanacak.
   2. SQL tabanlı (örn. PostgreSQL veya MySQL) bir veritabanı kullanılacak.
4. **Üçüncü Parti Servisler**
   1. **Ödeme Sistemi**: Stripe veya PayPal gibi üçüncü parti ödeme sağlayıcıları kullanılacak.
   2. **Tavsiye Motoru**: Kitap tavsiyeleri için makine öğrenimi modeli veya dış bir API entegrasyonu yapılacak.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **b) Modüller**
- ✓ Sistemi temel modüllere bölelim:

1. **Kullanıcı Yönetimi Modülü**: Kullanıcı kayıt, giriş, profil düzenleme, şifre sıfırlama işlemleri.
2. **Kitap Yönetimi Modülü**: Kitapları listeleme, arama, detaylarını görüntüleme.
3. **Alışveriş Sepeti ve Satın Alma Modülü**: Kullanıcıların kitapları sepete ekleyip ödeme işlemlerini yapabilmeleri.
4. **Yorum ve Puanlama Modülü**: Kullanıcıların kitaplar hakkında yorum yapıp puan verebilecekleri sistem.
5. **Yönetici Modülü**: Kitap ekleme, stok yönetimi ve siparişleri izleme.
6. **Bildirim ve Tavsiye Modülü**: Tavsiye edilen kitapları ve bildirimleri kullanıcılara gösterme.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **c) Veri Akışı**
- ✓ Kullanıcıların sistemi nasıl kullanacağını bir örnek üzerinden görelim:

1. Kullanıcı siteye giriş yapar.
2. Kitap araması yapar ve bir kitabın detay sayfasına gider.
3. Kitabı sepete ekler ve ödeme adımlarına geçer.
4. Ödeme başarılı olursa, sipariş tamamlanır.
5. Kullanıcı siparişini profili altındaki siparişler bölümünde görüntüler.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- **2. Detaylı Tasarım (Low-Level Design - LLD)**
- Bu aşamada, her modül ve bileşen için daha ayrıntılı bir tasarım yapıyoruz.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

✓ **a) Modül Tasarımı**

✓ Her bir modülün nasıl çalışacağını ve alt modüllerini detaylandırmamız gerekiyor.

1. **Kullanıcı Yönetimi Modülü**
   1. **Kayıt İşlemi**: Kullanıcı formu doldurur, e-posta doğrulaması yapılır.
   2. **Giriş İşlemi**: JWT (JSON Web Token) ile kullanıcı oturumu güvenli şekilde yönetilir.
   3. **Profil Düzenleme**: Kullanıcı kendi bilgilerini güncelleyebilir.
2. **Kitap Yönetimi Modülü**
   1. **Kitap Listeleme**: Tüm kitaplar ana sayfada listelenir. Arama ve filtreleme fonksiyonları ile belirli türlerde kitaplar aranabilir.
   2. **Kitap Detay Sayfası**: Kitap ismi, yazar, fiyat, kullanıcı yorumları ve puanlama bilgileri burada görüntülenir.
3. **Alışveriş Sepeti ve Satın Alma Modülü**
   1. **Sepet Yönetimi**: Kullanıcılar bir veya daha fazla kitabı sepete ekleyebilir. Sepette kitap sayısı güncellenebilir veya kitap silinebilir.
   2. **Ödeme İşlemi**: Kullanıcı Stripe API'si üzerinden ödeme yapar. Başarılı bir ödeme durumunda sipariş veritabanına kaydedilir ve kullanıcıya onay mesajı gönderilir.
4. **Yorum ve Puanlama Modülü**
   1. **Yorum Ekleme**: Kullanıcılar sadece satın aldıkları kitaplara yorum yapabilir.
   2. **Puanlama**: Kitaplar için 1-5 arasında puan verilebilir. Kitabın ortalama puanı görüntülenir.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **b) Sınıf Diyagramları**
- ✓ Sınıf diyagramları, sistemdeki sınıfları ve bunların birbirleriyle ilişkilerini tanımlar. Örnek olarak birkaç sınıf:
- **User**: Kullanıcıyı temsil eden sınıf. Attributeleri: kullanıcı adı, e-posta, şifre, sipariş geçmişi.
- **Book**: Kitap sınıfı. Attributeleri: kitap adı, yazar, tür, fiyat, stok durumu, yorumlar.
- **Order**: Sipariş sınıfı. Attributeleri: sipariş numarası, kullanıcı, kitaplar, toplam tutar, sipariş durumu.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **c) Veritabanı Tasarımı**
- ✓ Veritabanı yapısını tanımlayalım:

1. **Users** tablosu: Kullanıcı bilgilerini içerir (id, ad, soyad, e-posta, şifre).
2. **Books** tablosu: Kitap bilgilerini içerir (id, isim, yazar, fiyat, stok, kategori).
3. **Orders** tablosu: Sipariş bilgilerini içerir (id, kullanıcı_id, toplam_tutar, sipariş_tarihi).
4. **Reviews** tablosu: Yorum ve puanları içerir (id, kitap_id, kullanıcı_id, puan, yorum).

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **d) API Tasarımı**
- ✓ RESTful API ile veritabanı ve arayüz arasındaki veri alışverişini yönetelim. Örnek bazı API uç noktaları:
- **POST /login**: Kullanıcı girişi.
- **GET /books**: Tüm kitapları listele.
- **POST /cart**: Sepete kitap ekle.
- **POST /checkout**: Ödeme yap ve siparişi tamamla.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **e) Algoritma ve Akış Diyagramları**
- ✓ Kullanıcı bir kitap satın aldığında şu adımları izleyen bir algoritma çalışır:

1. Kullanıcı sepetteki kitapları onaylar ve ödeme sayfasına yönlendirilir.
2. Stripe API ile ödeme doğrulanır.
3. Ödeme başarılı ise, sipariş veritabanına kaydedilir ve kullanıcıya e-posta ile onay gönderilir.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

✓ **f) Hata Yönetimi**

- Ödeme başarısız olduğunda kullanıcıya hata mesajı gösterilir.
- Yetersiz stok durumunda sepet güncellenir ve kullanıcıya bildirilir.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **g) Performans Optimizasyonu**
- **Önbellekleme**: Popüler kitaplar Redis kullanılarak önbelleğe alınır.
- **Sorgu Optimizasyonu**: Veritabanı sorguları için indeksleme yapılır.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **g) Performans Optimizasyonu**
- **Önbellekleme**: Popüler kitaplar Redis kullanılarak önbelleğe alınır.
- **Sorgu Optimizasyonu**: Veritabanı sorguları için indeksleme yapılır.

Sistem Tasarımı – Örnek Proje : Online Kitap Satış Platformu

- ✓ **h) Güvenlik**
- Kullanıcı şifreleri şifrelenir ve güvenli bir şekilde saklanır.
- API'ler JWT ile korunur.
- HTTPS kullanılarak veri transferleri güvenli hale getirilir.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- **Proje Hedefi:**
- Kullanıcıların restoranlardan online olarak yemek sipariş verebilecekleri, sipariş sürecini takip edebilecekleri ve restoranların menülerini yönetip siparişleri işleyebilecekleri bir platform geliştirmek.
- **1. Yüksek Seviyeli Tasarım (High-Level Design - HLD)**
- Bu aşamada, sistemin genel yapısı ve ana bileşenlerini tasarlıyoruz.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

✓ **a) Sistem Mimarisi**

✓ Sistem üç ana bileşenden oluşacak:

1. **Kullanıcı Arayüzü (Frontend)**
    1. Kullanıcılar ve restoranlar için web veya mobil arayüz olacak.
    2. Kullanıcı arayüzü yemek siparişi verme, restoran listelerini görme, sepeti yönetme ve siparişi takip etme işlemlerini içerecek.
    3. Restoranlar ise menü yönetimi ve sipariş takibi için bir panel kullanacak.
    4. Angular, React.js veya mobil uygulamalar için Flutter gibi frontend teknolojileri kullanılabilir.
2. **Sunucu Katmanı (Backend)**
    1. İş mantığının yönetileceği yer olacak.
    2. Restoranlar, menüler, siparişler, kullanıcılar ve ödeme işlemlerini yönetecek bir API sağlayacak.
    3. Java Spring Boot, Node.js veya Python Django gibi backend framework'leri kullanılabilir.
3. **Veritabanı Katmanı**
    1. Kullanıcılar, restoranlar, yemekler, siparişler ve ödemelerle ilgili veriler saklanacak.
    2. SQL tabanlı (PostgreSQL, MySQL) veya NoSQL (MongoDB) kullanılabilir.
4. **Üçüncü Parti Servisler**
    1. **Ödeme Servisi**: Stripe, PayPal veya yerel bir ödeme sağlayıcı ile ödeme işlemleri yapılacak.
    2. **Harita ve Adres Servisleri**: Google Maps API kullanılarak adresler doğrulanacak ve teslimat güzergahları belirlenecek.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **b) Modüller**
- ✓ Sistemi temel modüllere bölelim:

1. **Kullanıcı Yönetimi Modülü**: Kullanıcı kayıt, giriş, profil düzenleme.
2. **Restoran Yönetimi Modülü**: Restoranlar menülerini yönetebilir, yeni yemekler ekleyebilir ve fiyatları güncelleyebilir.
3. **Yemek ve Menü Yönetimi Modülü**: Her restoranın menüsünde yer alan yemekleri listeleme, kategoriye göre filtreleme ve detaylarını görüntüleme.
4. **Sipariş Yönetimi Modülü**: Kullanıcılar sepeti yönetir, sipariş verir ve siparişin durumunu takip eder.
5. **Ödeme Yönetimi Modülü**: Kullanıcılar ödeme işlemlerini gerçekleştirir.
6. **Yorum ve Puanlama Modülü**: Kullanıcılar restoranlar ve yemekler hakkında yorum yapabilir ve puan verebilir.
7. **Restoran Paneli**: Restoranlar siparişleri görüntüler, onaylar ve teslimat sürecini yönetir.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **c) Veri Akışı**
- ✓ Kullanıcıların sistemi nasıl kullanacağını basit bir örnekle inceleyelim:

1. Kullanıcı siteye giriş yapar.
2. Yakındaki restoranları görüntüler ve bir restorandan yemek seçer.
3. Seçilen yemekler sepete eklenir ve kullanıcı ödeme adımına geçer.
4. Ödeme işlemi tamamlandığında sipariş oluşturulur ve kullanıcı sipariş takibini başlatır.
5. Restoran siparişi onaylar ve yemek hazırlanmaya başlar.
6. Teslimat yapıldıktan sonra kullanıcı siparişi kapatabilir ve yorum yapabilir.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- **2. Detaylı Tasarım (Low-Level Design - LLD)**
- Bu aşamada, her modül ve bileşen için daha ayrıntılı bir tasarım yapıyoruz.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

✓ **a) Modül Tasarımı**

✓ Her bir modülün nasıl çalışacağını ve alt modüllerini detaylandırmamız gerekiyor.

1. **Kullanıcı Yönetimi Modülü**
   1. **Kayıt ve Giriş İşlemleri**: Kullanıcı, e-posta doğrulaması ile sisteme kayıt olabilir ve giriş yapabilir.
   2. **Profil Düzenleme**: Kullanıcı, adres bilgilerini ve ödeme bilgilerini güncelleyebilir.
2. **Restoran Yönetimi Modülü**
   1. **Menü Yönetimi**: Restoran, menüsüne yemek ekleyebilir, fiyatları güncelleyebilir ve yemekleri düzenleyebilir.
   2. **Sipariş Yönetimi**: Restoran, aldığı siparişleri görüntüleyebilir, siparişi hazırlamaya başlayabilir ve teslimat sürecini yönetebilir.
3. **Yemek ve Menü Yönetimi Modülü**
   1. **Yemek Listeleme**: Kullanıcı, restoranların menüsünü görüntüler. Yemekler fiyat, puan, kategori gibi filtrelerle sıralanabilir.
   2. **Yemek Detayı**: Yemek ismi, malzemeler, fiyat, kalori bilgisi ve restoran bilgileri görüntülenir.
4. **Sipariş Yönetimi Modülü**
   1. **Sepet Yönetimi**: Kullanıcılar bir veya daha fazla yemeği sepete ekleyebilir. Sepet güncellenebilir ya da temizlenebilir.
   2. **Sipariş Takibi**: Siparişin durumu (hazırlanıyor, yolda, teslim edildi) kullanıcıya gösterilir.
5. **Ödeme Yönetimi Modülü**
   1. **Ödeme İşlemleri**: Kullanıcılar, Stripe API veya benzeri bir ödeme servisi aracılığıyla ödemelerini tamamlayabilirler.
   2. **Ödeme Onayı**: Ödeme başarılı olduğunda sipariş veritabanına kaydedilir ve kullanıcıya e-posta ile onay gönderilir.
6. **Yorum ve Puanlama Modülü**
   1. **Yorum Yapma**: Kullanıcılar sadece sipariş verdikleri restoranlara ve yemeklere yorum yapabilirler.
   2. **Puanlama**: Her yemek ve restoran için 1-5 arası puanlama yapılabilir.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **b) Sınıf Diyagramları**
- ✓ Sınıf diyagramları, sistemdeki sınıfları ve ilişkilerini tanımlar. Örnek bazı sınıflar:
- **User**: Kullanıcıyı temsil eder. Attributeleri: kullanıcı adı, e-posta, şifre, adresler.
- **Restaurant**: Restoranı temsil eder. Attributeleri: isim, adres, menü, puanlama.
- **Order**: Siparişleri temsil eder. Attributeleri: sipariş numarası, kullanıcı, restoran, yemekler, durum.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **c) Veritabanı Tasarımı**
- ✓ Veritabanı yapısını tanımlayalım:

1. **Users** tablosu: Kullanıcı bilgilerini içerir (id, ad, e-posta, şifre, adres).
2. **Restaurants** tablosu: Restoran bilgilerini içerir (id, isim, adres, puan).
3. **Dishes** tablosu: Yemek bilgilerini içerir (id, restoran_id, isim, fiyat, kategori).
4. **Orders** tablosu: Sipariş bilgilerini içerir (id, kullanıcı_id, restoran_id, toplam_tutar, sipariş_tarihi, sipariş_durumu).
5. **Reviews** tablosu: Yorum ve puanları içerir (id, yemek_id, kullanıcı_id, puan, yorum).

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **d) API Tasarımı**
- ✓ RESTful API ile veritabanı ve arayüz arasındaki veri alışverişini yönetelim. Örnek API uç noktaları:
- **POST /login**: Kullanıcı girişi.
- **GET /restaurants**: Restoranları listele.
- **POST /order**: Yeni sipariş oluştur.
- **GET /order/**
- : Sipariş durumunu görüntüle.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **e) Algoritma ve Akış Diyagramları**
- ✓ Kullanıcı bir yemek siparişi verdiğinde şu adımları izleyen bir algoritma çalışır:

1. Kullanıcı sepetini onaylar ve ödeme sayfasına yönlendirilir.
2. Ödeme Stripe API ile doğrulanır.
3. Ödeme başarılıysa sipariş veritabanına kaydedilir ve restoran bilgilendirilir.
4. Restoran, sipariş durumunu günceller (hazırlanıyor -> teslim ediliyor -> teslim edildi).
5. Teslimat sonrası kullanıcıya teslimat bildirimi yapılır.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

✓ **f) Hata Yönetimi**

- Ödeme başarısız olduğunda kullanıcıya hata mesajı gösterilir.
- Yetersiz stok veya restoranın hizmet dışı olduğu durumlar kullanıcıya bildirilir.

Sistem Tasarımı – Örnek Proje 2 : Online Yemek Sipariş Platformu

- ✓ **g) Performans Optimizasyonu**
- **Önbellekleme**: Popüler restoranlar ve yemekler Redis kullanılarak önbelleğe alınır.
- **Sorgu Optimizasyonu**: Restoran aramaları ve yemek listeleri için verit

LAB Ödevi

- ✓ Üniversite Öğrenci Bilgi Sistemi (ÖBS) Sistem Tasarım Adımlarını Uygulayınız

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Kodlama

- ✓ Yazılım yaşam döngüsünde "kodlama" adımı, belirlenen gereksinimlere ve tasarıma uygun olarak yazılımın yazıldığı aşamadır. Kodlama adımı, çeşitli alt aşamalardan oluşur.

- ✓ 1. **Kodlama Planının Yapılması**
- ✓ 2. **Modül ve Sınıf Yapısının Oluşturulması**
- ✓ 3. **Fonksiyonların Geliştirilmesi**
- ✓ 4. **Veritabanı ve Diğer Harici Bağlantıların Kodlanması**
- ✓ 5. **Kodun Test Edilmesi ve Hata Ayıklama (Debugging)**
- ✓ 6. **Optimizasyon**
- ✓ 7. **Kodun Gözden Geçirilmesi (Code Review)**
- ✓ 8. **Dökümantasyon**

Kodlama - 1. Kodlama Planının Yapılması

Kodlama Planının Yapılması aşaması, yazılım geliştirme sürecinde kodlama adımının temelini oluşturan bir hazırlık sürecidir.

Bu aşamada, geliştirme sürecinin etkili ve düzenli bir şekilde ilerleyebilmesi için bazı önemli planlamalar yapılır

Kodlama - 1. Kodlama Planının Yapılması

✓ **1. Programlama Dillerinin ve Araçların Belirlenmesi**

- **Programlama Dili Seçimi**: Projede kullanılacak programlama dili veya dillerini belirlemek, kodlamanın ilk adımıdır. Programlama dili, projenin ihtiyaçlarına, performans gereksinimlerine, ekibin bilgi birikimine ve hedef platformlara göre seçilir. Örneğin, yüksek performans gerektiren sistemlerde C++ veya Rust gibi diller tercih edilirken, web tabanlı projelerde JavaScript veya Python kullanılması yaygındır.
- **Geliştirme Ortamı (IDE) ve Araçların Belirlenmesi**: Kod yazmak için kullanılacak geliştirme ortamı (IDE), sürüm kontrol sistemi (örneğin, Git), hata ayıklama araçları ve test araçları gibi gerekli tüm yazılımlar ve araçlar belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

- ✓ **2. Kodlama Standartlarının Belirlenmesi**
- **Kodlama Standartları**: Proje boyunca bir standart belirlemek, ekip üyelerinin kodu daha okunabilir ve anlaşılır şekilde yazmasını sağlar. Bu standartlar, isimlendirme kuralları (örneğin, değişken ve fonksiyon isimleri), kod biçimlendirme kuralları (girintiler, boşluk kullanımı) ve yorum ekleme kurallarını içerir.
- **Güvenlik ve Performans Standartları**: Projede uyulması gereken güvenlik ve performans standartları tanımlanır. Örneğin, şifreleme yöntemleri, güvenlik açıklarının nasıl ele alınacağı ve kodun hızlı çalışması için optimizasyon kriterleri belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

- ✓ **3. Modüler Yapının ve Sorumlulukların Belirlenmesi**
- **Modül Yapısının Tanımlanması**: Yazılımın modüllere veya katmanlara ayrılarak geliştirilmesi, kodun daha düzenli ve yönetilebilir olmasını sağlar. Kodlama planında, hangi modüllerin hangi işlevleri gerçekleştireceği detaylandırılır. Örneğin, veritabanı işlemlerini yöneten bir modül, kullanıcı arayüzünü oluşturan bir modül vb. şeklinde ayrımlar yapılır.
- **Sorumlulukların Dağıtılması**: Ekip üyeleri arasındaki iş bölümü planlanır. Hangi geliştiricinin hangi modüllerden veya işlevlerden sorumlu olacağı belirlenir. Bu, ekip içindeki iş birliğini artırır ve süreçlerin daha hızlı ilerlemesini sağlar.

