



2025-2026 AKADEMİK YILI GÜZ DÖNEMİ

BİL403 YAZILIM MÜHENDİSLİĞİ

8. DERS

Dr. Öğr. Üyesi Ertürk ERDAĞI
erturk.erdagi@medeniyet.edu.tr

Yazılım Testi Nedir?

- Yazılım testi, geliştirilmiş bir yazılım ürününün **beklenen davranışını gösterip göstermediğini kontrol etmek** için yapılan tüm faaliyetlerin bütünüdür. Temel amaç; yazılımın hatasız olduğunu kanıtlamak değil, **hataları ortaya çıkarmaktır.**
- Modern yazılım projelerinde test, yalnızca geliştirme sonrası yapılan bir etkinlik değil; **planlama aşamasından ürünün canlı kullanımına kadar** süren bir kalite güvencesi sürecidir.
- Doğrulama (Verification) ve Geçerleme (Validation): Temel Fark
- **Verification : “Doğru ürünü doğru yapıyor muyuz?”**
 - Yazılımın gereksinimlere uygun geliştirilip geliştirilmediği incelenir.
 - Dokümanlar, tasarımlar, kodlar kontrol edilir.
 - Denetim, gözden geçirme, walkthrough gibi teknikler içerir.
 - Kod çalıştırılmaz.
- **Validation : “Doğru ürünü yapıyor muyuz?”**
 - Kullanıcının ihtiyaçlarını gerçekten karşılayıp karşılamadığı test edilir.
 - Sistem çalıştırılarak doğrulama yapılır.
 - Son kullanıcı perspektifi önemlidir.

Yazılım Testi Nedir?

- Temel Kavramlar: Error, Fault, Failure
- **Error (Hata Nedeni / İnsan Kaynaklı Hata)**
 - Geliştiricinin veya analistin yaptığı düşünme hatasıdır.
 - Örnek: Algoritmada unutulan bir koşul.
- **Fault / Bug (Hata / Kusur)**
 - Error sonucu koda yansıyan yanlışlıktır.
 - Örnek: \geq yerine $>$ yazılması.
- **Failure (Arıza / Yanlış Davranış)**
 - Fault çalışırken kullanıcıya yansığında oluşur.
 - Örnek: Banka uygulamasının yanlış bakiye göstermesi.

Yazılım Testinin Amaçları

- Yazılım testi yalnızca hata bulmak için değil, yazılımın tüm yönleriyle kalitesini garanti altına almak için yapılır. Başlıca amaçlar:
- **1. Hataları Tespit Etmek**
 - Yazılımda bulunan kusurların keşfedilmesi.
 - Ne kadar erken bulunursa maliyet o kadar düşer.
- **2. Yazılım Kalitesini Artırmak**
 - Test süreci, yazılımın güvenilirliği, kullanılabilirliği ve bütünlüğünü yükseltir.
- **3. Riskleri Azaltmak**
 - Güvenlik açıkları
 - Performans sorunları
 - Veri kayıpları gibi risklerin önceden tespit edilmesi.
- **4. Müşteri Memnuniyetini Sağlamak**
 - Kullanıcının ihtiyaçlarına uygun, çalışır, stabil bir ürün üretmek.
- **5. Ürün Yeterliliğini Kanıtlamak**
 - Gereksinimlerin karşılanıp karşılanmadığını göstermek.
 - Kabul testlerinde başarı elde etmek.

Test Neden Gereklidir? (Maliyet – Zaman – Kalite İlişkisi)

- Modern yazılım mühendisliğinde “erken test” prensibi kritik önem taşır.
- Hatanın Bulunma Maliyeti
- Aynı hata farklı aşamalarda farklı maliyet taşır:

Aşama	Hata Bulma Maliyeti
Analiz	Çok düşük
Tasarım	Düşük
Kodlama	Orta
Test	Yüksek
Ürün canlıda	Çok yüksek / kritik

- Örneğin; canlıdaki bir hata
- marka itibarını zedeler
- maddi kayıp oluşturabilir
- güvenlik riski doğurabilir
- Bu nedenle **test, maliyet azaltma stratejisidir.**

