İSTANBUL MEDENİYET ÜNİVERSİTESİ

2025-2026 AKADEMİK YILI GÜZ DÖNEMİ

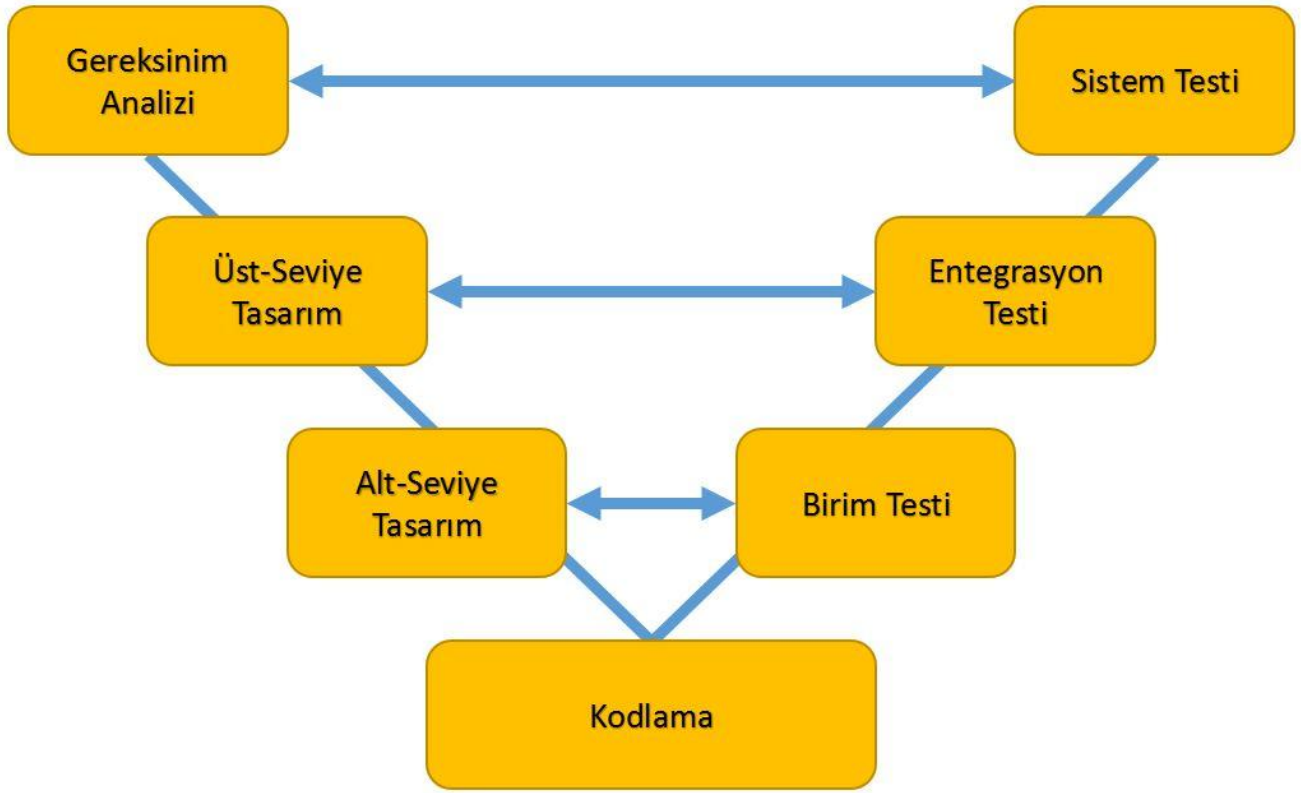
BİL403 YAZILIM MÜHENDİSLİĞİ

3. DERS

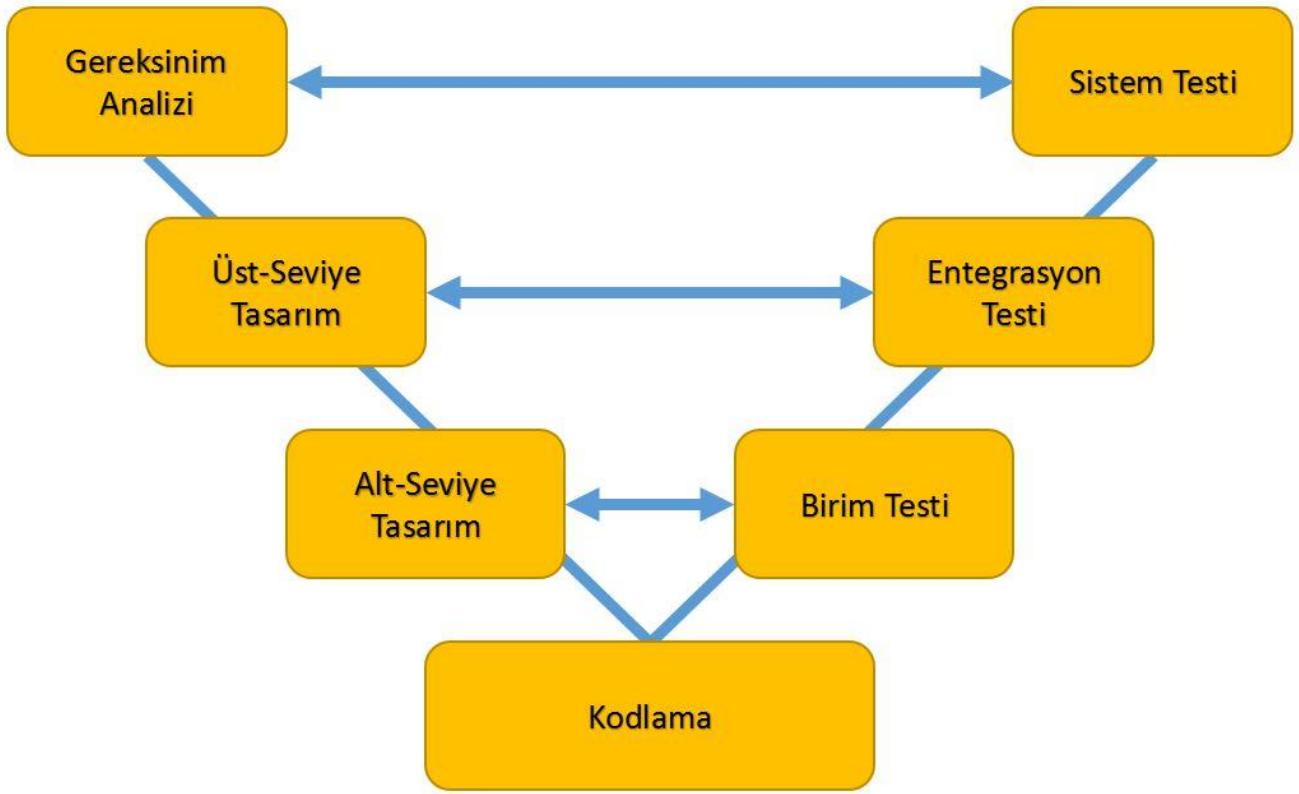
Dr. Öğr. Üyesi Ertürk ERDAĞI
erturk.erdagi@medeniyet.edu.tr

V-Model (Doğrulama ve Geçerleme Modeli)

- Şelale modelinin geliştirilmiş hali, test süreçlerini geliştirme adımlarına paralel olarak tanımlar. Her geliştirme aşamasının eş zamanlı bir test aşaması vardır.
- **Aşamalar ve Test İlişkisi:**
  - Gereksinim → Kabul Testi
  - Sistem Tasarımı → Sistem Testi
  - Mimari Tasarım → Entegrasyon Testi
  - Modül Tasarımı → Birim Testi
- Avantajlar :
  - Testler baştan planlanır
  - Hatalar erken aşamada tespit edilir
  - Daha yüksek kalite sağlanır
- Dezavantajlar :
  - Şelale gibi katıdır, değişime kapalı
  - Karmaşık projeler için esnek değildir
- Kullanım Alanları:
  - Kritik sistemler (havacılık, savunma, tıp yazılımları)