Kodlama - 1. Kodlama Planının Yapılması

✓ **4. Zaman Çizelgesi ve İş Takviminin Oluşturulması**

- **Kodlama Zaman Çizelgesi**: Kodlama sürecinin hangi aşamalarla ve ne zaman tamamlanması gerektiği planlanır. Bu süreç, genellikle Agile veya Waterfall gibi bir yazılım geliştirme metodolojisine göre organize edilir.
- **Milestones (Kilit Noktalar)**: Belirli tarihlere göre tamamlanması gereken önemli aşamalar (milestones) belirlenir. Örneğin, ilk versiyonun tamamlanma tarihi, birim testlerin yapılması için son tarih gibi kilometre taşları belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓ **5. Risklerin ve Önlemlerin Belirlenmesi**

- **Risk Analizi**: Kodlama sırasında karşılaşılabilecek teknik zorluklar, kaynak yetersizlikleri, güvenlik sorunları gibi potansiyel riskler belirlenir. Her riskin etkisi değerlendirilir ve önlemler planlanır.
- **Yedekleme ve Sürüm Kontrolü**: Kodun güvenliğini sağlamak ve verimliliği artırmak için sürüm kontrol sistemleri (örneğin, Git) kullanılır. Kodlama planı yapılırken, kod yedeklemelerinin nasıl ve ne sıklıkla yapılacağı belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓ **6. Test Planının Hazırlanması**

- **Test Stratejileri**: Kodlama sırasında veya sonrasında yapılacak olan testlerin türleri belirlenir (örneğin, birim testi, entegrasyon testi, kullanıcı kabul testi). Test stratejisi ve testin kapsamı belirlenir.
- **Otomatik Test Araçlarının Seçilmesi**: Eğer proje otomatik test gerektiriyorsa, bu testlerin hangi araçlarla yapılacağı ve otomatikleştirilmiş testlerin kodlama sırasında nasıl kullanılacağı belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓ **7. Dökümantasyon Planı**

- **Dökümantasyonun Kapsamı**: Kodlama sürecinde veya sonunda hazırlanacak teknik dökümantasyonun kapsamı belirlenir. Kodun nasıl çalıştığını, hangi işlevleri yerine getirdiğini açıklayan belgeler hazırlanır.
- **Dökümantasyon Standartları**: Projeye uygun dökümantasyon standartları ve formatları belirlenir. Örneğin, hangi kodların yorumlanacağı, API belgelerinin hangi formatta yazılacağı, kullanım talimatlarının nasıl hazırlanacağı gibi konulara karar verilir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- **Modül ve Sınıf Yapısının Oluşturulması** aşaması, yazılımın bileşenlerine ayrılarak daha yönetilebilir ve anlaşılabilir hale getirilmesini sağlar.

- Bu adım, yazılımın işlevlerini mantıksal olarak gruplandırarak modüller ve sınıflar oluşturmak için kritik bir süreçtir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- ✓ **1. Modüler Tasarımın Planlanması**
- **Fonksiyonelliklere Göre Modüllerin Ayrılması**: Yazılımın işlevlerine göre mantıksal bileşenlere (modüllere) ayrılması gereklidir. Örneğin, bir e-ticaret uygulamasında "ürün yönetimi," "kullanıcı yönetimi" ve "sipariş yönetimi" gibi işlevsel modüller olabilir. Bu sayede her modül kendi sorumluluğuna odaklanır.
- **Bağımlılıkların Belirlenmesi**: Modüller arasındaki bağımlılıkları ve ilişkileri belirlemek, yazılımın sağlam bir şekilde yapılandırılması için önemlidir. Modüllerin birbirine bağımlılığını azaltmak, gelecekteki güncellemeleri ve değişiklikleri kolaylaştırır.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓ **2. Sınıfların Tanımlanması**

- **Temel Nesnelerin Belirlenmesi**: Sınıflar, nesne tabanlı programlama (OOP) paradigmasıyla tasarlanan yazılımlarda, sistemin temel nesnelerini temsil eder. Bu aşamada, yazılımın ana varlıkları (örneğin, “Kullanıcı,” “Ürün,” “Sipariş”) belirlenir ve bu varlıklar için sınıflar oluşturulur.
- **Sınıf Sorumluluklarının Belirlenmesi (Single Responsibility Principle)**: Her sınıf, tek bir işlevi yerine getirmekten sorumlu olmalıdır. Sınıfların sorumlulukları belirlenirken, sınıfın hangi işlevleri gerçekleştireceği ve hangi verileri içereceği tanımlanır. Örneğin, bir “Kullanıcı” sınıfı kullanıcı bilgilerini ve kullanıcıya ait işlemleri içerir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓ **3. Veri Yapıları ve Özelliklerin Belirlenmesi**

- **Sınıf Özellikleri (Attributes)**: Her sınıfın saklayacağı veri, yani özellikler belirlenir. Örneğin, bir "Kullanıcı" sınıfı için ad, soyad, e-posta gibi özellikler tanımlanabilir. Özellikler, sınıfın hangi veriyi temsil edeceğini gösterir.
- **Veri Tiplerinin Belirlenmesi**: Her özellik için uygun veri tipleri belirlenir. Örneğin, "Kullanıcı" sınıfındaki "doğum tarihi" özelliği bir tarih tipi olabilirken, "kullanıcı adı" ise metin tipi olabilir. Veri tiplerinin doğru seçilmesi, veri tutarlılığı ve kodun verimli çalışması için önemlidir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- ✓ **4. Sınıf Metotlarının (Fonksiyonların) Belirlenmesi**
- **İşlevlerin Tanımlanması**: Her sınıfın, kendi verilerini işlemek için gereken işlevleri (metotları) belirlenir. Metotlar, sınıfın hangi işlevleri gerçekleştireceğini ve hangi işlemleri yapacağını belirler. Örneğin, "Sipariş" sınıfında sipariş durumu güncelleme, ürün ekleme veya çıkarma gibi metotlar tanımlanabilir.
- **Erişim Belirleyicilerinin Kullanılması (Encapsulation)**: Sınıfın özelliklerinin ve metotlarının nasıl erişileceği, yani hangi erişim belirleyicilerinin (public, private, protected gibi) kullanılacağı belirlenir. Bu, sınıfın verilerini dış müdahalelere karşı koruma altına almak için önemlidir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓ **5. Sınıflar Arası İlişkilerin Tanımlanması**

- **İlişki Türlerinin Belirlenmesi**: Sınıflar arasındaki ilişkiler belirlenir. Nesne tabanlı programlamada, sınıflar arasındaki ilişkiler dört ana türde olabilir: **Birleşim (Aggregation)**, **Bileşim (Composition)**, **Kalıtım (Inheritance)** ve **Birliktelik (Association)**. Örneğin, bir "Sipariş" sınıfı "Kullanıcı" sınıfıyla bir birliktelik ilişkisi kurabilir.
- **Kalıtım İlişkilerinin Kurulması**: Sınıflar arasındaki kalıtım ilişkileri, ortak özelliklerin ve işlevlerin bir üst sınıfa taşınmasıyla kurulur. Kalıtım, kod tekrarını azaltır ve kodun daha modüler hale gelmesini sağlar. Örneğin, "Müşteri" ve "Çalışan" sınıfları, ortak özellikleri "Kişi" sınıfından miras alabilir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- ✓ **6. Arayüz (Interface) ve Soyut Sınıfların Oluşturulması**
- **Arayüzlerin (Interfaces) Tanımlanması**: Belirli işlevlerin farklı sınıflar tarafından uygulanmasını sağlamak için arayüzler tanımlanır. Arayüzler, sınıfların belirli bir işlevselliği yerine getirmesi gerektiğini belirtir ve böylece kodda esneklik sağlar.
- **Soyut Sınıfların Tanımlanması**: Eğer sınıfların bazı ortak işlevleri olacak ama her sınıfın bu işlevleri farklı bir şekilde uygulaması gerekiyorsa, soyut sınıflar oluşturulabilir. Soyut sınıflar, belirli işlevlerin miras alınan sınıflar tarafından uygulanmasını zorunlu kılar.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- ✓ **7. Modüller ve Sınıflar İçin Yorum ve Dökümantasyon Yazılması**
- **Yorum Eklemek**: Her sınıfın ve modülün işlevini açıklayan yorumlar eklenir. Bu, kodun daha sonra bakımını yapacak kişilerin veya yeni geliştiricilerin kodu anlamasını kolaylaştırır.
- **Dökümantasyon Hazırlanması**: Modüllerin ve sınıfların hangi işlevleri gerçekleştirdiğini ve nasıl kullanıldığını anlatan teknik dökümantasyonlar hazırlanır. Özellikle karmaşık sınıflar için dökümantasyon, projeye yeni katılan ekip üyelerinin süreci anlamasını hızlandırır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- **Fonksiyonların Geliştirilmesi** aşaması, yazılımın gerçekleştirmesi gereken işlevlerin kodlandığı süreçtir.

- Bu aşamada, her modül veya sınıf için planlanan işlevler ve bu işlevlerin çalışması gereken fonksiyonlar detaylandırılarak kodlanır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **1. Fonksiyon Gereksinimlerinin Belirlenmesi**
- **İşlevlerin Tanımlanması**: Her fonksiyonun ne yapması gerektiği belirlenir. Bu, fonksiyonun amacını, hangi işi gerçekleştirdiğini ve hangi problemi çözdüğünü netleştirir. Örneğin, bir "kullanıcı girişi" fonksiyonu, kullanıcının doğru kimlik bilgileriyle giriş yapıp yapmadığını kontrol eder.
- **Girdilerin ve Çıktıların Belirlenmesi**: Fonksiyonun hangi girdileri alacağı ve hangi çıktıları üreteceği tanımlanır. Bu, fonksiyonun çalışma mantığının anlaşılması için önemlidir. Örneğin, bir "fiyat hesaplama" fonksiyonu için ürün miktarı ve birim fiyat girdi olarak alınabilir, toplam fiyat ise çıktı olarak döndürülür.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **2. Fonksiyon İsimlendirme Standartlarına Uygun İsim Verilmesi**
- **Anlamlı ve Tutarlı İsimlendirme**: Fonksiyon isimlerinin ne iş yaptığını anlatacak şekilde seçilmesi önemlidir. Örneğin, “hesapla()” yerine, “toplamFiyatHesapla()” gibi daha açıklayıcı bir isim tercih edilmelidir.
- **Kodlama Standartlarına Uygunluk**: Projede kullanılan isimlendirme standartlarına (örneğin, camelCase veya snake_case) uygun olarak isimlendirme yapılır. Bu, kodun okunabilirliğini ve sürdürülebilirliğini artırır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **3. Fonksiyonun İç Yapısının Tasarlanması**
- **Kontrol Akışının Belirlenmesi**: Fonksiyonun gerçekleştireceği işlemler sıralanır ve kontrol akışları (if-else, switch-case, döngüler gibi) tasarlanır. Örneğin, bir "sipariş durumu güncelle" fonksiyonu için kontrol akışı, sipariş durumuna göre değişebilir.
- **Koşullar ve Döngülerin Kullanımı**: Fonksiyonun işlemlerini gerçekleştirmek için gereken koşullar (if, else, switch) ve tekrarlayan işlemler için döngüler (for, while) tanımlanır. Kontrol yapıları, fonksiyonun iş akışını düzenli bir şekilde yönetir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **4. Fonksiyon İçerisinde Veri İşlemleri Yapma**
- **Veri Manipülasyonları**: Fonksiyon içerisinde, girdilerin işlenmesi, hesaplanması veya değiştirilmesi gibi veri işlemleri yapılır. Örneğin, bir “indirim hesapla” fonksiyonu, toplam fiyata indirim oranını uygulayarak yeni bir değer döndürebilir.
- **Yerel Değişkenler ve Parametrelerin Kullanımı**: Fonksiyon içinde kullanılan değişkenlerin ve parametrelerin doğru bir şekilde tanımlanması ve kullanılması önemlidir. Yerel değişkenler, fonksiyonun dışında kullanılmaz ve böylece kodun güvenliği ve okunabilirliği artar.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **5. Hata Yönetimi ve Hata Ayıklama (Debugging)**
- **Hata Kontrolleri (Error Handling)**: Fonksiyon içerisinde karşılaşılabilecek olası hatalar için hata yönetimi yapılır. Örneğin, bir "dosya aç" fonksiyonu dosya mevcut değilse veya erişilemiyorsa uygun bir hata mesajı ile kullanıcıyı bilgilendirebilir.
- **Hata Ayıklama (Debugging)**: Fonksiyon geliştirme sürecinde veya sonrasında hata ayıklama işlemi yapılır. Hata ayıklama, fonksiyonun beklenmedik durumlarda nasıl davrandığını görmek için önemlidir ve gerektiğinde sorunlu kod parçalarını düzeltmeyi sağlar.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓ **6. Fonksiyon Testlerinin Yazılması**

- **Birim Testleri (Unit Testing)**: Fonksiyonların tek başına doğru çalışıp çalışmadığını kontrol etmek için birim testleri yazılır. Her fonksiyon, beklenen sonuçları verip vermediği açısından test edilir. Örneğin, “toplamFiyatHesapla” fonksiyonu, belirli bir giriş için doğru toplam fiyatı verip vermediği açısından test edilebilir.
- **Sınır Durum Testleri**: Fonksiyonların olağan dışı durumlarda (örneğin, negatif veya sıfır değerler, çok büyük sayılar gibi) nasıl çalıştığını kontrol etmek için sınır durum testleri yapılır. Bu testler, fonksiyonun güvenilirliğini ve sağlamlığını ölçer.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **7. Optimizasyon ve Performans İyileştirmeleri**
- **Algoritma İyileştirmeleri**: Fonksiyonun verimli çalışması için kullanılan algoritmalar optimize edilir. Daha hızlı ve daha az kaynak tüketen algoritmalar tercih edilir. Örneğin, büyük bir veri kümesi üzerinde işlem yapacaksa, daha hızlı algoritmalar veya veri yapıları kullanılabilir.
- **Bellek Kullanımı ve Kaynak Yönetimi**: Fonksiyonun bellek kullanımı kontrol edilir ve mümkünse azaltılır. Gereksiz veriler bellekte tutulmaz ve belleği etkin kullanacak düzenlemeler yapılır. Büyük veri işlemlerinde belleğin sınırlı olduğu durumlarda bu oldukça kritiktir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **8. Kodun Anlaşılabilirliğini Artırmak için Yorumlar Eklemek**
- **Açıklayıcı Yorumlar**: Fonksiyonun içindeki önemli kod bölümlerine açıklayıcı yorumlar eklenir. Özellikle karmaşık işlemler gerçekleştiren fonksiyonlar için yorumlar, kodun ileride bakımını yapacak kişilerin kodu daha kolay anlamasını sağlar.
- **Dokümantasyon Standartlarına Uyum**: Yorumlar belirli bir dokümantasyon standardına (örneğin, Javadoc, docstrings gibi) uygun olarak yazılırsa, hem geliştirme sırasında hem de sonrasında daha kolay referans alınabilir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓ **9. Gözden Geçirme (Code Review)**
- **Kod İncelemesi (Code Review)**: Fonksiyon geliştirme işlemi tamamlandıktan sonra, kod bir veya birkaç ekip üyesi tarafından gözden geçirilir. Kod incelemesi, kodun okunabilirliği, hatalardan arındırılması ve kod standartlarına uygunluğunu sağlamak için yapılır.
- **İyileştirme Önerileri**: Gözden geçirme sırasında yapılan öneriler doğrultusunda fonksiyonlar yeniden düzenlenebilir ve geliştirilebilir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓ **10. Fonksiyonların Dökümantasyonu**

- **Fonksiyon Açıklamaları**: Fonksiyonun ne yaptığı, hangi parametreleri aldığı, ne tür bir çıktı verdiği gibi bilgiler dokümante edilir. Bu, hem ekip üyeleri için hem de yazılımın gelecekteki sürümleri için referans kaynağı sağlar.
- **Girdi ve Çıktı Açıklamaları**: Her parametrenin neyi ifade ettiği, hangi değerleri kabul edebileceği ve fonksiyonun döndüreceği sonuçlar açıklanır. Bu bilgiler, yazılımın sürdürülebilirliği için önemlidir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

- **Veritabanı ve Diğer Harici Bağlantıların Kodlanması** aşaması, yazılımın ihtiyaç duyduğu veri depolama, veri erişim ve harici sistemlerle bağlantı kurma işlemlerinin kodlandığı adımdır.

- Bu adımda, yazılımın veritabanına bağlanması, harici API'ler ve servislerle etkileşime geçmesi sağlanır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **1. Veritabanı Tasarımının ve Yapısının Belirlenmesi**

- **Veritabanı Seçimi**: Projenin ihtiyaçlarına göre uygun bir veritabanı türü (örneğin, ilişkisel veritabanı için MySQL, PostgreSQL veya NoSQL veritabanı için MongoDB, Redis) seçilir. Veritabanı türü, yazılımın veri yapısına, ölçeklenebilirlik ihtiyaçlarına ve performans gereksinimlerine göre belirlenir.
- **Tablo Yapısının Tasarlanması**: İlişkisel veritabanı kullanılıyorsa, gerekli tablolar, sütunlar ve veri türleri tasarlanır. Örneğin, bir e-ticaret uygulamasında "Kullanıcı," "Ürün," ve "Sipariş" gibi tablolar oluşturulabilir.
- **İlişkiler ve Yabancı Anahtarlar**: Tablolar arasında ilişkiler (birçoktan-bire, birden-çoğa) belirlenir ve yabancı anahtarlar tanımlanır. Bu, verilerin ilişkili bir şekilde düzenlenmesini sağlar. Örneğin, "Sipariş" tablosu ile "Kullanıcı" tablosu arasında bir ilişki kurulabilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **2. Veritabanı Bağlantısının Kurulması**

- **Bağlantı Yapılandırması**: Yazılımın veritabanına bağlanabilmesi için veritabanı bağlantı bilgileri (host, port, kullanıcı adı, şifre gibi) yapılandırılır. Bu bağlantı bilgileri genellikle yapılandırma dosyalarında veya çevre değişkenlerinde tutulur.
- **Bağlantı Kütüphanesi Seçimi**: Veritabanına bağlantıyı yönetmek için uygun kütüphane veya çerçeve (örneğin, Python için SQLAlchemy, Java için JDBC) seçilir ve entegre edilir. Bu kütüphaneler, veritabanı bağlantılarını ve işlemlerini daha güvenli ve düzenli bir şekilde yönetir.
- **Bağlantı Havuzu (Connection Pooling)**: Uygulamanın performansını artırmak için bağlantı havuzu kullanılması düşünülür. Bağlantı havuzu, aynı anda birden çok veritabanı bağlantısını yöneten ve her işlem için yeni bir bağlantı açma maliyetini düşüren bir tekniktir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **3. CRUD İşlemlerinin Kodlanması (Create, Read, Update, Delete)**

- **Veri Ekleme (Create)**: Yeni verilerin eklenmesi için gerekli SQL sorguları veya ORM (Nesne İlişkisel Eşleme) işlemleri kodlanır. Örneğin, yeni bir kullanıcı eklemek için bir INSERT sorgusu veya ORM kullanılıyorsa add_user() gibi bir fonksiyon oluşturulur.