Yazılım Geliştirme Yaşam Döngüsünde Testin Rolü

- Test, Waterfall ve Agile hangi modeli kullanırsanız kullanın, tüm süreçlerde yer alır:
 - Gereksinim doğrulama
 - Tasarım incelemeleri
 - Birim testi
 - Entegrasyon testi
 - Sistem testi
 - Kabul testi
 - Regresyon testleri
- Özellikle **Agile** dünyasında test sürekli tekrarlanır ve ‘done’ tanımının merkezinde yer alır.
- **Testin Yazılım Kalitesine Etkisi**
 - Yazılım kalitesi; sadece işlevsellik değil:
 - Performans
 - Güvenlik
 - Kullanılabilirlik
 - Taşınabilirlik
 - Doğruluk
 - Güvenilirlik gibi pek çok ölçüte bağlıdır. Test süreçleri bu metriklerin doğrulandığı temel mekanizmadır.

Testin Gereksiz Olduğu Yanılgısı

- Bazı geliştiriciler tarafından düşünülen hatalı inanışlar:
 - “Kodum zaten çalışıyor.”
 - “Test yazmak zaman kaybı.”
 - “Küçük projelerde teste gerek yok.”
 - “Son kullanıcı test eder.”
- Profesyonel projelerde bu yaklaşım **büyük kayıplara yol açar**. Örneğin:
 - Knight Capital (2012) yazılım hatası → 45 dakikada 440 milyon dolar zarar
 - NASA Mars Climate Orbiter (1999) birim dönüşüm hatası → 327 milyon dolar kayıp
- **Bir e-ticaret sitesinde: “Miktarı girilen ürünün stok kontrolü düzgün çalışmadığı için kullanıcı eksik veya fazla ürün alabiliyor.”**
 - Bu durum Error / Fault / Failure tiplerinden hangilerine örnektir?
 - Bu hata hangi aşamada daha düşük maliyetle bulunabilirdi?
 - Bu örneğe uygun test senaryaları neler olabilir?

Test Süreci ve Test Yaşam Döngüsü

- Test süreci aşağıdaki aşamalardan oluşur:
 - Test Planlama (Test Planning)
 - Test Analizi ve Test Tasarımı
 - Test Ortamının Hazırlanması
 - Test Verisi Hazırlama
 - Test Uygulama (Execution)
 - Hata Raporlama (Defect Tracking)
 - Testin İzlenmesi ve Kontrolü
 - Test Sonlandırma (Closure)
- Bu döngü, V-Model, Waterfall ve Agile süreçleri için geçerlidir ancak uygulama derinliği modele göre değişir.

Test Planlama

- **Amaç:** Test stratejisinin belirlenmesi ve test kapsamının tanımlanması.
- **Bu aşamada yapılanlar:**
 - Test stratejisi ve yaklaşımı belirlenir (ör. risk bazlı test).
 - Test kapsamı ve kapsam dışı alanlar tanımlanır.
 - Test seviyeleri seçilir (Unit, Integration, System, Acceptance).
 - Gerekli kaynaklar (insan, zaman, araçlar) belirlenir.
 - Test araçları kararlaştırılır (Selenium, JUnit, Postman, JMeter vb.)
 - Test ortamı gereksinimleri hazırlanır.
 - Başarı kriterleri (exit criteria) belirlenir.
 - Test zaman çizelgesi oluşturulur (Gantt chart).
- **Örnek:**
 - Bir bankacılık uygulamasının para transferi modülü için test planında:
 - Güvenlik testleri (OWASP top 10)
 - Fonksiyonel testler
 - Performans testleri (yük ve stres testi) zorunlu olarak belirtilmelidir.