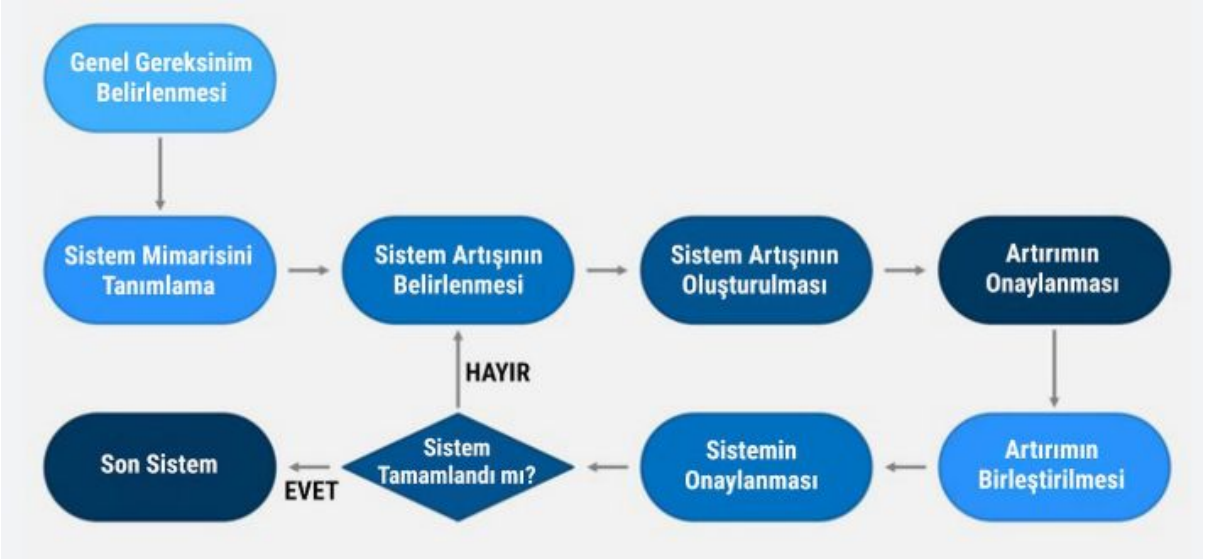
V-Model (Doğrulama ve Geçerleme Modeli)



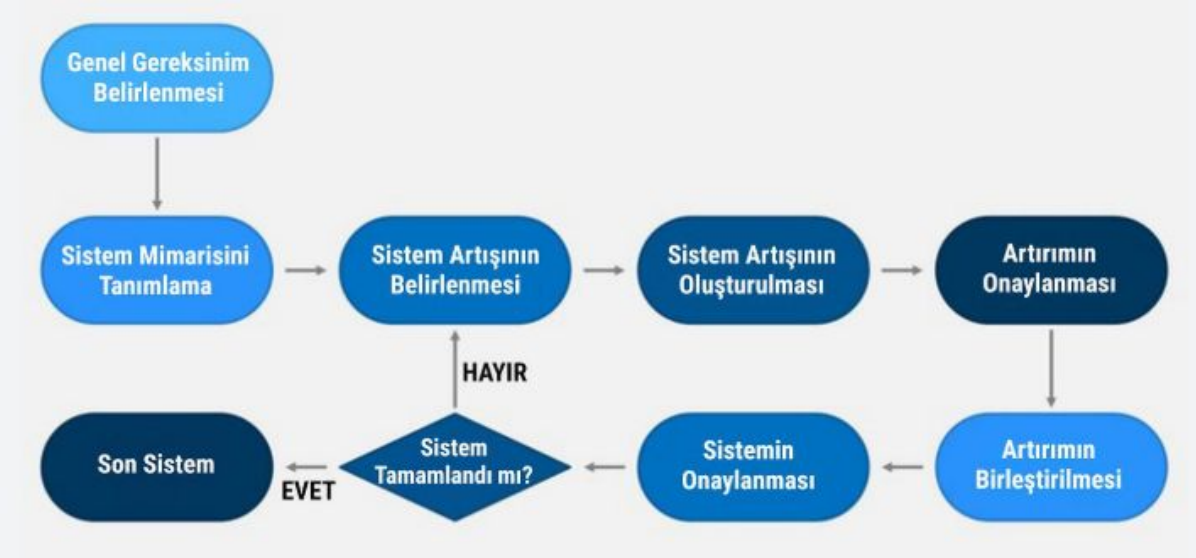
- V-Model'de her geliştirme aşamasının eş zamanlı bir test aşaması vardır. Sizce bu yaklaşım **hataları erken tespit etmek** açısından neden avantajlıdır?
- Kritik sistemlerde (ör. havacılık, sağlık) V-Model'in tercih edilme nedenleri nelerdir?