- **Veri Okuma (Read)**: Verilerin okunması için gerekli sorgular veya ORM işlemleri yazılır. Bu adım, veri tabanından spesifik verilerin çekilmesi veya listelenmesi için kullanılır. Örneğin, tüm kullanıcıları listeleyen veya belirli bir ürün bilgisini getiren sorgular.

- **Veri Güncelleme (Update)**: Mevcut verilerin güncellenmesi için güncelleme sorguları veya işlemleri oluşturulur. Örneğin, kullanıcının e-posta adresini güncellemek için bir UPDATE sorgusu yazılabilir.

- **Veri Silme (Delete)**: Veritabanındaki verilerin silinmesi işlemlerini gerçekleştiren kodlar yazılır. Örneğin, eski sipariş kayıtlarını silmek için DELETE sorguları veya ORM metodları kullanılır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **4. Veri Güvenliğinin Sağlanması**

- **SQL Injection Önleme**: SQL enjeksiyonuna karşı güvenlik önlemleri alınır. Parametreli sorgular kullanmak ve doğrudan kullanıcı girdilerini SQL sorgularına eklememek gibi teknikler, veritabanı güvenliğini sağlar.
- **Veri Şifreleme ve Anonimleştirme**: Özellikle hassas bilgiler (örneğin, şifreler, kredi kartı bilgileri) için veri şifreleme veya maskeleme uygulanır. Bu bilgiler, şifrelenmiş veya anonimleştirilmiş halde saklanarak güvenlik sağlanır.
- **Yetkilendirme ve Erişim Kontrolleri**: Veritabanına erişim, belirli kullanıcı veya kullanıcı grupları için kısıtlanabilir. Verilere yalnızca belirli kullanıcılar erişebilmeli ve yalnızca belirli işlemleri gerçekleştirebilmelidir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

- ✓ **5. Veri Aktarımı ve Dönüştürme (ETL)**
- **Veri Alma (Extract)**: Eğer farklı kaynaklardan veri alınacaksa, bu verilerin nasıl alınacağı kodlanır. Harici bir API veya dosya sisteminden veri çekilecekse gerekli bağlantılar yapılır.
- **Veri Dönüştürme (Transform)**: Harici kaynaklardan gelen verilerin uygun formata dönüştürülmesi sağlanır. Bu, verilerin yazılımda kullanılabilir hale gelmesi için gerekebilir.
- **Veri Yükleme (Load)**: Dönüştürülen verilerin veritabanına veya hedef veri deposuna yüklenmesi işlemleri yapılır. Bu adımda, ETL işlemleri sırasında veri doğruluğunu sağlamak için çeşitli kontroller yapılır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

- ✓ **6. Harici API ve Servislere Bağlantı Kodlarının Yazılması**
- **API Erişim Katmanının Oluşturulması**: Harici bir API'ye bağlanmak için gereken kod yapısı ve fonksiyonlar yazılır. API anahtarı, URL, parametreler gibi bağlantı bilgileri tanımlanır. Örneğin, bir hava durumu API'sine bağlanmak için gerekli bağlantı ve istek bilgileri düzenlenir.
- **API Çağrılarının Yönetilmesi**: API çağrıları belirli aralıklarla yapılacaksa veya belirli durumlarda tetiklenecekse, bu iş akışları kodlanır. Örneğin, kullanıcı oturum açtığında bir kimlik doğrulama API'sine istek gönderilebilir.
- **Gelen Verilerin İşlenmesi**: Harici API veya servislerden dönen veri, gerekli işlemler için işlenir ve uygun formata dönüştürülür. Dönen JSON, XML gibi veri formatları uygun veri yapısına çevrilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **7. Veritabanı ve Harici Bağlantıların Hata Yönetimi**

- **Bağlantı Hatalarının Yönetilmesi**: Veritabanı veya API bağlantısı başarısız olduğunda hata yönetimi yapılır. Bağlantı kesildiğinde veya sunucuya erişim sağlanamadığında, kullanıcıya bilgilendirme mesajları gösterilir veya otomatik yeniden bağlanma işlemleri uygulanır.
- **Hata Günlüğü Tutma**: Veritabanı veya harici bağlantılarda karşılaşılan hataların kaydedilmesi için hata günlüğü (log) tutulur. Böylece hatalar daha sonra analiz edilebilir ve gerektiğinde çözüm üretilebilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **8. Performans Optimizasyonu ve Önbellekleme (Caching)**

- **Veritabanı Sorgularının Optimizasyonu**: Sorguların hızlı çalışması için veritabanı indeksleri, optimize edilmiş sorgular ve SQL analizleri yapılır. Gereksiz veri getirmekten kaçınılır ve büyük veri kümesi içeren sorgularda optimizasyon uygulanır.
- **Önbellekleme (Caching)**: Sık kullanılan veriler için önbellekleme yapılır, böylece her sorguda veritabanına başvurmaya gerek kalmaz. Redis gibi önbellek çözümleri, yazılımın performansını artırır ve veritabanı üzerindeki yükü azaltır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **9. Bağlantı Testlerinin Yapılması**

- **Veritabanı Bağlantı Testleri**: Veritabanına yapılan bağlantıların doğruluğunu ve sürekliliğini test etmek için bağlantı testleri yapılır. Bu testler, bağlantının stabil ve hatasız çalıştığını garanti eder.
- **API ve Servis Testleri**: Harici API ve servislerin bağlantı testleri yapılır. API'den dönen verilerin beklenen formatta olup olmadığı kontrol edilir ve belirli durumlarda nasıl yanıt verdikleri test edilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **10. Dokümantasyon ve Yedekleme**

- **Veritabanı ve API Dokümantasyonu**: Veritabanı tablolarının yapısı, sütun açıklamaları ve kullanılan API'lerin dokümantasyonu hazırlanır. Bu bilgiler, geliştiricilerin ve bakım ekiplerinin veri yapısını anlamalarına yardımcı olur.
- **Yedekleme ve Kurtarma Prosedürleri**: Veritabanının yedeklenmesi ve acil durumlarda kurtarılabilmesi için yedekleme prosedürleri oluşturulur. Bu adım, verilerin güvenliği ve bütünlüğü açısından önemlidir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- ✓ **1. Kodun Test Edilmesi**
- ✓ Kodlama tamamlandıktan sonra, kodun beklendiği gibi çalıştığını doğrulamak için test işlemleri gerçekleştirilir. Bu aşamada, hem manuel hem de otomatik test yöntemleri kullanılabilir.
- ✓ **a) Test Türleri**
- **Birim Testleri (Unit Tests):**
  - Yazılımın her bir bileşeninin (örneğin bir fonksiyonun) bağımsız olarak test edilmesidir.
  - Amaç, her birim kodun doğru çalıştığından emin olmaktır.
  - Örnek araçlar: JUnit (Java), pytest (Python).
- **Entegrasyon Testleri (Integration Tests):**
  - Farklı bileşenlerin birlikte çalışabilirliğini kontrol eder.
  - Veri akışı, API çağrıları gibi modüller arası iletişim test edilir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- **Sistem Testleri (System Tests):**
  - Yazılımın tamamının test edilmesi.
  - Kullanıcı senaryoları üzerinden yazılımın bütünsel olarak doğru çalışıp çalışmadığı kontrol edilir.
- **Kabul Testleri (Acceptance Tests):**
  - Müşteri veya son kullanıcı tarafından yazılımın gereksinimlere uygunluğu test edilir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- ✓ **b) Test Güdümlü Geliştirme (Test-Driven Development - TDD)**
- Önce testlerin yazıldığı, ardından bu testleri geçecek kodun geliştirildiği bir yöntemdir.
- Bu yöntem, yazılım hatalarını en aza indirgemek için güçlü bir yaklaşımdır.
- ✓ **c) Test Otomasyonu**
- Test süreçlerinin otomatikleştirilmesi, tekrarlayan iş yükünü azaltır ve hızlı geri bildirim sağlar.
- Örnek araçlar: Selenium, Appium, Jenkins.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- ✓ **2. Hata Ayıklama (Debugging)**
- ✓ Hata ayıklama, test sırasında veya geliştirme sürecinde ortaya çıkan hataların nedenlerini bulma ve düzeltme sürecidir. Bu süreç, yazılımın hatasız ve doğru çalışmasını sağlamak için kritik öneme sahiptir.
- ✓ **a) Debugging Süreci**

1. **Hatayı Belirleme:**
    1. Testler sırasında ya da kullanıcıdan gelen geri bildirimler doğrultusunda hatalar tespit edilir.
    2. Örnek: Hatalı bir çıktı, çökme, beklenmeyen davranış.
2. **Hatanın İzolasyonu:**
    1. Hatanın kaynağını bulmak için kod parça parça incelenir.
    2. Bunun için kod içindeki belirli noktalara loglama veya breakpoint'ler eklenir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓ **2. Hata Ayıklama (Debugging)**

3. Hatanın Analizi:

1. Hatanın temel sebebi analiz edilir. Bu, genellikle yanlış algoritma kullanımı, eksik gereksinimlerin kodlanması ya da veri türü uyumsuzluğu gibi nedenlere dayanabilir.

4. Hatanın Düzeltilmesi:

1. Analiz sonucunda kodda gerekli değişiklikler yapılır.

5. Düzeltmenin Test Edilmesi:

1. Yapılan değişikliğin sorunu çözüp çözmediği test edilir.
2. Yeni hatalar yaratmadığından emin olmak için regresyon testi yapılabilir.

b) Debugging Yöntemleri

- **Manuel Debugging:**
  - Kodun adım adım çalıştırılarak hata kaynaklarının manuel olarak bulunması.
  - Örnek: Basit print komutları veya loglama teknikleri.
- **Araçlarla Debugging:**
  - IDE'lerin sunduğu hata ayıklama araçları kullanılır.
  - Breakpoint'ler ile kod belirli noktalarda durdurularak değişken değerleri incelenir.
  - Örnek araçlar: Visual Studio Debugger, PyCharm Debugger.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- ✓ **c) Yaygın Hata Türleri**
- **Sözdizimi Hataları (Syntax Errors):**
  - Dilin kurallarına aykırı yazım hataları. Örnek: Unutulan noktalı virgül, yanlış parantez kullanımı.
- **Mantıksal Hatalar (Logical Errors):**
  - Kod doğru çalışsa bile beklenen sonucu vermemesi. Örnek: Yanlış bir koşul yazımı.
- **Çalışma Zamanı Hataları (Runtime Errors):**
  - Kod çalışırken oluşan hatalar. Örnek: Sıfıra bölme hatası, null referans hatası.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- ✓ **d) Hata Ayıklama Teknikleri**
- **Binary Search Debugging:**
  - Kodun farklı bölümlerini kapatarak veya test ederek hatanın bulunduğu yeri daraltma.
- **Log Analizi:**
  - Kodun çalışması sırasında yazdırılan log mesajlarını inceleyerek hatanın nedenini tespit etme.
- **Stack Trace İncelemesi:**
  - Hata mesajları sırasında verilen "stack trace" bilgilerini kullanarak hangi fonksiyonların sorun yarattığını bulma.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓ **3. Test ve Debugging Araçları**

- **Test Araçları:** JUnit, pytest, Selenium.
- **Debugging Araçları:** Visual Studio, Eclipse Debugger, Chrome Developer Tools.

Kodlama - 6. Optimizasyon

- **Optimizasyon İşlemi**, kodlama adımında yazılımın performansını, verimliliğini ve kaynak kullanımını iyileştirmek için yapılan işlemleri ifade eder.

- Bu süreç, kodun daha hızlı çalışmasını, daha az kaynak tüketmesini ve daha az hata üretmesini sağlar. Optimizasyon, yazılımın hem kullanıcı deneyimini artırır hem de maliyetleri düşürür.

Kodlama - 6. Optimizasyon

- ✓ **1. Optimizasyon İşleminin Amacı**
- Yazılımın hızını artırmak.
- Daha az bellek, işlemci ve ağ kaynağı kullanmak.
- Kullanıcıların daha sorunsuz bir deneyim yaşamasını sağlamak.
- Teknik borcu (technical debt) azaltarak sürdürülebilirliği artırmak.
- Yazılımın büyük veri veya kullanıcı yüklerinde de çalışabilirliğini sağlamak (ölçeklenebilirlik).

Kodlama - 6. Optimizasyon

2. Optimizasyon Çeşitleri

a) Kod Optimizasyonu

•Kodun karmaşıklığını azaltarak daha verimli çalışmasını sağlamak.

•Gereksiz veya yinelenen kod parçalarını kaldırmak.

•Daha iyi algoritmalar veya veri yapıları kullanmak.

•Örnekler:

- Zaman karmaşıklığını O(n2)O(n^2)O(n2)'den O(nlogn)O(n \log n)O(nlogn)'e düşürmek.
- Tekrar eden işlemleri önlemek için önbellekleme (caching) kullanımı.

b) Veri Tabanı Optimizasyonu

•Veri tabanı sorgularının daha hızlı çalışmasını sağlamak.

•Gereksiz sorguları ve büyük veri taşımasını önlemek.

•Örnekler:

- İndeksleme ile sorgu hızını artırmak.
- Gereksiz JOIN işlemlerini azaltmak.
- Normalizasyon ve gerektiğinde denormalizasyon kullanmak.

Kodlama - 6. Optimizasyon

c) Bellek Optimizasyonu

- Yazılımın daha az bellek tüketmesi sağlanır.
- Bellek sızıntılarının (memory leak) önüne geçilir.
- Örnekler:
  - Gereksiz değişkenleri temizlemek.
  - Daha hafif veri tiplerini kullanmak (ör. float yerine int).
  - Gerekmediğinde nesneleri heap yerine stack'te tutmak.

d) İletişim ve Ağ Optimizasyonu

- Sunucu ve istemci arasındaki veri trafiğini minimize etmek.
- Daha hızlı veri aktarımı sağlamak.
- Örnekler:
  - Daha küçük ve sıkıştırılmış veri paketleri kullanmak (ör. gzip).
  - Gereksiz API çağrılarını azaltmak.
  - CDN (Content Delivery Network) kullanımı.

Kodlama - 6. Optimizasyon

e) Algoritma ve İşlem Optimizasyonu

•Daha etkili algoritmalar ve yöntemler kullanarak yazılımın verimliliğini artırmak.

•Örnekler:

- Büyük veri setleriyle çalışırken paralelleştirme (multi-threading).
- Zaman alan işlemler için asenkron programlama (asynchronous programming).

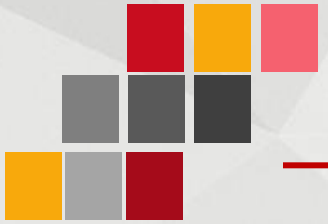
f) Kullanıcı Arayüzü (UI/UX) Optimizasyonu

•Arayüz elemanlarının daha hızlı yüklenmesini sağlamak.

•Daha düşük gecikme süreleriyle kullanıcılara daha iyi bir deneyim sunmak.

•Örnekler:

- Görselleri optimize etmek (ör. PNG yerine WebP).
- Yavaş yükleme (lazy loading) tekniklerini uygulamak.



Kodlama - 6. Optimizasyon

3. Optimizasyon İşlemi Adımları

a) Mevcut Performansı Değerlendirme

•Performans ölçüm araçları kullanarak yazılımın darboğazlarını belirlemek.

•Örnek araçlar:

- Kod için: **SonarQube**, **Linting araçları**.
- Performans için: **New Relic**, **AppDynamics**.
- Veri tabanı için: **SQL Profiler**, **EXPLAIN** komutları.

b) Sorunları Tespit Etme

•Performansı etkileyen kritik noktaları belirleme.

•Örnekler:

- Yavaş çalışan SQL sorguları.
- İhtiyaç duyulmayan API çağrıları.
- Karmaşık ve büyük algoritmalar.

Kodlama - 6. Optimizasyon

c) Çözüm Önerileri Geliştirme

•Alternatif yöntemler, algoritmalar veya araçlar belirlenir.

•Maliyet ve etkileri değerlendirilerek en uygun çözüm seçilir.

d) Kodun Düzenlenmesi

•Belirlenen çözümler uygulanır ve kodda gerekli değişiklikler yapılır.

•Kodun okunabilirliği ve sürdürülebilirliği korunur.

e) Test ve Doğrulama

•Optimizasyon sonrası yazılımın tüm işlevlerinin doğru çalıştığından emin olunur.

•Performans ve stres testleri yapılır.

f) Sonuçların İzlenmesi

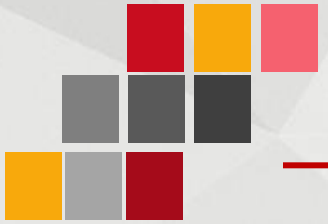
•Optimizasyonun yazılım üzerindeki etkileri izlenir.

•Gerektiğinde ek iyileştirmeler yapılır.

Kodlama - 6. Optimizasyon

4. Optimizasyon Sürecinde Kullanılan Araçlar

Araç/Platform	Kullanım Amacı
SonarQube	Kod kalitesini ve güvenliğini analiz etme
JProfiler, YourKit	Bellek ve CPU performansı izleme
SQL Profiler	Veri tabanı sorgularını analiz etme
JMeter, Locust	Performans ve yük testi
Google Lighthouse	Web uygulamaları için hız ve kalite ölçümü



Kodlama - 6. Optimizasyon

5. Optimizasyon İşleminin Zorlukları

- **Kodun Karmaşıklığı:** Karmaşık kodların optimize edilmesi zordur ve yanlış optimizasyon yeni hatalara neden olabilir.
- **Zaman ve Maliyet:** Optimizasyon süreci ek zaman ve maliyet gerektirebilir.
- **Performans-Tasarruf Dengesi:** Performansı artırırken kodun okunabilirliği ve bakımı zorlaşabilir.
- **Ölçüm Hataları:** Optimizasyon öncesi ve sonrası etkileri doğru ölçümlemek zor olabilir.