Test Analizi & Test Tasarımı

- Amaç: Gereksinimler doğrultusunda test senaryolarının oluşturulması.
- Bu aşamada yapılanlar:
 - Gereksinim analizi
 - Risk analizi
 - Test durumlarının belirlenmesi
 - Test adımlarının yazılması
 - Expected result (beklenen sonuç) oluşturma
 - Gereksiz testlerin elenmesi
 - Test veri ihtiyaçlarının belirlenmesi
- Kullanılan teknikler:
 - Eşdeğerlik Bölümleme
 - Sınır Değer Analizi
 - Karar Tabloları
 - Durum Geçiş Diyagramları
- Örnek Test Case (Kısa):
 - **Test Case:** Minimum bakiye yetersizse EFT işlemi reddedilmeli
 - Expected: “Bakiye yetersiz” uyarısı

Test Ortamının Hazırlanması ve Test Verisi Hazırlama

- **Test Ortamının Hazırlanması :** Testlerin güvenilir şekilde yapılacağı fiziksel/veri ortamının kurulması.
- **İçerik:**
 - Sunucuların yapılandırılması
 - Uygulama sürümünün yüklenmesi
 - Veritabanı bağlantılarının sağlanması
 - API endpoint'lerinin aktif hale getirilmesi
 - Test kullanıcılarının oluşturulması
 - Bağımlılıkların (cache, queue, microservice) hazırlanması
 - “Test ortamı = Canlı ortamın gerçekçi bir kopyası” olmalıdır.
- **Test Verisi Hazırlama :** Test senaryolarının koşabileceği veri setlerinin oluşturulması.
- **Veri Kaynakları:**
 - Gerçek veriden maskeleme yapılmış setler
 - Sentetik (üretilmiş) test verisi
 - Otomatik veri üreticiler (Faker gibi)
 - Edge-case veri üretimi
- **Örnek:**
 - Banka hesap testlerinde
 - 0 hakiye, 1 TL hakiye, Milisaniyelik zamanlı işlemler, 99.999 TL transfer limiti gibi sınırlı değerli test verileri hazırlanmalıdır.

Test Uygulama ve Hata Raporlama

- **Test Uygulama :** Yazılmış testlerin çalıştırılması ve sonuçların kaydedilmesi. Bu aşamada:
 - Test case'ler sırayla çalıştırılır
 - Test sonuçları kaydedilir (pass / fail)
 - Hatalar loglanır
 - Test raporları güncellenir
 - Otomasyon testleri tetiklenir
- **Otomasyon Örneği:**
 - Selenium ile UI testleri, JUnit ile birim test, Postman/Newman ile API testleri, JMeter ile performans testi
- **Hata Raporlama :** Bulunan hataların standart bir formatta kayıt altına alınması. Bir hata raporunda yer alması gerekenler:
 - Başlık (Summary)
 - Adımlar (Steps to Reproduce)
 - Beklenen sonuç, Gerçekleşen sonuç
 - Önem derecesi (Severity), Öncelik (Priority)
 - Ekran görüntüleri
 - Ortam bilgisi (Environment)
- **Severity – Priority farkı:**
 - **Severity:** Teknik etki

Testin İzlenmesi - Kontrolü ve Test Sonlandırma

- **Testin İzlenmesi - Kontrolü :** Test sürecinin hedeflere uygun ilerleyip ilerlemediğini izlemek. Bu aşamada:
 - Test ilerleme raporları hazırlanır
 - Açık/kapatılan hata istatistikleri takip edilir
 - Test başarı oranı hesaplanır
 - Riskler yeniden değerlendirilir
 - Gerektiğinde test planı güncellenir
- **Yaygın kullanılan metrikler:**
 - Passed test sayısı
 - Failed test sayısı
 - Açık hata sayısı
 - Fix rate (tamir oranı)
 - Test coverage (kapsama oranı)
- **Test Sonlandırma (Test Closure) :** Test sürecinin resmi olarak tamamlanması. Kapanış adımları:
 - Test raporlarının hazırlanması
 - Bulunan hataların sınıflandırılması
 - Test sonuçlarının yönetimle paylaşılması
 - Çıkarılan dersler (lessons learned)
 - Test dokümanlarının arşivlenmesi
 - Test ortamının kapatılması
- **Cıktılar:**