Kodlama - 6. Optimizasyon

6. Optimizasyon İşlemi İçin En İyi Uygulamalar

1.Önceliklendirme: En kritik sorunlara odaklanın ve yazılımın kullanıcı deneyimini en çok etkileyen darboğazları çözün.

2.İzleme ve Analiz: Performans sorunlarını düzenli olarak analiz edin ve optimize edin.

3.Modüler Yaklaşım: Kodun optimize edilmesi gerektiğinde, modüler bir yaklaşım benimseyerek işlemi daha kolay hale getirin.

4.Test Otomasyonu: Optimizasyon sonrası yazılımın doğru çalıştığından emin olmak için test süreçlerini otomatikleştirin.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Kodun Gözden Geçirilmesi (Code Review), yazılım geliştirme sürecinde, bir kişinin veya bir ekip üyesinin yazdığı kodun başka bir geliştirici ya da ekip tarafından incelenmesi işlemidir.

Bu süreç, kodun kalitesini artırmayı, hataları en aza indirmeyi ve kodun standartlara uygunluğunu sağlamayı hedefler.

Kodun gözden geçirilmesi, yazılım projelerinin sürdürülebilirliği ve ekip içi iş birliği açısından önemli bir adımdır.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

1. Kodun Gözden Geçirilmesinin Amaçları

- **Hata Tespiti:** Kodun hatalara karşı kontrol edilmesi ve hataların erkenden tespit edilmesi.
- **Kod Kalitesini Artırma:** Kodun okunabilir, sürdürülebilir ve modüler olmasını sağlama.
- **Standartlara Uygunluk:** Kodun belirlenen kodlama standartlarına ve en iyi uygulamalara uygunluğunu kontrol etme.
- **Ekip Öğrenimi ve İş Birliği:** Geliştiriciler arasında bilgi paylaşımını teşvik etme ve daha iyi çözümler geliştirme.
- **Teknik Borcun Azaltılması:** Gelecekteki bakım sürecini kolaylaştırmak için kodun düzenli olmasını sağlama.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

2. Kod Gözden Geçirme Türleri

a) Resmi Kod Gözden Geçirme (Formal Code Review)

•Detaylı ve yapılandırılmış bir inceleme sürecidir.

•Belirli bir prosedür izlenir; örneğin, kod belgeleri hazırlanır ve ekip bir toplantıda kodu tartışır.

•Daha çok büyük projelerde ve kritik sistemlerde uygulanır.

b) Gayri Resmi Kod Gözden Geçirme (Informal Code Review)

•Geliştiriciler arasında genellikle kısa bir toplantı veya birebir yapılan incelemelerdir.

•Daha hızlı ve esnek bir süreçtir.

•Örnek: Bir geliştiricinin kodunu yazdıktan sonra başka bir geliştiriciye göstermesi.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

c) Çift Programlama (Pair Programming)

•İki geliştiricinin aynı anda aynı kod üzerinde çalıştığı süreçtir.

•Kod yazılırken eş zamanlı olarak incelenir.

•Özellikle Agile metodolojilerinde sıkça kullanılır.

d) Araç Destekli Gözden Geçirme (Tool-Assisted Code Review)

•Kod gözden geçirme araçları kullanılarak yapılan inceleme.

•Örnek araçlar: GitHub Pull Requests, Bitbucket, GitLab, Crucible.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

3. Kod Gözden Geçirme Süreci

Adım 1: Kodun Hazırlanması

•Kodun düzgün bir şekilde formatlanması ve gereksiz dosyaların temizlenmesi.

•Kodun belgelendirilmesi; bu, gözden geçiren kişinin süreci anlamasını kolaylaştırır.

•Kodun çalışma şeklinin kısa bir özeti hazırlanır.

Adım 2: Gözden Geçirilecek Kodun Seçimi

•Küçük ve odaklanmış bir kod parçacığı seçilir (örneğin, bir fonksiyon veya modül).

•Büyük kod parçalarının gözden geçirilmesi daha zor ve zaman alıcıdır.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Adım 3: Gözden Geçiricinin Belirlenmesi

•Kodun, proje bağlamını iyi bilen bir ekip üyesi veya teknik lider tarafından incelenmesi tercih edilir.

•Bazı durumlarda, ekip dışından bağımsız bir uzman da bu sürece dahil edilebilir.

Adım 4: Kodun İncelenmesi

•Gözden geçiren kişi aşağıdaki unsurları kontrol eder:

- **Kod Kalitesi:** Kod okunabilir mi, anlaşılırlığı yüksek mi?
- **Hata ve Sorunlar:** Mantıksal hatalar veya eksiklikler var mı?
- **Performans:** Kod verimli çalışıyor mu, daha iyi bir çözüm uygulanabilir mi?
- **Güvenlik:** Güvenlik açıkları veya tehditler içeriyor mu?
- **Uyumluluk:** Kod, ekip standartlarına veya projedeki diğer kodlara uyumlu mu?

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Adım 5: Geri Bildirim Verme

•Gözden geçirme sonuçları geliştiriciye geri bildirilir.

•Geri bildirimler yapıcı ve açıklayıcı olmalıdır.

•Örnek bir geri bildirim:

- **İyi:** “Burada bu algoritma doğru çalışıyor, ancak performans açısından şu alternatif algoritmayı düşünebilirsin.”
- **Kötü:** “Bu kod yanlış, baştan yap.”

Adım 6: Revizyon

•Geliştirici, gözden geçirme sırasında belirtilen sorunları veya eksiklikleri düzeltir.

•Revize edilen kod tekrar gözden geçirilir.

Adım 7: Onay ve Birleştirme

•Kod, gerekli düzenlemeler yapıldıktan sonra onaylanır ve kod deposuna (repository) birleştirilir.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

4. Kod Gözden Geçirme Sürecinde Kullanılan Araçlar

Araç/Platform	Amacı
GitHub Pull Request	Kod değişikliklerini inceleme ve yorumlama
GitLab Merge Request	Kod birleşme işlemlerini yönetme
Crucible	Resmi kod inceleme süreci sağlama
Phabricator	Ekip bazlı gözden geçirme ve yönetim
SonarQube	Kod kalitesi ve güvenliği analiz etme

Kodlama - 8. Dökümantasyon

Kodlama Adımında Dökümantasyon İşlemi

Dökümantasyon, yazılım geliştirme sürecinde, yazılımın işleyişi, yapısı, kullanım kılavuzları ve teknik detaylarının yazılı olarak kaydedilmesi işlemidir.

Kodlama aşamasında yapılan dökümantasyon, kodun anlaşılabilirliğini artırır, ekip içi iletişimi kolaylaştırır ve yazılımın bakımını ve geliştirilmesini daha sürdürülebilir hale getirir.

Kodlama - 8. Dökümantasyon

1. Dökümantasyonun Amacı

•**Kodun Anlaşılabilirliğini Artırmak:** Kodun işlevlerini ve yapısını kolayca anlamak için açıklamalar sağlamak.

•**Ekip İşbirliğini Kolaylaştırmak:** Yeni ekip üyelerinin projeye hızlı bir şekilde uyum sağlamasına yardımcı olmak.

•**Bakımı Kolaylaştırmak:** Yazılımın ileride güncellenmesi, genişletilmesi veya düzeltilmesi gerektiğinde süreçleri hızlandırmak.

•**Kullanıcı Rehberi Sağlamak:** Son kullanıcılar için yazılımın nasıl kullanılacağını açıklamak.

•**Yasal ve Uyumluluk Gereksinimlerini Karşılamak:** Belirli sektörlerde yasal düzenlemelere uygunluğu sağlamak için teknik belgeler oluşturmak.

Kodlama - 8. Dökümantasyon

2. Dökümantasyon Türleri

a) Kod Düzeyinde Dökümantasyon

•Kod içerisine eklenen açıklamalar ve yorumlardır.

•Örnekler:

- Fonksiyon açıklamaları (ne işe yarar, hangi parametreleri alır, ne döndürür).
- Karmaşık algoritmaların açıklamaları.
- Kullanılan kütüphaneler ve modüller hakkında bilgi.

b) Teknik Dökümantasyon

•Yazılımın teknik yapısını ve mimarisini açıklar.

•İçerik:

- Sistem mimarisi (modüller, veri akışları, sistem bileşenleri).
- Kullanılan teknolojiler.
- API dokümantasyonu (endpointler, veri formatları, yanıtlar).
- Veri tabanı şemaları.

Kodlama - 8. Dökümantasyon

c) Kullanıcı Dökümantasyonu

•Son kullanıcıların yazılımı doğru şekilde kullanabilmesi için hazırlanır.

•İçerik:

- Kurulum kılavuzları.
- Kullanım talimatları.
- Yaygın hatalar ve çözümleri (SSS bölümü).

d) Proje Yönetimi Dökümantasyonu

•Yazılım geliştirme sürecindeki işlerin planlanması ve takibi için oluşturulur.

•İçerik:

- İş gereksinimleri ve kapsam belgeleri.
- Proje zaman çizelgesi.
- Sprint notları (Agile projelerde).
- Test planları ve sonuçları.

Kodlama - 8. Dökümantasyon

e) Bakım ve Destek Dökümantasyonu

•Yazılımın bakımını üstlenecek kişiler için teknik detaylar ve yönergeler sağlar.

•İçerik:

- Güncellenmesi gereken modüller.
- Log dosyalarının analizi.
- Performans sorunlarını çözme adımları.

Kodlama - 8. Dökümantasyon

3. Dökümantasyon Süreci

Adım 1: Dökümantasyon Planlama

•Projenin kapsamına uygun dökümantasyon türleri belirlenir.

•Hangi dökümantasyon türlerinin kimler tarafından oluşturulacağı ve ne zaman tamamlanacağı planlanır.

Adım 2: İçerik Hazırlama

•**Hedef Kitle:** Dökümantasyonun, okuyucunun bilgi düzeyine uygun bir dil ve ayrıntı seviyesi kullanılarak hazırlanması gerekir.

- Örneğin, teknik belgeler mühendisler için yazılırken kullanıcı kılavuzları daha sade bir dilde yazılır.

•Kod ile ilgili temel bilgilerin yanında, yazılımın genel amacı ve işlevselliği hakkında bilgiler eklenir.

Kodlama - 8. Dökümantasyon

Adım 3: Görselleştirme

•Karmaşık süreçlerin daha kolay anlaşılması için görsel araçlar kullanılabilir:

- Diyagramlar (örneğin, UML diyagramları).
- Akış şemaları.
- Örnek ekran görüntüleri.

Adım 4: Revizyon ve Güncelleme

•Dökümantasyon, düzenli olarak gözden geçirilir ve yazılımın güncellenmesi durumunda ilgili bölümler değiştirilir.

•Dökümantasyonun eski veya hatalı bilgiler içermediğinden emin olunur.

Adım 5: Paylaşım

•Dökümantasyon ekip üyeleri, son kullanıcılar veya ilgili taraflarla paylaşılır.

•Paylaşım platformları: Confluence, Notion, Google Docs, GitHub Wiki.

Kodlama - 8. Dökümantasyon

4. Dökümantasyon Sürecinde Kullanılan Araçlar

Araç/Platform	Amacı
Markdown	Basit ve okunabilir teknik dokümanlar yazma.
Doxygen	Koddan otomatik dökümantasyon oluşturma.
Swagger	API dökümantasyonlarını oluşturma.
Confluence, Notion	Proje belgelerini merkezi bir yerde tutma.
PlantUML, Lucidchart	Diyagramlar ve akış şemaları oluşturma.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Test Etme

- Yazılım yaşam döngüsünde **test etme adımı**, geliştirilen yazılımın belirlenen gereksinimlere, tasarım dokümanlarına ve kullanıcı beklentilerine uygun olup olmadığını doğrulamak amacıyla yapılan çalışmaları kapsar.

- Test etme süreci, yazılımın hatalarını bulmayı ve çözmeyi, böylece kaliteli ve güvenilir bir ürün elde etmeyi hedefler.

Test Etme

✓ Test Etme Adımının Detayları

- Test Planlama
- Test Tasarımı
- Test Ortamının Hazırlanması
- Testlerin Gerçekleştirilmesi
- Hata Düzeltme ve Yeniden Test
- Testlerin Sonlandırılması ve Değerlendirme

Test Etme

- ✓ **Test Planlama**
- **Amaç:** Test sürecinin kapsamını, yöntemlerini, araçlarını ve takvimini belirlemek.
- **Faaliyetler:**
  - Test stratejisi oluşturulur.
  - Kaynaklar (ekip ve araçlar) planlanır.
  - Test senaryoları ve test ortamları tanımlanır.
- **Sonuç:** Bir test planı dokümanı hazırlanır.

Test Etme

- **Örnek Test Planı**
  - **Proje Adı:**
  - E-Ticaret Platformu Geliştirme Projesi
  - **Test Planı Versiyonu:**
  - 1.0
  - **Hazırlayan:**
  - Test Ekibi Lideri - [İsim]
  - **Tarih:**
  - [xx/xx/xxxx]

Test Etme

- ✓ **Örnek Test Planı**

1. **Amaç**

Bu test planı, e-ticaret platformunun geliştirme sürecindeki test faaliyetlerini düzenlemek ve yazılımın gereksinimlere uygun çalıştığını doğrulamak amacıyla hazırlanmıştır.

Test Etme

✓ **Örnek Test Planı**

2. Test Kapsamı

✓ **Test Edilecek Modüller:**

- Kullanıcı Yönetimi (kayıt, giriş, profil güncelleme)
- Ürün Arama ve Kategorilendirme
- Sepet ve Ödeme Süreci
- Sipariş Yönetimi
- Raporlama ve İstatistikler

✓ **Test Edilmeyecek Alanlar:**

- Üçüncü taraf entegrasyonlarının iç mimarisi
- Üçüncü taraf ödeme sağlayıcıların harici sistemleri

Test Etme

✓ **Örnek Test Planı**

✓ **3. Test Stratejisi**

- **Test Türleri:**
  - Fonksiyonel Testler
  - Performans Testleri
  - Güvenlik Testleri
  - Kullanıcı Kabul Testleri
- **Test Yaklaşımları:**
  - Manuel test senaryoları uygulanacaktır.
  - Kritik fonksiyonlar için otomasyon test araçları kullanılacaktır (ör. Selenium).
- **Test Seviyeleri:**
  - Birim Testleri: Geliştirici ekip tarafından yapılacaktır.
  - Entegrasyon Testleri: Test ekibi tarafından yapılacaktır.
  - Sistem Testleri: Tüm modüller tamamlandıktan sonra yapılacaktır.
  - Kabul Testleri: Müşteri temsilcileri tarafından yapılacaktır.

Test Etme

- ✓ **Örnek Test Planı**

- ✓ **4. Test Ortamı ve Gereksinimler**
- ✓ **Ortam:**
- Test Sunucusu: [test.example.com]
- Tarayıcılar: Chrome, Firefox, Safari, Edge (son sürümler)
- Mobil Test: Android (v10+), iOS (v14+)
- ✓ **Gereksinimler:**
- Test veritabanı (dummy kullanıcı ve ürün kayıtları)
- Sanal test ortamında test kullanıcı hesapları

Test Etme

✓ **Örnek Test Planı**

✓ **5. Kaynaklar**

✓ **Ekip Üyeleri:**

- Test Ekibi Lideri: [İsim]
- QA Mühendisi: [İsim]
- Otomasyon Uzmanı: [İsim]

✓ **Araçlar:**

- Test Yönetim Aracı: JIRA
- Otomasyon Test Aracı: Selenium
- Performans Test Aracı: Apache JMeter

Test Etme

6. Zaman Çizelgesi

Aktivite	Başlangıç Tarihi	Bitiş Tarihi	Sorumlu
Test Planlama	[xx/xx/xxxx]	[xx/xx/xxxx]	Test Ekibi Lideri
Test Tasarımı	[xx/xx/xxxx]	[xx/xx/xxxx]	QA Mühendisi
Test Ortamı Kurulumu	[xx/xx/xxxx]	[xx/xx/xxxx]	Otomasyon Uzmanı
Testlerin Gerçekleştirilmesi	[xx/xx/xxxx]	[xx/xx/xxxx]	QA Ekibi
Test Sonlandırma	[xx/xx/xxxx]	[xx/xx/xxxx]	Test Ekibi Lideri

Test Etme

7. Riskler ve Önlemler

Risk	Etkisi	Önlem
Test ortamında gecikme	Yüksek	Ortamın erken kurulumu ve test edilmesi
Gereksinim değişiklikleri	Orta	Gereksinimlere uygunluk düzenli gözden geçirilecek
Yetersiz test verisi	Orta	Veriler önceden hazırlanacak ve kontrol edilecek

Test Etme

- ✓ **Örnek Test Planı**

- ✓ **8. Test Kriterleri**
- ✓ **Başarı Kriterleri:**
  - Kritik fonksiyonların %100 başarılı olması
  - Önemli olmayan hataların düzeltilebilir seviyede olması
- ✓ **Başarısızlık Kriterleri:**
  - Kritik veya yüksek önem düzeyindeki hataların varlığı
  - Performans kriterlerinin karşılanmaması

Test Etme

✓ **Örnek Test Planı**

✓ **9. Test Çıkış Kriterleri**

- Tüm planlanan testlerin tamamlanmış ve raporlanmış olması
- Kritik ve yüksek öncelikli hataların çözülmüş olması

Test Etme

✓ **Test Tasarımı**

- **Amaç:** Test senaryolarını ve test vakalarını detaylandırmak.
- **Faaliyetler:**
  - Gereksinimlerden yola çıkarak test vakaları oluşturulur.
  - Kullanıcı senaryoları baz alınır.
  - Test veri setleri hazırlanır.
- **Sonuç:** Testlerin çalıştırılacağı durumları içeren detaylı bir doküman.

Test Etme

- **Örnek Test Tasarım Raporu**
- **Proje Adı:**
- E-Ticaret Platformu
- **Hazırlayan:**
- Test Ekibi Lideri - [İsim]
- **Tarih:**
- [xx/xx/xxxx]

Test Etme

- **Örnek Test Tasarım Raporu**
- **1. Amaç**
- Bu rapor, e-ticaret platformunun test aşamasında kullanılacak test senaryolarını ve test vakalarını detaylandırmak için hazırlanmıştır. Test tasarımı, yazılımın işlevsel ve performans gereksinimlerini karşılayıp karşılamadığını değerlendirmek için temel bir kılavuz sağlar.

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **2. Kapsam**
- Test Edilecek Modüller:
  - Kullanıcı Yönetimi
  - Ürün Arama
  - Sepet ve Ödeme Süreci
- Test Edilmeyecek Alanlar:
  - Üçüncü taraf API'ler

Test Etme

3. Test Senaryoları

Test Senaryosu ID	Senaryo Açıklaması	Ön Koşullar
TS001	Kullanıcının başarıyla giriş yapması	Kullanıcı kayıtlı olmalı, doğru bilgiler girmeli
TS002	Ürünlerin kategoriye göre aranması	Ürünler kategorilere atanmış olmalı
TS003	Sepete ürün eklenmesi	Kullanıcı giriş yapmış ve ürün sayfasında olmalı
TS004	Ödeme işleminin başarıyla tamamlanması	Kullanıcının kredi kartı bilgileri doğru olmalı

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **4. Test Vakaları**
- ✓ **Test Vakası ID: TC001**
- **Senaryo:** Kullanıcının giriş yapması
- **Girdi Verisi:**
  - Kullanıcı Adı: testuser@example.com
  - Şifre: Test1234
- **Adımlar:**
  - Giriş sayfasına gidilir.
  - Kullanıcı adı ve şifre girilir.
  - "Giriş Yap" butonuna tıklanır.

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **4. Test Vakaları**
- **Beklenen Sonuç:**
  Kullanıcı ana sayfaya yönlendirilir ve profil bilgileri görüntülenir.
- ✓ **Test Vakası ID: TC002**
- **Senaryo:** Ürünlerin kategoriye göre aranması
- **Girdi Verisi:**
  - Kategori: Elektronik
- **Adımlar:**
  - Anasayfadan "Kategoriler" menüsü seçilir.
  - "Elektronik" kategorisi seçilir.

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **4. Test Vakaları**
- **Beklenen Sonuç:**
  Elektronik kategorisine ait ürünler listelenir.
- ✓ **Test Vakası ID: TC003**
- **Senaryo:** Sepete ürün eklenmesi
- **Girdi Verisi:**
  - Ürün: Akıllı Telefon
- **Adımlar:**
  - Ürün sayfasına gidilir.
  - "Sepete Ekle" butonuna tıklanır.

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **4. Test Vakaları**
- **Beklenen Sonuç:**
  Ürün sepete eklenir ve "Ürün sepetinize eklendi" mesajı görüntülenir.
- ✓ **Test Vakası ID: TC004**
- **Senaryo:** Ödeme işlemi
- **Girdi Verisi:**
  - Kart Bilgileri: 1234-5678-9012-3456, 12/26, 123
- **Adımlar:**
  - Sepet sayfasına gidilir.
  - Ödeme bilgileri girilir.
  - "Ödemeyi Tamamla" butonuna tıklanır.
- **Beklenen Sonuç:**
  "Ödeme başarıyla tamamlandı" mesajı görüntülenir.

Test Etme

- ✓ Örnek Test Tasarım Raporu

5. Test Verileri

Veri Seti ID	Açıklama	Örnek Veri
DS001	Kullanıcı bilgileri	testuser@example.com, Test1234
DS002	Ürün kategorileri	Elektronik, Moda, Kitap
DS003	Kredi Kartı Bilgileri	1234-5678-9012-3456, 12/26, 123

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **6. Test Kriterleri**
- ✓ **Başarı Kriterleri:**
  - Tüm senaryoların %90'ı başarıyla çalışmalıdır.
  - Kritik işlevlerde herhangi bir hata olmamalıdır.
- ✓ **Başarısızlık Kriterleri:**
  - Kritik işlevlerden birinin çalışmaması.
  - Test senaryolarının %10'undan fazlasının başarısız olması.

Test Etme

- ✓ **Örnek Test Tasarım Raporu**
- ✓ **7. Araçlar ve Ortam**
- **Araçlar:** Selenium, Postman
- **Ortam:**
  - Test Sunucusu: test.example.com
  - Tarayıcı: Chrome (v116+), Firefox (v112+)
  - Mobil: Android 11, iOS 14

Test Etme

- ✓ **Test Ortamının Hazırlanması**
- **Amaç:** Testlerin yürütüleceği uygun bir ortam hazırlamak.
- **Faaliyetler:**
  - Yazılımın çalışacağı platform veya cihazlar yapılandırılır.
  - Test araçları (örneğin otomasyon araçları) kurulur.
  - Test verileri ve kullanıcı hesapları oluşturulur.
- **Sonuç:** Yazılımın gerçek koşullarda test edilebileceği bir ortam.

Test Etme

✓ **Testlerin Gerçekleştirilmesi**

- **Amaç:** Yazılımın işlevselliklerini, performansını, güvenliğini ve diğer kalite faktörlerini değerlendirmek.
- **Faaliyetler:**
  - Test senaryoları manuel veya otomatik olarak yürütülür.
  - Hatalar kaydedilir ve raporlanır.
  - Bulunan hatalar ilgili ekibe iletilir.
- **Sonuç:** Test raporları ve hata takip kayıtları.

Test Etme

- **Örnek Test Gerçekleştirme Raporu**
- **Proje Adı:**
- E-Ticaret Platformu
- **Hazırlayan:**
- QA Mühendisi - [İsim]
- **Tarih:**
- [xx/xx/xxxx]

Test Etme

- **Örnek Test Gerçekleştirme Raporu**
- **1. Amaç**
- Bu rapor, e-ticaret platformunun test gerçekleştirme aşamasında yapılan testlerin sonuçlarını ve gözlemleri belgelemek için hazırlanmıştır. Amaç, test senaryolarının doğru bir şekilde uygulanıp uygulanmadığını, bulunan hataları ve çözüm süreçlerini raporlamaktır.

Test Etme

- ✓ **Örnek Test Gerçekleştirme Raporu**
- ✓ **2. Test Süreci Özeti**
- **Toplam Test Vakası:** 10
- **Başarıyla Geçen:** 8
- **Başarısız Olan:** 2
- **Test Başarı Oranı:** %80

3. Test Edilen Senaryolar ve Sonuçlar

Test Vakası ID	Senaryo Açıklaması	Durum	Gözlem
TC001	Kullanıcının giriş yapması	Başarılı	Giriş işlemi doğru bilgilerle başarılı şekilde gerçekleşti.
TC002	Ürünlerin kategoriye göre aranması	Başarılı	Tüm kategoriler doğru şekilde listelendi.
TC003	Sepete ürün eklenmesi	Başarılı	Ürün sepete başarıyla eklendi ve bildirim gösterildi.
TC004	Ödeme işleminin tamamlanması	Başarısız	Ödeme sayfasında "Geçersiz Kart Bilgisi" hatası alındı.
TC005	Kullanıcı çıkış yapması	Başarılı	Kullanıcı sistemden başarıyla çıkış yaptı.
TC006	Sipariş geçmişinin görüntülenmesi	Başarısız	Sipariş geçmişi yüklenirken sayfa yanıt vermedi.
TC007	Şifre sıfırlama işlemi	Başarılı	Şifre sıfırlama e-postası başarıyla gönderildi.
TC008	Hızlı arama fonksiyonu	Başarılı	Arama sonuçları hızlı ve doğru şekilde listelendi.
TC009	Sepet içeriğinin güncellenmesi	Başarılı	Ürün miktarı başarıyla güncellendi.
TC010	Kullanıcı profil bilgileri düzenlemesi	↓ şarılı	Kullanıcı bilgileri sorunsuz şekilde güncellendi.

Test Etme

- ✓ **Örnek Test Gerçekleştirme Raporu**
- ✓ **4. Test Hataları**
- ✓ **Hata ID: BUG001**
- **Test Vakası ID:** TC004
- **Hata Açıklaması:**
  Ödeme sayfasında geçerli kredi kartı bilgileri girildiğinde "Geçersiz Kart Bilgisi" hatası alınıyor.
- **Hata Seviyesi:** Kritik
- **Çözüm Durumu:** Geliştirici ekibe iletildi. Çözüm bekleniyor.
- ✓ **Hata ID: BUG002**
- **Test Vakası ID:** TC006
- **Hata Açıklaması:**
  Sipariş geçmişi sayfası yüklenirken "504 Gateway Timeout" hatası alınıyor.
- **Hata Seviyesi:** Yüksek
- **Çözüm Durumu:** Geliştirici ekibe iletildi. Sunucu yanıt süreleri optimize edilecek.

Test Etme

- ✓ **Örnek Test Gerçekleştirme Raporu**
- ✓ **5. Test Performansı**
- **Toplam Test Süresi:** 4 saat
- **Manuel Test Oranı:** %70
- **Otomasyon Kullanım Oranı:** %30

Test Etme

- ✓ **Örnek Test Gerçekleştirme Raporu**
- ✓ **6. Sonuçlar ve Öneriler**
- ✓ **Sonuçlar:**
- Çoğu test vakası başarıyla geçti ve yazılımın temel işlevsellikleri doğru çalışıyor.
- Kritik hatalar (TC004 ve TC006) müşteri deneyimini doğrudan etkileyebileceğinden öncelikli olarak çözülmelidir.
- ✓ **Öneriler:**

1. Ödeme altyapısındaki hataların çözümü için geliştirme ekibiyle detaylı bir analiz yapılmalı.
2. Sipariş geçmişi sayfasının yanıt süresi optimize edilmelidir.
3. Performans testleri artırılarak yüksek yük altındaki sistemin tepkisi test edilmelidir.

Test Etme

- ✓ **Örnek Test Gerçekleştirme Raporu**
- ✓ **7-Ekler**
- **Ekran Görüntüleri:** Hatalarla ilgili ekran görüntüleri ekte sunulmuştur.
- **Hata Takip Dokümanı:** JIRA üzerinden takip edilen hata kayıtlarının bağlantısı: [JIRA Bağlantısı]

Test Etme

- ✓ **Hata Düzeltme ve Yeniden Test**
- **Amaç:** Tespit edilen hataların giderildiğini doğrulamak.
- **Faaliyetler:**
  - Geliştiriciler hataları düzeltir.
  - Hataların düzeltilip düzeltilmediği test edilir (yeniden test).
  - Düzeltmelerin başka sorunlara yol açıp açmadığı kontrol edilir (regresyon testi).
- **Sonuç:** Gözden geçirilmiş ve doğrulanmış bir yazılım.

Test Etme

✓ **Testlerin Sonlandırılması ve Değerlendirme**

- **Amaç:** Test sürecinin başarıyla tamamlandığını doğrulamak ve yazılımın yayına hazır olduğunu belgelemek.
- **Faaliyetler:**
  - Test sonuçları analiz edilir.
  - Kalan riskler değerlendirilir.
  - Test kapanış raporu hazırlanır.
- **Sonuç:** Yazılımın belirlenen kriterlere uygun olduğunu gösteren onay.

Test Etme

✓ **Testlerin Sonlandırılması ve Değerlendirme**

- **Amaç:** Test sürecinin başarıyla tamamlandığını doğrulamak ve yazılımın yayına hazır olduğunu belgelemek.
- **Faaliyetler:**
  - Test sonuçları analiz edilir.
  - Kalan riskler değerlendirilir.
  - Test kapanış raporu hazırlanır.
- **Sonuç:** Yazılımın belirlenen kriterlere uygun olduğunu gösteren onay.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Kurulum (Deployment)

- **Kurulum (Deployment)** adımı, yazılım geliştirme sürecinin önemli bir dönüm noktasıdır.
- Bu adımda, geliştirilen yazılım, kullanıcıların erişimine sunulacak şekilde gerçek bir çalışma ortamına taşınır.
- Bu süreç, sadece yazılımın kullanıcıya ulaştırılmasını değil, aynı zamanda kullanıma hazır hale getirilmesi için gerekli tüm faaliyetleri kapsar.

Kurulum (Deployment)

- **Kurulum (Deployment)** adımının detayları:

- 1. Çalışma Ortamının Hazırlanması
- 2. Versiyonlama ve Dağıtım
- 3. Veri Geçişi
- 4. Test ve Onay
- 5. Eğitim ve Dökümantasyon
- 6. Canlıya Geçiş (Go-Live)
- 7. İzleme ve İlk Destek

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması
- ✓ **Çalışma ortamının hazırlanması**, yazılımın doğru ve verimli bir şekilde çalışabilmesi için gerekli teknik altyapının ve sistem kaynaklarının hazırlanması sürecidir.
- ✓ Bu aşama, yazılımın hedeflenen çalışma ortamında sorunsuz bir şekilde çalışmasını sağlamak amacıyla hem donanımsal hem de yazılımsal yapılandırmaları kapsar.

Kurulum (Deployment)

✓ Çalışma Ortamının Hazırlanması

✓ **1. Donanım Altyapısının Hazırlanması**

- **Sunucular:** Yazılımın çalıştırılacağı fiziksel veya sanal sunucular yapılandırılır. Performans ihtiyaçlarına göre işlemci (CPU), bellek (RAM), disk alanı gibi kaynaklar optimize edilir.
- **Ağ Yapısı:** Yazılımın çalışacağı ağ ortamı, kullanıcıların erişimini sağlayacak şekilde yapılandırılır. Gerektiğinde yük dengeleme (load balancing) ve ağ güvenlik duvarları (firewall) eklenir.
- **Yedekleme Sistemleri:** Çalışma ortamının güvenliğini sağlamak için veri yedekleme mekanizmaları kurulur.

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **2. Yazılım Altyapısının Kurulumu**
- **İşletim Sistemi:** Yazılımın gerektirdiği işletim sistemi (Windows, Linux vb.) sunuculara yüklenir ve optimize edilir.
- **Ortam Yazılımları:** Yazılımın çalışması için gerekli olan:
  - **Web sunucusu** (ör. Apache, NGINX, IIS),
  - **Uygulama sunucusu** (ör. Tomcat, Wildfly),
  - **Veritabanı yönetim sistemi** (ör. MySQL, PostgreSQL, Oracle) gibi bileşenler kurulur.
- **Kütüphane ve Bağımlılıklar:** Yazılımın çalışması için gereken tüm bağımlılıklar ve kütüphaneler yüklenir ve güncel sürümlerde olduğundan emin olunur.

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **3. Güvenlik Önlemleri**
- **Kullanıcı ve Yetki Yönetimi:** Ortama erişebilecek kullanıcılar için rol tabanlı erişim kontrolü (RBAC) yapılandırılır.
- **Şifreleme:** Verilerin güvenliği için gerekli şifreleme protokolleri (ör. TLS, SSL) devreye alınır.
- **Güvenlik Duvarları ve IP Kısıtlamaları:** Yetkisiz erişimlerin önüne geçmek için güvenlik duvarları ve IP kısıtlamaları uygulanır.

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **4. Test Ortamının Oluşturulması**
- Üretim ortamına geçmeden önce yazılım, test ortamında çalıştırılarak test edilir. Bu ortam, üretim ortamına benzer şekilde yapılandırılır.
- Yazılım performansı, veri işleme kapasitesi, güvenlik önlemleri gibi unsurlar test edilir.

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **5. Performans ve Kapasite Planlaması**
- **Yük Testleri:** Sistemin, beklenen kullanıcı trafiği altında nasıl çalıştığını ölçmek için yük ve stres testleri yapılır.
- **Kapasite Artırımı:** İleride artabilecek kullanıcı trafiğine karşı ölçeklenebilirlik özellikleri planlanır (ör. daha fazla sunucu eklenmesi).

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **6. Çevresel Entegrasyon**
- Yazılımın entegre çalışacağı diğer sistemler (ör. API'ler, üçüncü taraf hizmetler) ile uyumlu bir şekilde çalışabilmesi için ortam ayarları yapılır.
- Ağ bağlantıları ve veri akışı düzenlenir.

Kurulum (Deployment)

- ✓ Çalışma Ortamının Hazırlanması

- ✓ **7. Güncelleme ve Yama Yönetimi**
- İşletim sistemi, ortam yazılımları ve güvenlik araçları için gerekli güncellemeler yapılır.
- Güvenlik açıklarını kapatmak ve uyumluluk sorunlarını gidermek için yamalar uygulanır.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ Yazılım geliştirme sürecinde, **Versiyonlama ve Dağıtım**, yazılımın stabil bir sürümünün belirlenmesi ve bu sürümün hedef kullanıcılara veya sistemlere ulaştırılması işlemini kapsar.
- ✓ Bu süreç, yazılımın tutarlı bir şekilde güncellenmesini, izlenmesini ve kullanıcılar arasında sorunsuz bir şekilde yayılmasını sağlar.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım

- ✓ **1. Versiyonlama Süreci**
- ✓ Versiyonlama, yazılımın farklı sürümlerinin yönetilmesini ve takip edilmesini sağlar. Yazılım geliştirme sırasında her sürüm, belirli bir numaralandırma veya adlandırma sistemiyle tanımlanır.

- ✓ **a. Versiyonlama Neden Önemlidir?**
- Yazılım değişikliklerini takip etmeyi kolaylaştırır.
- Geliştirici ekipler arasında iletişimi ve iş birliğini güçlendirir.
- Hataların hangi sürümde ortaya çıktığını izleme olanağı sağlar.
- Kullanıcıların hangi sürümü kullandığını anlamayı kolaylaştırır.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **1. Versiyonlama Süreci**
- ✓ **b. Versiyonlama Formatı**
- ✓ Versiyonlama genellikle **Semantik Versiyonlama (Semantic Versioning)** kurallarına uygun şekilde yapılır:

- **Ana Sürüm (Major):** Büyük değişiklikler ve geriye dönük uyumluluğu bozan güncellemeler. Örnek: 1.0.0 → 2.0.0
- **Alt Sürüm (Minor):** Yeni özellikler eklenir ancak geriye dönük uyumluluk korunur. Örnek: 1.1.0 → 1.2.0
- **Düzeltme Sürümü (Patch):** Hata düzeltmeleri veya küçük iyileştirmeler. Örnek: 1.1.1 → 1.1.2

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **1. Versiyonlama Süreci**
- ✓ **c. Stabil Sürüm Hazırlığı**
  - Yazılım geliştirme tamamlandıktan sonra, çeşitli testlerden geçirilir.
  - Testlerden başarıyla geçen sürüm, **Release Candidate (RC)** olarak belirlenir.
  - Kullanıcı onay testleri (UAT) tamamlandıktan sonra sürüm stabil hale gelir ve resmi olarak yayınlanır.