Test Türleri

- Test türleri, yazılımın hangi bakış açısıyla, hangi kapsamda ve hangi seviyede değerlendirileceğini belirler.
- **1. Beyaz Kutu Testleri (White-Box Testing)**
- Beyaz kutu testleri, sistemin **İç yapısını, kontrol akışını ve kodun mantıksal yapısını** dikkate alarak yapılır. Geliştiriciler veya teknik test mühendisleri tarafından uygulanır.
- Kodun her yolunun çalıştığından emin olmak
- Gizli hataları bulmak (dead code, unreachable code, infinite loop)
- Koşul ve karar yapılarının doğruluğunu kontrol etmek
- **Statement Coverage (Deyim Kapsaması)** : Kodun içindeki her bir satırın/ifadenin en az bir kez çalıştırılması hedeflenir.
- **Branch Coverage (Dal Kapsaması)** : Koşullu ifadelerin (if/else, switch-case) tüm dallarının test edilmesidir.
- **Condition Coverage** : Bir koşulun tüm mantıksal bileşenlerinin *true* ve *false* olma durumları test edilir.
- **Path Coverage** : Kodun içerisindeki tüm **olası yürütme yollarının** test edilmesi. En kapsamlı ve maliyetli beyaz-kutu tekniğidir.
- **Unit Test (Birim Testi)** : Bir fonksiyonun, sınıfın veya metodun tek başına test edilmesidir. Python: pytest

- **2. Siyah Kutu Testleri (Black-Box Testing)**
- Siyah kutu testlerinde sistemin iç yapısı bilinmez; yalnızca **girdiler, çıktılar ve beklenen davranış** incelenir.
- Fonksiyonel testlerin çoğu siyah-kutudur.
- **Eşdeğerlik Bölütleme (Equivalence Partitioning)**
 - Girdi alanları eşdeğer sınıflara bölünür ve her sınıftan yalnızca 1 değer test edilir.
 - **Örnek:** Yaş doğrulama: 0–120 arası geçerli. Eşdeğerlik sınıfları: Negatif yaş (Geçersiz), 0–120 arası (Geçerli), 120'den büyük (Geçersiz)
 - Bu durumda sadece **3 test** yeterlidir.
- **Sınır Değer Analizi (Boundary Value Analysis)**
 - Hataların büyük kısmı **sınır değerlerde** ortaya çıktığı için sınır noktaları test edilir.
- **Karar Tablosu Tabanlı Test (Decision Table Testing)**
 - Çok sayıda koşulun farklı kombinasyonlarının olduğu sistemlerde kullanılır.
- **Durum Geçiş Testi (State Transition Testing)**
 - Durum makineleri kullanan sistemlerde geçiş kurallarının test edilmesidir. **Örnek:** ATM kart akışı:
- **Kullanılabilirlik Testi (Usability Testing)**
 - Kullanıcı deneyimi, erişilebilirlik, ekran tasarımlarının değerlendirilmesi.

Test Türleri

- **3. Gri Kutu Testleri (Gray-Box Testing)**
- Gri kutu test, hem siyah hem beyaz kutunun birleşimidir. Tester, sistemin iç yapısını **kısmen** bilir.
- **Kullanım Alanları**
 - Entegrasyon testleri
 - API testleri
 - Web uygulamalarında session handling
 - Cache davranışları
 - Veritabanı sorgusu doğrulama
- **Örnek:**
 - Bir kullanıcının ödeme yaptığı ekranın:
 - UI davranışı → siyah kutu
 - API'nin döndürdüğü JSON yanıtı → beyaz kutu incelenir.

Test Seviyeleri

- Test seviyeleri, yazılım geliştirme sürecinde testlerin **hangi aşamada, hangi kapsamda ve hangi amaçla** yapılacağını tanımlar. Bu seviye yapısı, özellikle **V-Model, Waterfall, Agile** süreçlerinde ortak bir test çerçevesi oluşturur.
- Klasik olarak 4 ana test seviyesi vardır:
 - **Unit Test (Birim Testi)**
 - **Integration Test (Entegrasyon Testi)**
 - **System Test (Sistem Testi)**
 - **Acceptance Test (Kabul Testi)**
- Ek olarak bazı metodolojiler:
 - Regression Test
 - Smoke Test
 - Sanity Test
 - Alpha/Beta Testleri gibi seviyeleri de destekleyici katmanlar olarak kabul eder.