Kurulum (Deployment)

- Versiyonlama ve Dağıtım
- **2. Dağıtım Süreci**

- Dağıtım (Deployment), yazılımın belirli bir sürümünün, kullanıcıların erişimine sunulmasını ve çalışır hale getirilmesini ifade eder. Dağıtım süreci, yazılımın türüne ve kullanım senaryosuna göre değişiklik gösterebilir.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **2. Dağıtım Süreci**

- ✓ **a. Dağıtım Türleri**
- **Yerinde Dağıtım (On-Premise Deployment):** Yazılım, müşteri tarafından kontrol edilen sunuculara veya cihazlara yüklenir.
  - Müşteri ortamına özel konfigürasyonlar yapılır.
  - Veri tabanı bağlantıları ve ağ yapılandırmaları müşteri ihtiyaçlarına göre düzenlenir.
- **Bulut Tabanlı Dağıtım (Cloud Deployment):** Yazılım, bir bulut platformu (AWS, Azure, Google Cloud vb.) üzerinde çalışır ve kullanıcılar internet üzerinden erişir.
  - Daha hızlı ve kolay bir dağıtım sağlar.
  - Yazılımın ölçeklenebilirliği artırılır.
- **Sürekli Dağıtım (Continuous Deployment):** Yazılım geliştirme ve güncellemeler, otomatize bir şekilde sürekli olarak kullanıcılara sunulur.
  - **CI/CD (Continuous Integration/Continuous Deployment)** sistemleri bu süreçte kullanılır.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **2. Dağıtım Süreci**

- ✓ **b. Dağıtım Adımları**

- ❖ **Dağıtım Stratejisinin Belirlenmesi**
- ❖ **Çalışma Ortamına Yükleme**
- ❖ **Yedekleme ve Geri Dönüş Planı**
- ❖ **Son Testler**

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **2. Dağıtım Süreci**

- ✓ **c. Dağıtım Sonrası İzleme**

- **Performans İzleme:** Dağıtımın ardından sistem performansı yakından takip edilir.
- **Hata Yönetimi:** Karşılaşılan hatalar hızla çözülür.
- **Kullanıcı Dönüşleri:** Kullanıcılardan alınan geri bildirimler değerlendirilir ve gerekirse iyileştirmeler yapılır.

Kurulum (Deployment)

- ✓ Versiyonlama ve Dağıtım
- ✓ **2. Dağıtım Süreci**

- ✓ **c. Dağıtım Sonrası İzleme**

- **Performans İzleme:** Dağıtımın ardından sistem performansı yakından takip edilir.
- **Hata Yönetimi:** Karşılaşılan hatalar hızla çözülür.
- **Kullanıcı Dönüşleri:** Kullanıcılardan alınan geri bildirimler değerlendirilir ve gerekirse iyileştirmeler yapılır.

Kurulum (Deployment)

- Veri Geçişi

- **Veri Geçişi**, bir sistemdeki mevcut verilerin başka bir sisteme taşınması, dönüştürülmesi ve yeni sistemde kullanılabilir hale getirilmesi işlemidir.
- Yazılım projelerinde, eski bir sistemden (legacy system) yeni bir sisteme geçiş yapılırken veya bir yazılımın farklı bir altyapıya taşınması gerektiğinde gerçekleştirilir.
- Veri geçişi işlemi, dikkatli bir planlama ve titizlik gerektiren, yazılım yaşam döngüsünün kritik bir adımıdır.

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **Veri Geçişi Sürecinin Detayları**
- ✓ **1. Planlama ve Hazırlık**
- **Mevcut Verilerin Analizi:**
  - Hangi verilerin taşınacağı belirlenir.
  - Verilerin büyüklüğü, formatı, ilişkileri ve hassasiyeti (örneğin kişisel veriler) analiz edilir.
  - Veritabanı yapısındaki farklılıklar, veri türleri ve bağımlılıklar incelenir.
- **Veri Geçişi Stratejisinin Belirlenmesi:**
  Geçiş için uygun yöntem ve araçlar seçilir. Yaygın stratejiler:
  - **Big Bang Geçişi:** Tüm veriler bir seferde taşınır. Hızlıdır ancak risklidir.
  - **Aşamalı Geçiş:** Veriler parça parça, belirli zaman dilimlerinde taşınır. Daha az risk taşır ancak daha uzun sürer.
- **Geri Dönüş Planı (Rollback Plan):**
  Geçiş sırasında bir sorun yaşanırsa eski sisteme dönmek için bir plan hazırlanır.

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **2. Veri Haritalama ve Dönüşüm**
- **Veri Haritalama (Data Mapping):**
  Eski sistemdeki veri yapısı, yeni sistemin veri yapısına uyarlanır.
  Örneğin:
  - Eski sistemde "customer_id" olarak geçen bir alan, yeni sistemde "client_id" olabilir.
- **Veri Dönüşümü (Data Transformation):**
  - Verilerin yeni sisteme uygun hale getirilmesi için format veya yapı değişiklikleri yapılır.
  - Eksik, hatalı veya gereksiz veriler temizlenir (data cleansing).

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **3. Veri Taşıma (Data Migration)**
- Veriler, seçilen araçlar ve yöntemlerle taşınır. Bu süreç, genellikle şu yöntemlerle gerçekleştirilir:
  - **Elle Taşıma:** Küçük çaplı veri geçişlerinde manuel işlemler yapılır.
  - **Otomatik Araçlar:** Büyük ölçekli veri geçişlerinde, ETL (Extract, Transform, Load) araçları kullanılır. Örneğin:
    - **Extract:** Veriler eski sistemden alınır.
    - **Transform:** Veriler yeni sistemin gereksinimlerine uygun şekilde dönüştürülür.
    - **Load:** Veriler yeni sisteme yüklenir.

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **4. Veri Geçişi Testi**
- Geçiş sonrası, taşınan verilerin doğruluğu ve bütünlüğü test edilir:
  - **Veri Doğrulama (Validation):** Yeni sistemdeki verilerin eski sistemle uyumlu olup olmadığı kontrol edilir.
  - **Veri Tutarlılığı:** Veritabanı içindeki ilişkilerin ve bağlantıların doğru şekilde taşındığına emin olunur.

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **5. Geçiş Sonrası İşlemler**
- **Sistem Performans Testi:** Yeni sistemin veri yükünü sorunsuz taşıdığı doğrulanır.
- **Kullanıcı Eğitimleri:** Kullanıcılar yeni sisteme alışabilmeleri için eğitilir.
- **Eski Sistemin Devre Dışı Bırakılması:** Veri geçişi tamamen başarılı olunca eski sistem kapatılır veya yedekte tutulur.

Kurulum (Deployment)

- ✓ Veri Geçişi
- ✓ **Karşılaşılan Zorluklar**

1. **Veri Uyumsuzlukları:** Eski ve yeni sistem arasındaki farklılıklar nedeniyle bazı veriler kaybolabilir veya hatalı taşınabilir.
2. **Büyük Veri Miktarı:** Büyük veri setleri geçiş sürecini zorlaştırabilir ve gecikmelere neden olabilir.
3. **Gizlilik ve Güvenlik:** Hassas verilerin taşınması sırasında güvenlik önlemleri alınmazsa veri ihlalleri yaşanabilir.

Kurulum (Deployment)

- Test ve Onay Süreci (Testing and Approval)
- **Test ve Onay**, yazılımın çalışma ortamında belirlenen gereksinimlere uygun şekilde çalışıp çalışmadığını doğrulamak için yapılan değerlendirme sürecidir.
- Bu aşama, hem teknik hem de işlevsel olarak yazılımın hazır olduğunu kanıtlamak için kritik öneme sahiptir.
- Testlerin ardından, ilgili paydaşlar yazılımın onayını verir ve bir sonraki aşamaya geçilmesine karar verilir.

Kurulum (Deployment)

- ✓ Test ve Onay Süreci (Testing and Approval)
- ✓ **Test Süreci**
- ✓ **1. Test Planlama**
- **Test Stratejisi:**
  Yazılımın hangi yönlerinin test edileceği, hangi araçların kullanılacağı ve hangi ekiplerin dahil olacağı belirlenir.
- **Test Kapsamı:**
  Tüm gereksinimler gözden geçirilir ve test edilecek özellikler netleştirilir.
  - **Fonksiyonel Testler:** Yazılımın işlevlerinin doğru çalışıp çalışmadığını doğrular.
  - **Performans Testleri:** Yazılımın hız, ölçeklenebilirlik ve yük altında davranışı test edilir.
  - **Güvenlik Testleri:** Yetkisiz erişimlere ve güvenlik açıklarına karşı testler yapılır.
  - **Kullanılabilirlik Testleri:** Kullanıcı deneyimi açısından yazılım değerlendirilir.

Kurulum (Deployment)

- ✓ Test ve Onay Süreci (Testing and Approval)
- ✓ **Test Süreci**
- ✓ **2. Test Türleri**
- **Birim Testleri (Unit Testing):**
  Kodun en küçük birimlerinin (ör. bir fonksiyonun) doğru çalıştığını kontrol eder.
  - Genelde otomasyon araçlarıyla yapılır.
- **Entegrasyon Testleri (Integration Testing):**
  Birbirine bağlı modüllerin birlikte doğru çalışıp çalışmadığını kontrol eder.
- **Sistem Testleri (System Testing):**
  Yazılımın uçtan uca, tüm modülleriyle birlikte çalışabilirliğini test eder.
- **Kabul Testleri (User Acceptance Testing - UAT):**
  Yazılımın kullanıcı gereksinimlerini karşılayıp karşılamadığını doğrulamak için son kullanıcılar tarafından yapılan testlerdir.

Kurulum (Deployment)

- ✓ Test ve Onay Süreci (Testing and Approval)
- ✓ **Onay Süreci**
- ✓ **1. Test Sonuçlarının İncelenmesi**
- Tüm test sonuçları detaylı şekilde raporlanır ve analiz edilir.
- Belirlenen hatalar veya eksiklikler, **kritik (blocker)** veya **düşük öncelikli (low priority)** olarak kategorize edilir.
  - **Kritik sorunlar:** Çözülmeden yazılım onaylanamaz.
  - **Düşük öncelikli sorunlar:** Gelecekte düzeltilebilir ancak şimdilik dağıtıma engel değildir.

Kurulum (Deployment)

- ✓ Test ve Onay Süreci (Testing and Approval)
- ✓ **2. Onay Sürecinde Dikkate Alınan Kriterler**
- **Gereksinimlerin Karşılanması:** Yazılımın tüm belirlenen işlevsel ve teknik gereksinimleri karşılaması beklenir.
- **Kullanıcı Onayı:** Son kullanıcıların yazılımı kabul etmesi, işlevselliğini ve kullanılabilirliğini onaylaması gerekir.
- **Performans Standartları:** Yazılımın performans kriterlerine (ör. hız, ölçeklenebilirlik) uygun olup olmadığı değerlendirilir.

Kurulum (Deployment)

- ✓ Test ve Onay Süreci (Testing and Approval)
- ✓ **3. Geri Bildirim ve Düzeltmeler**
- Onay sürecinde paydaşlardan alınan geri bildirimler değerlendirilir.
- Geri bildirimlere dayalı olarak yazılımda gerekli düzenlemeler yapılır.
- ✓ **4. Nihai Onay**
- Tüm testler başarıyla tamamlandığında ve geri bildirimler ele alındığında, yazılım resmi olarak **onaylanır** ve dağıtım sürecine hazır hale gelir.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Eğitim ve Dokümantasyon**, yazılımın kullanıcılar ve teknik ekipler tarafından verimli bir şekilde kullanılması, yönetilmesi ve desteklenmesi için gerekli bilgilerin aktarılması ve rehberlerin hazırlanması sürecidir.

- ✓ Bu adım, yazılımın etkin bir şekilde benimsenmesi ve uzun vadeli başarısı için kritik öneme sahiptir.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Eğitim Süreci**
- ✓ **1. Eğitim Planlaması**
- **Hedef Kitlenin Belirlenmesi:**
  Eğitim verilecek gruplar belirlenir.
  - **Son Kullanıcılar:** Yazılımı günlük işlerinde kullanan kişiler.
  - **Teknik Destek Ekibi:** Yazılımın bakımını ve sorun giderme işlemlerini yapan ekipler.
  - **Yöneticiler:** Yazılımın iş süreçleri üzerindeki etkisini değerlendiren ve raporlama yapan kişiler.
- **Eğitim İçeriğinin Hazırlanması:**
  Eğitim içeriği, hedef kitlenin ihtiyaçlarına göre uyarlanır.
  - **Son kullanıcılar için:** Yazılımın temel özellikleri, kullanım senaryoları ve sık yapılan işlemler.
  - **Teknik ekip için:** Yazılımın altyapısı, kurulum, bakım ve hata giderme prosedürleri.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Eğitim Süreci**
- ✓ **2. Eğitim Yöntemleri**
- **Yerinde Eğitim (On-Site Training):**
  Eğitimler doğrudan kullanıcıların çalışma ortamında yüz yüze gerçekleştirilir.
- **Uzaktan Eğitim (Online Training):**
  Çevrimiçi platformlar veya videolar aracılığıyla eğitim verilir. Bu yöntem, coğrafi olarak dağınık ekipler için idealdir.
- **Hibrit Eğitim:**
  Yerinde ve çevrimiçi eğitimlerin bir kombinasyonu kullanılır.
- **Pratik Çalışmalar:**
  Kullanıcıların yazılımı deneyimlemeleri için simülasyonlar veya test ortamları sağlanır.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Eğitim Süreci**
- ✓ **3. Eğitim Materyalleri**
- **Sunumlar ve Videolar:** Yazılımın özelliklerini açıklayan görsel materyaller.
- **Kılavuzlar ve Broşürler:** Yazılımın kullanımını adım adım açıklayan belgeler.
- **Sık Sorulan Sorular (FAQ):** Yaygın sorunlar ve çözümler hakkında bilgi sağlar.
- ✓ **4. Eğitim Değerlendirmesi**
- Eğitim sonrasında kullanıcıların bilgi seviyeleri ölçülür.
- Değerlendirme anketleri veya testler ile eksik kalan konular tespit edilir ve gerekli destek sağlanır.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Dokümantasyon Süreci**
- ✓ **1. Dokümantasyonun Amacı**
- Yazılımın doğru kullanımı ve sürdürülebilirliği için rehberlik sunmak.
- Teknik ve işlevsel süreçlerin açıkça anlaşılmasını sağlamak.
- Sorun giderme ve bakım işlemleri sırasında başvurulacak bir kaynak sağlamak.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Dokümantasyon Süreci**
- ✓ **2. Dokümantasyon Türleri**
- **Kullanıcı Kılavuzu (User Manual):**
  Yazılımın kullanıcı dostu bir şekilde nasıl kullanılacağını anlatır.
  - Örnek: "Yeni bir sipariş nasıl oluşturulur?" veya "Raporlar nasıl görüntülenir?"
- **Teknik Dokümantasyon (Technical Documentation):**
  Yazılımın altyapısı, mimarisi ve teknik detayları hakkında bilgi verir.
  - Örnek: Sistem gereksinimleri, API entegrasyonları, veritabanı şemaları.
- **Yönetim Dokümanları:**
  Yazılımın izlenmesi, performans yönetimi ve güvenlik süreçlerini kapsar.
- **Eğitim Materyalleri:**
  Kullanıcı eğitimleri için hazırlanmış özel dökümanlar ve sunumlar.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Dokümantasyon Süreci**
- ✓ **3. Dokümantasyon Araçları ve Formatları**
- **Araçlar:**
  - Microsoft Word, Google Docs (basit kılavuzlar için).
  - Confluence, Notion (işbirlikçi belgeler için).
  - GitHub Wiki, ReadTheDocs (teknik dokümantasyon için).
- **Formatlar:**
  - PDF veya basılı belgeler.
  - Web tabanlı rehberler.
  - Videolu anlatımlar ve infografikler.

Kurulum (Deployment)

- ✓ Eğitim ve Dökümantasyon
- ✓ **Dokümantasyon Süreci**
- ✓ **4. Dokümantasyonun Güncellenmesi**
- Yazılımda yapılan değişiklikler veya güncellemeler dokümantasyona yansıtılmalıdır.
- Güncel olmayan dokümantasyon, kullanıcıların yanlış bilgilerle hareket etmesine neden olabilir.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş**, yazılımın geliştirme ve test ortamlarından çıkarılarak, gerçek kullanıcıların erişimine açıldığı ve üretim ortamında kullanılmaya başlandığı süreçtir.
- ✓ Bu adım, yazılım geliştirme yaşam döngüsünün en kritik aşamalarından biridir.
- ✓ Başarılı bir canlıya geçiş süreci, titiz bir planlama, test ve koordinasyon gerektirir.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Sürecinin Adımları**
- ✓ **1. Canlıya Geçiş Planlaması**
- **Tarih ve Zamanlama:**
  - Canlıya geçiş için uygun bir tarih ve saat belirlenir. Genellikle iş yükünün az olduğu zamanlar (örneğin, hafta sonu veya gece saatleri) tercih edilir.
  - Kullanıcı etkisini en aza indirmek için geçiş sırasında gerekli duyurular yapılır.
- **Rol ve Sorumlulukların Belirlenmesi:**
  - Her ekip üyesinin görevleri ve sorumlulukları net şekilde tanımlanır.
  - Geliştiriciler, test ekipleri, sistem yöneticileri ve proje yöneticileri arasında koordinasyon sağlanır.
- **Risk Yönetimi ve Acil Durum Planları:**
  - Olası riskler ve bu risklere karşı alınacak önlemler belirlenir.
  - Geçiş sırasında beklenmeyen sorunlar ortaya çıkarsa geri dönüş (rollback) planı hazır olmalıdır.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Sürecinin Adımları**
- ✓ **2. Ön Hazırlık**
- **Üretim Ortamının Hazırlanması:**
  - Sunucular, ağlar ve veritabanları gibi altyapı bileşenlerinin canlıya geçişe uygun olduğundan emin olunur.
  - Güvenlik ayarları ve erişim kontrolleri yeniden gözden geçirilir.
- **Son Testlerin Yapılması:**
  - **Duman Testi (Smoke Test):** Sistem bileşenlerinin temel işlevlerinin çalıştığını doğrulamak için yapılır.
  - **Kabul Testi (Final Acceptance Test):** Yazılımın iş gereksinimlerini karşıladığını bir kez daha doğrulamak için son kullanıcılar tarafından yapılır.
- **Yedekleme:**
  - Veri kaybını önlemek için mevcut sistem ve verilerin tam bir yedeği alınır.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Sürecinin Adımları**
- ✓ **3. Geçiş İşlemi**
- **Veri Aktarımı:**
  Eğer eski bir sistemden yeni sisteme geçiliyorsa, veri aktarımı yapılır ve doğruluğu kontrol edilir.
- **Sistem Bileşenlerinin Yayına Alınması:**
  - Yazılım kodu, yapılandırma dosyaları ve gerekli tüm bileşenler üretim ortamına taşınır.
  - Hizmetlerin başlatılması ve gerekli süreçlerin aktif hale getirilmesi sağlanır.
- **Performans Kontrolü:**
  Sistem üzerinde ilk yük testi yapılarak performans sorunlarının olup olmadığı kontrol edilir.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Sürecinin Adımları**
- ✓ **4. Canlıya Geçiş Sonrası İzleme ve Destek**
- **Aktif İzleme:**
  - Yazılımın performansı, hata oranları ve sistem kararlılığı izlenir.
  - İzleme araçları (örneğin, New Relic, Splunk) ile sistemin canlıya geçiş sonrası davranışı analiz edilir.
- **Hata ve Sorun Giderme:**
  - Canlıya geçiş sırasında veya hemen sonrasında tespit edilen hatalar hızla çözülür.
  - Öncelik sırasına göre kritik hatalar hemen ele alınır.
- **Kullanıcı Geri Bildirimi:**
  - İlk kullanıcıların geri bildirimleri toplanır ve yazılım üzerinde gerekli düzeltmeler yapılır.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Modelleri**
- ✓ **1. Büyük Patlama (Big Bang Deployment):**
  - Tüm sistem bir kerede canlıya alınır.
  - Avantajı: Hızlıdır ve tek bir geçiş sürecini gerektirir.
  - Dezavantajı: Risklidir; büyük bir hata sistemin tamamını etkileyebilir.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Modelleri**
- ✓ **2. Aşamalı Geçiş (Phased Deployment):**
- Yazılımın belirli bölümleri veya kullanıcı grupları üzerinde aşamalı olarak etkinleştirilir.
- Avantajı: Hataların erken tespit edilmesi ve kullanıcı etkisinin sınırlı olmasıdır.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Modelleri**
- ✓ **3. Paralel Çalışma (Parallel Deployment):**
- Yeni sistem, eski sistemle birlikte çalıştırılır ve kullanıcıların yeni sisteme alışması sağlanır.
- Avantajı: Daha güvenlidir, çünkü eski sistem bir yedek olarak kullanılabilir.
- Dezavantajı: Ekstra maliyet ve karmaşıklık getirir.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Canlıya Geçiş Modelleri**
- ✓ **4. Mavi-Yeşil Dağıtım (Blue-Green Deployment):**
- Yeni sistem, "yeşil" ortamda aktif hale getirilir ve kullanıcılar eski sistem ("mavi" ortam) üzerinden yeni sisteme yönlendirilir.
- Avantajı: Sorunsuz geri dönüş yapılabilmesini sağlar.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Başarı Kriterleri**
- Yazılımın tüm temel işlevleri sorunsuz bir şekilde çalışıyor olmalıdır.
- Kullanıcılar tarafından rapor edilen kritik sorunların olmaması.
- Sistem performansının beklentilerle uyumlu olması.