Unit Test – Birim Testi

- Yazılımın en küçük parçalarını (fonksiyon, sınıf, method) **tek başına** test etmektir.
- **Özellikleri**
 - En erken uygulanan test seviyesidir.
 - Geliştiriciler tarafından yapılır.
 - Kod tabanlıdır.
 - Otomasyona en uygun test türüdür.
 - Hataları en düşük maliyetle yakalar.
- **İçerik**
 - Fonksiyonun doğru sonuç üretip üretmediği
 - Kenar/sınır durum testleri
 - Exception (hata fırlatma) testleri
- **Fonksiyon:** hesaplaIndirim(fiyat, indirimOrani)
 - fiyat = 100, indirim = %10 → beklenen: 90
 - fiyat = 0 → beklenen: 0
 - negatif indirim → exception beklenmeli

- Birimler tek başına sorunsuz çalışsa bile birbirleriyle entegrasyonlarında hatalar çıkabilir. Bu seviyede amaç **birimlerin birbiriyle nasıl iletişim kurduğunu test etmektir.**
- **Özellikleri**
 - Birimler, modüller veya servisler arasındaki veri akışı test edilir.
 - API → DB → Service zinciri kontrol edilir.
 - Gri kutu test yaklaşımı yaygındır.
 - Bağımlılıklar nedeniyle karmaşıktır.
- **Entegrasyon Stratejileri**
 - **Top-Down Integration**
 - Üst modüllerden başlanır, alt modüller mock'lanır.
 - **Bottom-Up Integration**
 - En alt modüllerden başlanır, üst modüller test edilir.
 - **Big Bang Integration**
 - Tüm birimler aynı anda birleştirilir. (Tavsiye edilmez)
- **Örnek**
 - Bir e-ticaret uygulamasında:
 - Ürün servisi + Sepet servisi + Ödeme servisi birlikte test edilir.
 - API yanında beklenen JSON formatı doğrulanır.

System Test – Sistem Testi

- Tamamlanmış yazılım ürünü bütünüyle test edilir. Hem fonksiyonel hem de fonksiyonel olmayan testlerin çoğu bu seviyede yapılır.
- **Amaç**
 - “Sistem tümüyle doğru çalışıyor mu?”, Ürün gereksinimleri karşılıyor mu?
- **Özellikleri**
 - QA/Test ekibi tarafından yapılır.
 - Siyah kutu teknigi yaygındır.
 - Donanım, yazılım, network birlikte değerlendirilir.
 - Kullanıcı senaryoları (end-to-end) çalıştırılır.
- **Kapsamı**
 - Fonksiyonel test
 - Performans testi
 - Güvenlik testi
 - Kullanılabilirlik testi
 - Yük testi (Load)
 - Stres testi (Stress Test)
- **Örnek**
 - Bir bankacılık mobil uygulamasında: Para transferi, Bakiye görüntüleme, QR ile ödeme, Bağlantı kopması simülasyonu bir bütün olarak doğrulanır.

Acceptance Test – Kabul Testi

- Acceptance Test, uygulamanın son kullanıcı veya müşteri tarafından **kabul edilip edilmeyeceğini** belirler.
- **Amaç**
 - Yazılımın müşteri gereksinimlerini karşılayıp karşılamadığını doğrulamak
 - Gerçek kullanım senaryolarını test etmek
 - Ürünün canlı kullanıma hazır olduğuna karar vermek
- **Aşamaları**
 - **UAT – User Acceptance Testing** : Son kullanıcılar tarafından yapılan test.
 - **BAT – Business Acceptance Testing** : İş birimi kullanıcıları yapar.
 - **Regression Acceptance Testing** : Yeni sürüm sonrası kritik alanlar test edilir.
 - **Contract Acceptance Test** : Sözleşmedeki maddelere göre test yapılır.
- **Örnek**
 - Bir ödeme sisteminde:
 - Kullanıcı ödeme yapabiliyor mu?
 - Doğrulama kodu geliyor mu?
 - Fiş e-mail olarak iletiliyor mu?
 - Acceptance testlerinde teknik detaydan çok **iş akışı** önemlidir.

Destekleyici Test Seviyeleri

- Bu seviyeler ana 4 seviyeyi destekleyen, sık kullanılan test türleridir.
- **1. Regression Test :** Yeni bir geliştirme yapıldığında, mevcut özelliklerin bozulup bozulmadığını test etmek. Otomasyon için en ideal test türlerinden biridir. **Örnek:** Sepete ekleme özelliğini güncelledikten sonra → Oturum açma, ödeme, bildirim vb. fonksiyonların etkilenip etkilenmediği test edilir.
- **2. Smoke Test :** Sistem ilk kez deploy edildiğinde, temel fonksiyonların çalışıp çalışmadığını hızlıca kontrol etmek için yapılan testtir. **Örnek:** Login oluyor mu?, Anasayfa açılıyor mu?, Menü çalışıyor mu?, 10–15 dakikalık hızlı test.
- **3. Sanity Test :** Eğer küçük bir düzeltme yapıldıysa (bug fix), sadece **ilgili fonksiyon** hızlıca test edilir. **Örnek:** Kampanya fiyatı yanlış hesaplanıyorsa, sadece fiyat hesaplama bölümüne kısa test yapılır.
- **4. Alpha Testi :** Geliştirici veya firma içindeki test ekibi yapar. Ürün henüz tam kullanıcıya sunulmadan uygulanır.
- **5. Beta Testi :** Ürün gerçek kullanıcılarla kontrollü bir şekilde sunulur. Gerçek kullanım geri bildirimleri alınır. **Örnek:** WhatsApp beta programı.

Test Ortamı

- Test ortamı, yazılımın test edilmesi için gerekli olan yazılım, donanım, veritabanı, servisler, ağ yapılandırması ve test araçlarının birlikte çalıştığı sistemdir.
- **Test Ortamının Özellikleri**
 - Profesyonel bir test ortamı şu özelliklere sahip olmalıdır:
 - **Canlı (Production) ortamı mümkün olduğunca taklit etmeli**
 - Versiyon yönetimi doğru yapılmalı (Deploy edilen sürüm numaraları)
 - Veritabanı, API, cache, message queue gibi bileşenler test versiyonlarıyla çalışmalı
 - Test kullanıcı hesapları önceden tanımlanmış olmalı
 - Bağımlı olduğu servisler (Auth, Payment, Notification vb.) konfigüre edilmeli
 - Test sırasında loglama aktif olmalı
 - Ölçümleme araçları (Grafana, Kibana) açık olmalı
- **Test Ortamı Çeşitleri**
 - **1. DEV (Development Environment)** : Geliştiricilerin geliştirme yaptığı ortam. Stabil değildir
 - **2. QA / TEST Environment** : Asıl testlerin yapıldığı ortam. Test mühendisleri tarafından kullanılır
 - **3. PRE-PROD / STAGING** : Canlıya (Production) birebir benzeyen ortam. Performans ve yük testleri için ideal
 - **4. PRODUCTION (Canlı Ortam)** : Son kullanıcıların eriştiği ortam. Bazı kritik sistemlerde *canlıda smoke test* yapılır

Test Verisi Kaynakları

- **1. Gerçek Veriden Maskeleme (Masking)** : Gerçek veri → anonimleştirilir Örnek:
İsim → “A**** Y****”, TCKN → Rastgele üretim, IBAN → Test IBAN formatı
- Avantaj: Gerçeğe yakın veri Dezavantaj: Maskeleme hataları risklidir
- **2. Sentetik Veri (Synthetic Data)** : Veri otomatik olarak üretilir. Örnek: Rastgele isim, Sahte kredi kart numarası, Farklı yaş aralıkları
- **3. Özel Amaçlı Test Veri Setleri**
 - Boundary test verileri
 - Negatif test verileri
 - Performans testi için büyük veri (Load Data)
 - Hata üretmek için özel veri (Error Injection Data)