Kurulum (Deployment)

- ✓ Canlıya Geçiş (Go-Live)
- ✓ **Karşılaşılabilecek Zorluklar**

1. **Beklenmeyen Teknik Hatalar:** Yazılımın üretim ortamında öngörülmeyen hatalar vermesi.
2. **Veri Kaybı:** Eksik veya hatalı veri aktarımı nedeniyle ortaya çıkan sorunlar.
3. **Performans Sorunları:** Sistem yavaşlığı veya kullanıcı yoğunluğuna dayanamama.
4. **Kullanıcı Direnci:** Yeni sisteme uyum sağlamakta zorlanan kullanıcılar.

Kurulum (Deployment)

- İzleme ve İlk Destek
- **İzleme ve İlk Destek**, yazılım canlıya geçiş yaptıktan sonra sistemin kararlılığını ve performansını takip etmek ve ortaya çıkabilecek sorunlara hızlıca müdahale etmek için yapılan işlemleri kapsar.

- Bu aşama, yazılımın başarılı bir şekilde kullanıcılar tarafından benimsenmesi ve kullanım sırasında aksaklıkların en aza indirilmesi açısından kritik bir adımdır.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **1.İzleme Süreci (Monitoring)**
- ✓ **Amaç:**
- ✓ Sistemin performansını, güvenilirliğini ve kullanıcı deneyimini gerçek zamanlı olarak takip etmek ve sorunları erken tespit ederek çözmek.
- ✓ **İzlenecek Unsurlar:**

a. Sistem Performansı:

1. CPU, bellek, ağ trafiği gibi kaynak kullanımları izlenir.
2. Yanıt süreleri, işlem hızları ve gecikmeler analiz edilir.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **1. İzleme Süreci (Monitoring)**

b. Hata Tespiti:

1. Sistem hataları, çökme durumları veya log dosyalarında oluşan uyarılar sürekli kontrol edilir.
2. Kritik hatalar için otomatik bildirim sistemleri kurulur.

c. Kullanıcı Davranışları:

1. Kullanıcıların yazılım üzerindeki etkileşimleri analiz edilerek beklenmeyen kullanım senaryoları tespit edilir.
2. Kullanıcıların sıklıkla zorlandığı alanlar belirlenir.

d. Güvenlik İzleme:

1. Yetkisiz erişim denemeleri, güvenlik açıkları ve anormal etkinlikler takip edilir.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **1. İzleme Süreci (Monitoring)**

- ✓ **İzleme Araçları:**
- **Uygulama İzleme:** New Relic, AppDynamics, Datadog.
- **Sistem İzleme:** Nagios, Zabbix, SolarWinds.
- **Log Analizi:** ELK Stack (Elasticsearch, Logstash, Kibana), Splunk.

- ✓ **İzleme Sürecinde Yapılan İşlemler:**
- **Gerçek Zamanlı Alarm Sistemleri:** Sistem performansında veya güvenlikte sorun olduğunda otomatik uyarılar gönderilir.
- **Trend Analizleri:** Uzun vadeli performans ve kullanım eğilimleri analiz edilerek gelecekteki ihtiyaçlar öngörülür.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **2. İlk Destek Süreci (Initial Support)**
- ✓ **Amaç:**
- ✓ Kullanıcıların canlıya geçiş sonrası yazılımla ilgili karşılaştığı sorunları çözmek, sorunsuz bir deneyim sunmak ve yazılımın benimsenmesini sağlamak.
- ✓ **İlk Destek Sürecinin Aşamaları:**

1. **Destek Ekibinin Hazırlığı:**
    1. Yazılım hakkında detaylı eğitim almış bir destek ekibi oluşturulur.
    2. Ekibe, sık karşılaşılabilecek sorunlar ve çözüm yöntemleri hakkında bilgi verilir.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **2. İlk Destek Süreci (Initial Support)**

1. **Sorun Bildirimi:**
    1. Kullanıcılar için kolay erişilebilir bir destek kanalı sağlanır.
        1. Örnek: Telefon hattı, e-posta, canlı sohbet, yardım masası (ticketing) sistemi.
2. **Sorunların Çözümü:**
    1. **Hızlı Müdahale:**
       Küçük ve çözümü kolay sorunlar için hızlı destek sağlanır.
    2. **Eskalasyon:**
       Karmaşık sorunlar, ikinci seviye destek ekibine veya yazılım geliştirme ekibine yönlendirilir.
3. **Kullanıcı Geri Bildirimi:**
    1. Kullanıcılardan alınan geri bildirimler toplanır ve yazılımın geliştirilmesi için değerlendirilir.
    2. Öneriler dokümante edilerek ileride yapılacak güncellemelerde dikkate alınır.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **2. İlk Destek Süreci (Initial Support)**
- ✓ **İlk Destek Sürecinde Yaygın Sorunlar:**
  - Kullanıcı giriş problemleri.
  - Sistem performansındaki yavaşlıklar.
  - Beklenmeyen hata mesajları.
  - Yeni özelliklerin kullanımında zorluklar.
- ✓ **Destek Araçları:**
  - **Yardım Masası Uygulamaları:** Jira Service Management, Zendesk, Freshdesk.
  - **Kullanıcı Eğitim Materyalleri:** Sık Sorulan Sorular (FAQ) sayfaları, rehber videolar.

Kurulum (Deployment)

- ✓ İzleme ve İlk Destek
- ✓ **3. İzleme ve İlk Destek Sürecinin Faydaları**
- **Hata Tespiti ve Çözümü:**
  Sorunlar erken tespit edilerek kullanıcıları olumsuz etkilemeden çözülebilir.
- **Kullanıcı Memnuniyeti:**
  Hızlı ve etkili destek, kullanıcıların yazılıma olan güvenini artırır.
- **Verimlilik:**
  Yazılımın performansındaki aksaklıklar hızla düzeltilerek işletme süreçleri kesintisiz devam eder.
- **Sürekli İyileştirme:**
  İzleme ve destek süreçlerinden elde edilen geri bildirimler, yazılımın uzun vadeli geliştirilmesine katkı sağlar.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Bakım (Maintenance)

- ✓ **Bakımın Amaçları**

1. **Hataların Düzeltilmesi (Corrective Maintenance):**
   1. Yazılımın kullanım sırasında tespit edilen hatalarını düzeltmek.
   2. Kullanıcı geri bildirimlerini değerlendirerek yazılımda iyileştirmeler yapmak.
   3. Beklenmeyen durumlar (bug'lar, performans problemleri) için hızlı çözümler geliştirmek.
2. **Yazılımın İyileştirilmesi (Perfective Maintenance):**
   1. Yazılımı daha verimli ve kullanıcı dostu hale getirmek için yeni özellikler eklemek.
   2. Kullanıcı gereksinimlerindeki değişikliklere cevap vermek.

Bakım (Maintenance)

- ✓ **Bakımın Amaçları**

3. Uyarlanabilirlik (Adaptive Maintenance):

1. Yazılımı yeni işletim sistemlerine, donanımlara veya teknolojik platformlara uyarlamak.
2. Mevcut yasal ve düzenleyici değişikliklere uyum sağlamak.

4. Önleyici Bakım (Preventive Maintenance):

1. Yazılımın gelecekteki olası sorunlarını önlemek için sistemdeki potansiyel zayıflıkları gidermek.
2. Yazılımın güvenlik açıklarını kapatmak.

Bakım (Maintenance)

- ✓ **Bakım Süreci**

1. **Sorun Tespiti:**
    1. Kullanıcılardan gelen şikayetler ve geri bildirimler incelenir.
    2. Sistem monitörleme araçlarıyla performans ve hata kayıtları analiz edilir.
2. **Değişiklik Planlama:**
    1. Hangi değişikliklerin yapılacağı ve bu değişikliklerin etkileri değerlendirilir.
    2. Gereksinim analizleri yapılır ve iş yükü tahmini yapılır.
3. **Kodun Güncellenmesi:**
    1. İlgili kod parçaları düzenlenir veya yeniden yazılır.
    2. Yeni özellikler eklenir veya mevcut özellikler optimize edilir.

Bakım (Maintenance)

- ✓ **Bakım Süreci**

4. Test Süreci:

1. Yapılan değişikliklerin sistemin diğer parçalarını olumsuz etkilemediğinden emin olmak için test edilir.
2. Yazılımın hem yeni hem de eski işlevlerinin sorunsuz çalıştığından emin olunur (regresyon testi).

5. Güncellemelerin Dağıtımı:

1. Değişiklikler kullanıcıya sunulur (örneğin, bir güncelleme paketi veya sürüm yükseltmesiyle).
2. Kullanıcılara güncellemelerle ilgili dokümantasyon ve eğitim sağlanabilir.

6. İzleme ve Değerlendirme:

1. Yeni değişikliklerin sistem üzerindeki etkisi izlenir.
2. Kullanıcıların memnuniyet düzeyi değerlendirilir.

Bakım (Maintenance)

- ✓ **Karşılaşılan Zorluklar**
- Eski yazılım teknolojileriyle çalışmak.
- Yetersiz dokümantasyon nedeniyle kodu anlamanın zor olması.
- Sürekli değişen kullanıcı gereksinimleri.
- Bakım sürecinde ortaya çıkan yeni hatalar.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **1. Sorun Tespiti**
- **Kullanıcı geri bildirimleri:** Kullanıcılardan bir kısmı, arama motorunda "kırmızı ayakkabı" yazdıklarında alakasız sonuçlar çıktığını bildirmiş.
- **Performans sorunları:** Kullanıcı sayısındaki artış nedeniyle bazı saatlerde site yavaşlıyor.
- **Hukuki gereklilikler:** Yeni bir veri gizliliği kanunu (örneğin, GDPR) yürürlüğe girmiş ve sitenin buna uyum sağlaması gerekiyor.
- **Yeni talepler:** Kullanıcılar, ürün değerlendirme ve yorumlarına resim eklemek istiyor.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **2. Değişiklik Planlama**
- **Hataların Düzeltilmesi (Corrective Maintenance):**
  - Arama motoru algoritmasındaki hataların tespit edilmesi ve düzeltilmesi planlanır.
- **Uyarlanabilirlik (Adaptive Maintenance):**
  - GDPR uyumluluğu için veri saklama politikalarının güncellenmesi, çerez yönetim sisteminin eklenmesi.
- **İyileştirme (Perfective Maintenance):**
  - Performans için altyapının ölçeklenebilir hale getirilmesi (örneğin, sunucu yük dengeleme).
- **Önleyici Bakım (Preventive Maintenance):**
  - Veri tabanı yedekleme süreçlerinin optimize edilmesi.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **3. Kodun Güncellenmesi**
- **Arama Motoru:**
  - Arama algoritması, ürünlerin açıklamaları ve anahtar kelimelerle eşleşmesini iyileştirecek şekilde yeniden yazılır.
- **Performans Optimizasyonu:**
  - Sunucu tarafında yük dengeleme yapılır, eski sorgular yeniden düzenlenir, gereksiz veriler temizlenir.
- **GDPR Uyumluluğu:**
  - Kullanıcılara, veri saklama izinlerini yönetebilecekleri bir kontrol paneli eklenir.
- **Yeni Özellikler:**
  - Kullanıcıların yorumlarına görsel ekleyebilmesi için bir "resim yükleme" modülü eklenir.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **4. Test Süreci**
- **Birim Testleri:** Arama motoru algoritmasında yapılan değişikliklerin doğru sonuçlar ürettiği kontrol edilir.
- **Performans Testi:** Artan yük altında sitenin nasıl çalıştığı simüle edilir.
- **Uyumluluk Testi:** GDPR uyumluluğu için veri akışları, çerez yönetimi ve kullanıcı onay süreçleri test edilir.
- **Yeni Özellik Testi:** Resim yükleme özelliği, farklı boyut ve formatlarda test edilerek kullanıcı dostu hale getirilir.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **5. Güncellemelerin Dağıtımı**
- Kod, önce test sunucusuna dağıtılır.
- Testler başarıyla tamamlandıktan sonra canlı ortama alınır.
- Kullanıcılara yeni özellikler hakkında bilgilendirme yapılır (örneğin, e-posta veya sitedeki bir duyuru ile).

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakım Adımının Detayları**
- ✓ **6. İzleme ve Değerlendirme**
- **Sorun tespiti:** Yeni arama algoritmasının performansı takip edilir. Kullanıcıların yanlış sonuç bildirme oranı izlenir.
- **Kullanıcı geri bildirimleri:** Resim yükleme özelliği için kullanıcıların yorumları alınır.
- **Performans izleme:** Site hızında ve sunucu yükündeki değişiklikler değerlendirilir.

Bakım (Maintenance)- Örnek Uygulama

- ✓ **Bakımın Getirileri**

1. **Kullanıcı memnuniyeti:** Arama sonuçlarının doğruluğu ve yeni özellikler memnuniyeti artırır.
2. **Yasal uyumluluk:** GDPR uyumuyla hukuki riskler ortadan kaldırılır.
3. **Performans artışı:** Site hızı ve ölçeklenebilirliği sayesinde daha fazla kullanıcıya hizmet verilebilir.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Sonlandırma (Retirement)

- ✓ Yazılım yaşam döngüsünün "Sonlandırma (Retirement)" adımı, bir yazılım sisteminin artık kullanılmaz hale geldiği, kullanım dışı bırakıldığı veya tamamen devre dışı bırakıldığı süreci ifade eder.
- ✓ Bu adım, yazılımın işlevsel ömrünün sona erdiğini ve aktif destek veya bakım gerektirmediğini gösterir.
- ✓ Bu süreç, dikkatli bir planlama ve uygulama gerektirir, çünkü kullanıcılar, veriler ve bağlı sistemler üzerindeki etkisi büyük olabilir.

Sonlandırma (Retirement)

- ✓ **Sonlandırma Adımının Detayları**
- ✓ **1. Nedenleri**
- ✓ Yazılımın sonlandırılma kararı aşağıdaki nedenlere dayanabilir:
- **Teknolojik Eskime:** Yazılım, artık çağın gereksinimlerini karşılamıyorsa (örneğin, eski bir programlama dilinde yazılmışsa veya modern işletim sistemlerinde çalışmıyorsa).
- **Alternatiflerin Geliştirilmesi:** Daha verimli, güvenli veya işlevsel bir yazılım geliştirilmişse.
- **Maliyet:** Yazılımın bakım ve destek maliyetlerinin, sağladığı faydalardan daha yüksek hale gelmesi.
- **Kullanım Azalması:** Yazılımın kullanıcı kitlesinin büyük ölçüde azalması.
- **Yasal veya Düzenleyici Gereklilikler:** Yazılımın, yeni düzenlemelere uyum sağlayamaması.

Sonlandırma (Retirement)

- ✓ **2. Sürecin Aşamaları**
- ✓ **a. Etki Analizi**
- Yazılımın sonlandırılmasının kullanıcılar, veriler ve ilişkili sistemler üzerindeki etkileri değerlendirilir.
- Bağımlı sistemler, entegrasyonlar ve kullanıcı verileri incelenir.
- ✓ **b. Kullanıcı ve Paydaş Bildirimi**
- Yazılımın sonlandırılacağına dair tüm kullanıcılar, müşteriler ve ilgili paydaşlar bilgilendirilir.
- Bildirim genellikle sonlandırma tarihinden belirli bir süre önce yapılır, böylece kullanıcılar yeni bir yazılıma geçiş için hazırlık yapabilir.

Sonlandırma (Retirement)

- ✓ **2. Sürecin Aşamaları**
- ✓ **c. Verilerin Taşınması veya Yedeklenmesi**
- Yazılımın kullandığı veriler, kullanıcıların erişebileceği bir şekilde taşınır veya yedeklenir.
- Bazı durumlarda, veriler yeni bir sisteme aktarılır veya düzenleyici gereklilikler doğrultusunda arşivlenir.
- ✓ **d. Sistemden Ayırma**
- Yazılım, mevcut altyapıdan ve bağlı sistemlerden güvenli bir şekilde ayrılır.
- Sunucular, veri tabanları ve diğer bağlı bileşenler temizlenir veya devre dışı bırakılır.

Sonlandırma (Retirement)

- ✓ **2. Sürecin Aşamaları**
- ✓ **e. Sonlandırma Testleri**
- Sonlandırma sırasında sistemin diğer bileşenlere zarar vermediğinden emin olmak için test yapılır.
- Veri kaybı, güvenlik açıkları veya erişim sorunlarının oluşmaması sağlanır.
- ✓ **f. Kapatma**
- Yazılım tamamen kapatılır.
- Kullanıcıların yazılıma erişimi sonlandırılır ve gerekli tüm sistem kaynakları serbest bırakılır.
- ✓ **g. Sonlandırma Belgelerinin Hazırlanması**
- Yazılımın sonlandırılmasıyla ilgili tüm süreçler dokümante edilir.
- Gelecekte referans olması için nedenler, süreç ve sonuçlar kaydedilir.