Test Otomasyonu

- Test otomasyonu, yazılım testlerinin otomatik araçlar ve çerçeveler kullanılarak makine tarafından çalıştırılmasıdır. Modern yazılım geliştirmede kaçınılmazdır.
- **Test Otomasyonunun Amacı**
 - Regresyon testlerini otomatikleştirmek
 - Manuel test yükünü azaltmak
 - Hızlı geri bildirim sağlamak
 - CI/CD pipeline ile entegrasyon
 - Ürünün kalitesini artırmak
 - Tekrarlanabilirlik sağlamak
 - İnsan hatasını azaltmak

- Proje Teslimi : 21 Aralık Pazar saat 22.00
- Proje Teslim Yeri : Google Classroom
- **Teslim Edilecek Dosyalar : Proje Raporu, Sunum dosyası, Proje Github linki**
- Proje Sunumları : 24 Aralık Çarşamba – 31 Aralık Çarşamba – 7 Ocak Çarşamba
(Kimlerin ne zaman sunacakları 17 Aralık günü derste kura yöntemiyle belirlenecektir)
- **Proje sunumu yapılmayan projeler teslim edilmemiş olarak değerlendirilecektir.**
- Proje Türü : Bireysel, iki kişilik ya da üç kişilik projeler. **Ekip üyeleri teslimde ve sunumda birlikte sorumludur. 3 Aralık günü derste ekip üyelerinin bilgileri alınacaktır.**
- Projenin Amacı : Bu dönem projesinin amacı, öğrencilerin yazılım mühendisliği süreçlerini uçtan uca deneyimleyerek modern teknolojileri kullanarak gerçek bir yazılım çözümü geliştirmelerini sağlamaktır.
- **Proje basit bir ödev değildir; bir mini araştırma projesidir. Ders notunun %18'ini kapsar.**

- Öğrenciler:
 - Gereksinim analizi yapacak,
 - Uygun yazılım geliştirme metodolojisini seçecek,
 - Tasarım ve mimari oluşturacak,
 - Son teknolojilerle prototip/proje geliştirecek,
 - Test–doğrulama süreçlerini uygulayacak,
 - Güvenlik ve kalite unsurlarını değerlendirecek,
 - Projelerini raporlayacak ve sınıfta sunacaklardır.
- **Basit uygulamalar kabul edilmeyecektir.** Projede **modern teknolojiler** kullanmalı ve özgün bir problem çözmeliidir.

- Öğrenciler proje konusunu **kendileri** belirler.
Projenin güncel teknoloji kullanma zorunluluğu vardır. Aşağıdaki teknoloji alanlarından en az bir tanesi kullanılmalıdır:
- **Önerilen Teknoloji Alanları**
 - **Bulut Tabanlı Uygulamalar** (AWS, GCP, Azure, Firebase)
 - **Mikroservis Mimarisi**
 - **Konteyner Teknolojileri** (Docker, Kubernetes)
 - **Yapay Zekâ / Makine Öğrenmesi entegrasyonu**
 - **NLP, Görüntü İşleme, LLM tabanlı servisler**
 - **Sunucusuz (Serverless) mimari**
 - **Modern Web Çatılar** (React, Next.js, Vue, Angular)
 - **Mobil Uygulama Geliştirme** (Flutter, React Native)
 - **API Tabanlı Uygulamalar**
 - **CI/CD Pipeline kurulumu** (GitHub Actions, GitLab CI)
 - **Güvenli Kod Geliştirme Pratikleri**
 - **Test Otomasyonu** (JUnit, Selenium, Postman/Newman)
- **Basit örnekler kabul edilmez.** Örneğin: Sadece CRUD yapan uygulamalar, Basit hesap makineleri, Sadece veri gösteren web siteleri **geçersiz** sayılacaktır.