Sonlandırma (Retirement)

- ✓ **Sonlandırmanın Önemi**
- **Kaynakların Serbest Bırakılması:** Yazılımın bakım ve destek yükünü ortadan kaldırarak kaynakların diğer projelere aktarılmasını sağlar.
- **Kullanıcı Güveni:** Kullanıcıların doğru bir şekilde yönlendirilmesi ve desteklenmesi, marka güvenini artırır.
- **Gelecekteki Projeler İçin Dersler:** Sonlandırma sürecinden çıkarılan dersler, yeni projelerde daha etkili bir planlama yapılmasını sağlar.

Sonlandırma (Retirement)

- ✓ **Sonlandırma Sürecinde Dikkat Edilmesi Gerekenler**

1. **Kullanıcı Verisi:** Kullanıcı verilerinin güvenliği ve gizliliği sağlanmalıdır.
2. **Alternatif Çözümler:** Yazılımı kullanan kullanıcılar için alternatif sistemlerin önerilmesi.
3. **Regülasyonlar:** Özellikle veri gizliliği ve güvenliğiyle ilgili yasal düzenlemelere uygun hareket edilmesi.
4. **Geri Bildirim:** Kullanıcılardan ve paydaşlardan alınan geri bildirimlerin değerlendirilmesi.

Sonlandırma (Retirement)- Örnek Uygulama

- ✓ **Sonlandırma Sürecinin Aşamaları**
- ✓ **1. Etki Analizi**
- **Kullanıcı Analizi:**
  - Mevcut CRM yazılımında çalışan 50 kullanıcı var.
  - Veriler aktif olarak satış, pazarlama ve müşteri destek ekipleri tarafından kullanılıyor.
- **Bağımlılıklar:**
  - CRM yazılımı, şirketin e-posta sistemine, müşteri destek yazılımına ve finans sistemine entegre.
- **Riskler:**
  - Veri kaybı veya entegrasyon sorunları, müşteri memnuniyetini olumsuz etkileyebilir.

Sonlandırma (Retirement)- Örnek Uygulama

- ✓ **Sonlandırma Sürecinin Aşamaları**
- ✓ **2. Kullanıcı ve Paydaş Bildirimi**
- Şirket içindeki tüm ekipler, mevcut sistemin kullanım dışı bırakılacağı ve yeni sisteme geçileceği hakkında bilgilendirildi.
- Bildirim, bir geçiş planıyla birlikte yapıldı. Sonlandırma tarihi ve yeni sistem eğitim programı açıklandı.
- Kullanıcılara, CRM'den alabilecekleri raporlar ve geçmiş veriler hakkında bilgi verildi.

Sonlandırma (Retirement)- Örnek Uygulama

- ✓ **Sonlandırma Sürecinin Aşamaları**
- ✓ **3. Verilerin Taşınması veya Yedeklenmesi**
- **Veri Taşıma:**
  - Mevcut müşteri bilgileri, satış raporları ve destek kayıtları yeni CRM sistemine aktarıldı.
  - Taşıma işlemi sırasında veri formatları dönüştürüldü.
- **Yedekleme:**
  - Tüm veriler, güvenli bir şekilde arşivlendi ve yalnızca yetkili kişilerin erişebileceği bir yerde saklandı.
  - Verilerin 5 yıl boyunca yasal düzenlemelere uygun şekilde saklanacağı belirlendi.

Sonlandırma (Retirement)- Örnek Uygulama

- ✓ **Sonlandırma Sürecinin Aşamaları**
- ✓ **4. Sistemden Ayırma**
- Eski CRM sistemi, şirketin e-posta ve finansal sistemlerinden ayrıldı.
- Tüm API bağlantıları kapatıldı ve kullanıcıların erişimi sonlandırıldı.
- ✓ **5. Sonlandırma Testleri**
- Veri taşıma işleminin doğruluğunu test etmek için, eski ve yeni CRM sistemlerindeki veriler karşılaştırıldı.
- Eski sistemin kapatılmasının diğer sistemlere etkisi test edildi (örneğin, e-posta entegrasyonunun kesintisiz çalışmaya devam ettiği kontrol edildi).

Sonlandırma (Retirement)- Örnek Uygulama

- ✓ **Sonlandırma Sürecinin Aşamaları**
- ✓ **6. Kapatma**
- Eski CRM sistemi, tamamen devre dışı bırakıldı.
- Sunucular ve altyapı, başka projeler için yeniden kullanılmak üzere serbest bırakıldı.
- Yazılım lisansı yenilenmedi ve ilgili destek anlaşmaları sonlandırıldı.
- ✓ **7. Sonlandırma Belgelerinin Hazırlanması**
- Sonlandırma sürecinin tüm detayları raporlandı.
- Kullanıcı geri bildirimleri ve süreç boyunca yaşanan sorunlar belgelendi.
- Veri yedekleme politikaları ve sistem kapatma tarihi kaydedildi.

Sonlandırma (Retirement)- Örnek Uygulama

✓ **Sonlandırma Sürecinde Dikkat Edilen Noktalar**

1. **Kullanıcı İhtiyaçları:** Kullanıcıların eski sisteme ihtiyaç duymaması için yeni sistemin eksiksiz çalıştığından emin olundu.
2. **Veri Güvenliği:** Tüm müşteri verilerinin yeni sisteme başarılı bir şekilde aktarıldığı doğrulandı.
3. **Eğitim ve Destek:** Kullanıcılara, yeni sistemi kullanmaya başlamadan önce eğitim verildi ve destek sağlandı.
4. **Hukuki Uyumluluk:** Veri yedekleme ve saklama süreçlerinin yasal gerekliliklere uygun olmasına dikkat edildi.

Sonlandırma (Retirement)- Örnek Uygulama

✓ **Sonlandırmanın Sonuçları**

- Eski sistemin kapatılmasıyla birlikte, bakım ve lisans maliyetlerinden tasarruf edildi.
- Yeni CRM yazılımı sayesinde kullanıcı deneyimi iyileştirildi ve iş süreçleri hızlandı.
- Sonlandırma süreci, şirketin diğer sistemlerini etkilemeden başarılı bir şekilde tamamlandı.

Yazılım Yaşam Döngüsü Adımları

- ✓ 1. **Gereksinim Analizi (Requirement Analysis)**
- ✓ 2. **Fizibilite Çalışması (Feasibility Study)**
- ✓ 3. **Sistem Tasarımı (System Design)**
- ✓ 4. **Kodlama (Coding)**
- ✓ 5. **Test Etme (Testing)**
- ✓ 6. **Kurulum (Deployment)**
- ✓ 7. **Bakım (Maintenance)**
- ✓ 8. **Sonlandırma (Retirement)**

Yazılım Kalite Yönetimi

- Yazılım kalite yönetimi, bir yazılım ürününün istenen gereksinimleri, müşteri beklentilerini ve endüstri standartlarını karşılamasını sağlamak için planlama, uygulama, kontrol ve sürekli iyileştirme süreçlerini kapsayan bir disiplindir.

- Bu süreç, yazılım geliştirme döngüsünün her aşamasında kaliteyi sağlamayı ve yazılımın güvenilir, kullanılabilir, verimli ve sürdürülebilir olmasını amaçlar.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetimi Bileşenleri**
- ✓ Yazılım kalite yönetimi, üç ana bileşen üzerine inşa edilmiştir:
- ✓ **1. Kalite Planlama (Quality Planning)**
- Yazılım geliştirme sürecinde kalite hedeflerini belirleme ve bu hedeflere ulaşmak için gerekli süreçleri planlama aşamasıdır.
- **Temel Aktiviteler:**
  - Kalite hedeflerini tanımlama.
  - Gereksinimlere uygun kalite standartlarını seçme (örneğin, ISO 9001, CMMI, Six Sigma).
  - Kalite güvence ve kontrol süreçlerini tasarlama.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetimi Bileşenleri**
- ✓ **2. Kalite Güvence (Quality Assurance - QA)**
- Yazılımın istenen kalite standartlarına uygun olarak geliştirilmesini sağlamak için önleyici süreçler içerir.
- **Temel Aktiviteler:**
  - Yazılım geliştirme süreçlerini denetleme.
  - Belirlenen süreçlerin uygun şekilde uygulanmasını sağlama.
  - Süreçleri sürekli izleyerek geliştirme.
- **Örnek Teknikler:**
  - Denetimler (audit).
  - Yazılım metrikleri kullanarak süreç analizi.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetimi Bileşenleri**
- ✓ **3. Kalite Kontrolü (Quality Control - QC)**
- Yazılımın son ürününün belirlenen standartlara uygunluğunu değerlendirmek için yapılan testler ve incelemeleri kapsar.
- **Temel Aktiviteler:**
  - Ürün testi ve hata tespiti.
  - Test sonuçlarına göre düzeltici eylemlerin uygulanması.
  - Çıkış kalitesinin onaylanması.
- **Örnek Teknikler:**
  - Birim testi, entegrasyon testi, sistem testi, kabul testi.
  - İnceleme (review) ve denetimler.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetiminde Kullanılan Standartlar**

1. **ISO 9001:** Kalite yönetim sistemleri için uluslararası bir standart.
2. **CMMI (Capability Maturity Model Integration):** Yazılım geliştirme süreçlerini değerlendirmek ve iyileştirmek için kullanılan bir model.
3. **Six Sigma:** Hata oranını azaltmayı ve süreçleri optimize etmeyi hedefleyen bir yöntem.
4. **IEEE Standartları:** Yazılım mühendisliği için tanımlanmış kalite gereksinimleri ve metrikler sunar.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetiminde Kullanılan Metrikler**
- ✓ Kalite yönetimi sürecinde, yazılımın performansını ve güvenilirliğini ölçmek için çeşitli metrikler kullanılır:

1. **Hata Yoğunluğu:** Belirli bir kod birimi başına bulunan hata sayısı.
2. **Kod Kapsamı:** Testlerin, yazılım kodunun ne kadarını kapsadığını ölçer.
3. **Performans Metrikleri:** Yazılımın hızı, kaynak kullanımı ve yanıt süresi gibi ölçütler.
4. **Güvenilirlik Metrikleri:** Yazılımın hatasız çalışma oranını ve hata süresini ölçer.
5. **Müşteri Memnuniyeti:** Yazılımın müşteri beklentilerini ne derece karşıladığını belirler.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetiminin Süreci**
- ✓ **1. Gereksinim Belirleme**
- Kalite yönetimi süreci, yazılım gereksinimlerini doğru anlamakla başlar.
- Gereksinimlere uygun kalite kriterleri tanımlanır.
- ✓ **2. Kalite Planının Hazırlanması**
- Kalite hedefleri belirlenir.
- Kullanılacak araçlar, teknikler ve test yöntemleri tanımlanır.
- ✓ **3. Geliştirme Sürecinin İzlenmesi**
- Süreç boyunca geliştiricilerin belirlenen kalite standartlarına uyup uymadığı izlenir.
- Kalite güvence ekibi, süreç denetimleri gerçekleştirir.

Yazılım Kalite Yönetimi

- ✓ **Yazılım Kalite Yönetiminin Süreci**
- ✓ **4. Test ve Doğrulama**
- Ürün, kalite kontrol ekibi tarafından test edilir.
- Testler sonucunda bulunan hatalar düzeltici faaliyetlerle giderilir.
- ✓ **5. Müşteri Onayı ve Dağıtım**
- Ürün, müşteri kabul testlerine tabi tutulur.
- Müşteri memnuniyeti sağlandığında ürün teslim edilir.
- ✓ **6. Sürekli İyileştirme**
- Geçmiş projelerden öğrenilen derslerle kalite süreçleri iyileştirilir.
- Performans verileri analiz edilerek yeni standartlar belirlenir.

Yazılım Kalite Yönetimi

✓ **Yazılım Kalite Yönetiminin Faydaları**

1. **Müşteri Memnuniyeti:** Yüksek kaliteli ürünler, müşteri ihtiyaçlarını daha iyi karşılar.
2. **Maliyet Azaltma:** Hataların erken aşamalarda tespit edilmesiyle, maliyetli düzeltmelerden kaçınılır.
3. **Güvenilirlik:** Yazılımın kararlılığı artar, kullanım sırasında sorunlar azalır.
4. **Rekabet Avantajı:** Kaliteli yazılım, piyasada daha yüksek bir değer yaratır.
5. **Süreç İyileştirme:** Kalite yönetimi, yazılım geliştirme süreçlerinin sürekli iyileştirilmesini teşvik eder.

Yazılım Kalite Yönetimi – Örnek Uygulama

✓ **Proje Özeti**

- **Proje Adı:** "ShopEasy" E-ticaret Platformu
- **Amaç:** Kullanıcıların ürün arama, sipariş verme ve ödeme işlemlerini kolayca yapabileceği bir platform geliştirmek.
- **Hedef:** Yüksek kullanıcı memnuniyeti ve minimum hata oranı ile güvenilir bir sistem oluşturmak.
- **Takım:** Yazılım mühendisleri, test mühendisleri, proje yöneticileri ve kalite güvence uzmanları.

Yazılım Kalite Yönetimi – Örnek Uygulama

- ✓ **Yazılım Kalite Yönetimi Uygulaması**
- ✓ **1. Kalite Planlama (Quality Planning)**
- ✓ Kalite yönetiminin ilk adımı, hedefleri belirlemek ve bu hedeflere ulaşmak için bir plan yapmaktır.

1. **Kalite Hedefleri:**
   1. Platformun %99,9 oranında çalışma süresi (uptime) sağlaması.
   2. Sayfa yüklenme hızının maksimum 2 saniye olması.
   3. Kullanıcılar tarafından bildirilen hata oranının %1'in altında tutulması.
   4. Ödeme işlemlerinde %100 güvenlik.
2. **Kalite Standartları:**
   1. ISO/IEC 25010 (Yazılım Ürün Kalitesi Modeli).
   2. OWASP standartlarına uygun güvenlik kontrolleri.
3. **Araç ve Teknikler:**
   1. **Kod Kalitesi:** SonarQube, Codacy gibi araçlarla kod analizi.
   2. **Test Süreçleri:** Selenium ile otomasyon testleri, JMeter ile yük testleri.

Yazılım Kalite Yönetimi – Örnek Uygulama

- ✓ **Yazılım Kalite Yönetimi Uygulaması**
- ✓ **2. Kalite Güvence (Quality Assurance - QA)**
- ✓ Kalite güvence, yazılım geliştirme süreçlerinin belirlenen standartlara uygun şekilde yürütülmesini sağlar.

1. **Süreç Denetimi:**
   1. Yazılım geliştirme sırasında, belirlenen yazılım yaşam döngüsü süreçlerine (örneğin Agile) uyulması denetlenir.
   2. Sprint planlama toplantılarında kalite gereksinimleri gözden geçirilir.
2. **Kod İncelemeleri:**
   1. Kod, birinci aşamada yazılım mühendisleri tarafından yazılır.
   2. Yazılım kalite uzmanları ve ekip liderleri, kodun kalite standartlarına uygunluğunu manuel olarak inceler.
3. **Dökümantasyon:**
   1. Gereksinim dokümanları ve tasarım belgeleri QA ekibi tarafından kontrol edilerek eksiksiz ve açık olduğu doğrulanır.

Yazılım Kalite Yönetimi – Örnek Uygulama

- ✓ **Yazılım Kalite Yönetimi Uygulaması**
- ✓ **3. Kalite Kontrolü (Quality Control - QC)**
- ✓ Ürünün kalite gereksinimlerini karşılayıp karşılamadığını test etmek için kontrol süreçleri uygulanır.

1. **Test Süreçleri:**
    1. **Fonksiyonel Test:** Kullanıcıların ürün arama, sepet ekleme ve sipariş verme gibi işlemlerini sorunsuz yapabildiği test edilir.
    2. **Performans Testi:** JMeter ile simüle edilen yüksek kullanıcı yükünde sistemin yanıt süresi ölçülür.
    3. **Güvenlik Testi:** OWASP ZAP ile SQL enjeksiyonu, XSS (Cross-site scripting) gibi güvenlik açıkları test edilir.
    4. **Kabul Testi:** Müşteri tarafından belirlenen senaryoların başarıyla tamamlanıp tamamlanmadığı kontrol edilir.
2. **Hata Yönetimi:**
    1. Tespit edilen her hata, Jira gibi bir hata takip sistemi kullanılarak kaydedilir.
    2. Hatalar önceliklendirilir (kritik, orta, düşük) ve çözülmesi için yazılım ekibine atanır.
    3. Çözülen hatalar, tekrar test edilerek doğrulanır.

Yazılım Kalite Yönetimi – Örnek Uygulama

- ✓ **Yazılım Kalite Yönetimi Uygulaması**
- ✓ **4. Sürekli İzleme ve İyileştirme**
- ✓ Projenin tamamlanmasından sonra bile kalite yönetimi devam eder.

1. **Canlı Sistem İzleme:**
   1. Sistem performansı, uptime süresi ve hata oranları New Relic gibi araçlarla sürekli izlenir.
   2. Kullanıcı geri bildirimleri analiz edilir ve iyileştirmeler için yol haritası oluşturulur.
2. **Köklü İyileştirme (Root Cause Analysis):**
   1. Önemli bir hata yaşanması durumunda, nedenlerini belirlemek için kök neden analizi yapılır.
   2. Süreçlerde veya sistemlerde gereken değişiklikler uygulanır.
3. **Ders Çıkarma:**
   1. Proje boyunca toplanan kalite verileri analiz edilir.
   2. Gelecek projelerde aynı hataların tekrarlanmaması için öneriler hazırlanır.

Yazılım Kalite Yönetimi – Örnek Uygulama

✓ **Sonuçlar**

- **Başarıyla Tamamlanan Süreçler:**
  - Platform, %99,95 çalışma süresi sağladı.
  - Ortalama sayfa yüklenme süresi 1,8 saniyede tutuldu.
  - Kullanıcı şikayetlerinde önceki benzer projelere göre %30 azalma gözlemlendi.
- **Kazançlar:**
  - Platformun yüksek kaliteli olması, müşteri memnuniyetini artırdı.
  - Güvenilir ve hızlı bir sistem, şirketin marka değerine olumlu katkı sağladı.
  - Sürekli izleme ve iyileştirme sayesinde sistemin uzun vadeli sürdürülebilirliği sağlandı.

Kaynaklar

✓-*Doç. Dr. Recep ERYİĞİT*

✓*Doç. Dr. Resul DAŞ*

✓*Doç. Dr. Fatih ÖZKAYNAK*