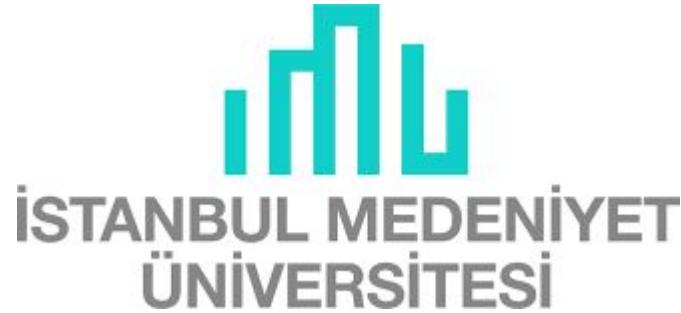
- Rapor (PDF – En az 15 sayfa)
- Proje raporu ve sunumu aşağıdaki bölümlere sahip olmalıdır:
- **1. Gereksinim Analizi**
 - Problemin tanımı
 - Kullanıcı ihtiyaçları
 - Fonksiyonel ve fonksiyonel olmayan gereksinimler
 - Kullanım senaryoları (Use-Case Diagram zorunlu)
- **2. Süreç Modeli**
 - Seçilen geliştirme modeli (Scrum, Kanban, vb.)
 - Gerekçesi
 - Sprint planı veya iş kırılımı
- **3. Yazılım Tasarımı ve Mimari**
 - Genel mimari (Component Diagram / Deployment Diagram önerilir)
 - Veri modeli
 - Sınıf diyagramları
 - UI/UX tasarımları örnekleri
- **4. Gerçekleştirilen Yazılım**
 - Kullanılan teknolojiler
 - Modüller ve işlevler
- **5. Test Süreçleri**
 - Test stratejisi
 - Unit test örnekleri
 - API testleri (Postman vs.)
 - Hata raporları
- **6. Yazılım Kalitesi ve Güvenliği**
 - Kullanılan kalite standartları / metrikler
 - Güvenlik kontrolleri
 - Kod kalite araçları kullanımı (Örn: SonarQube ekranı)
- **7. Sonuç ve Tartışma**
 - Hedeflere ulaşma düzeyi
 - Teknik zorluklar
 - Geliştirilebilecek yönler
- **Kaynaklar IEEE stilinde yazılmalıdır.**

- **Proje Sunumu**
- Her ekip **15 dakika** sunum yapacaktır.
- **Sunum içeriği:**
 - Problem ve amaç
 - Teknoloji seçimi gerekçeleri
 - Mimari tasarım
 - Uygulama demosu
 - Test ve kalite çıktıları
 - Sonuç
 - Ekip üyeleri görev dağılımı / yapılan işler
- **Sunum kuralları:**
 - En fazla 10 slayt
 - Her slayt sade ve anlaşılır olmalı
 - Demosu çalışmayan uygulamalara **puan kesintisi** uygulanacaktır

BİL403 – Yazılım Mühendisliği Dersi Dönem Projesi

31

Bileşen	Açıklama	Puan
1. Gereksinim Analizi	Problem tanımı, gereksinimler, use-case diyagramları	15
2. Süreç Modeli ve Planlama	Doğru model seçimi, sprint/iş planı	10
3. Tasarım ve Mimari	Diyagramlar, tasarım kalıpları, teknoloji mimarisi	20
4. Uygulama Geliştirme	Modern teknoloji kullanımı, fonksiyonların kapsamı	25
5. Test ve Kalite	Test çıktıları, kalite ölçütleri, güvenlik unsurları	10
6. Rapor Kalitesi	Yapı, anlatım, teknik doğruluk, görseller	10
7. Sunum	Anlatım, demo, zaman yönetimi	10



KATILIMINIZ İÇİN TEŞEKKÜR EDERİM.

YOKLAMA İÇİN İMZANIZI ATMAYI UNUTMAYINIZ.
CLASSROOM ÜZERİNDEN SINİFA DAHİL OLmayı UNUTMAYINIZ

Sınıf Kodu : pua2tnoe

Dr. Öğr. Üyesi Ertürk ERDAĞI
erturk.erdagi@medeniyet.edu.tr