Artımlı Model (Incremental Model)

- Yazılım, işlevsel parçalar halinde geliştirilir ve her artışta yeni özellikler eklenir. İlk sürüm temel işlevleri içerir, sonraki artışlarla yazılım genişler.
- Avantajlar:
  - Kullanıcı erken sürümü deneyebilir
  - Riskler daha kolay yönetilir
  - Parça parça geliştirme zaman yönetimini kolaylaştırır
- Dezavantajlar:
  - Sistem genel tasarımının dikkatli yapılması gerekir
  - Her artışın entegrasyonu dikkatle planlanmalıdır
- Kullanım Alanları:
  - Büyük ve karmaşık yazılımlar
  - Müşteri geri bildirimine önem verilen projeler



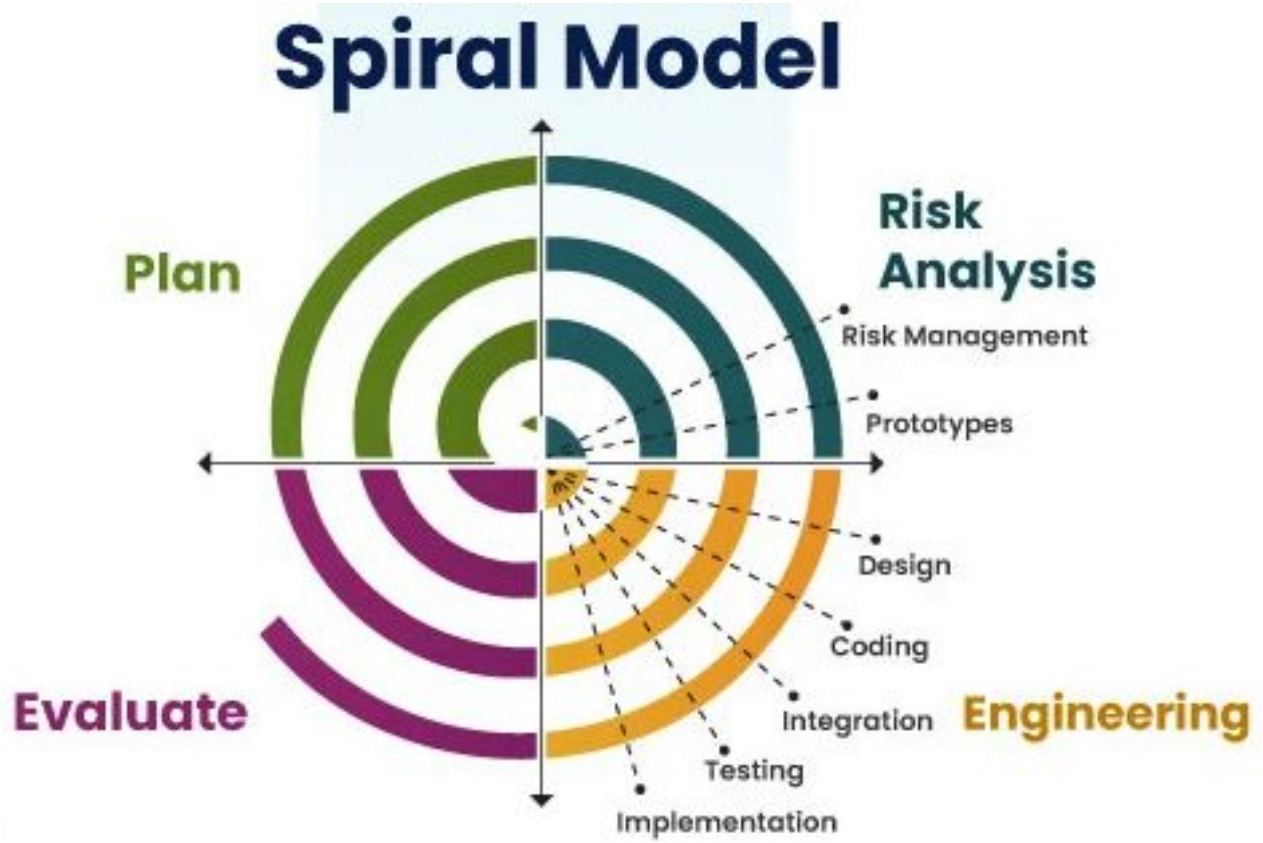
Artımlı Model (Incremental Model)



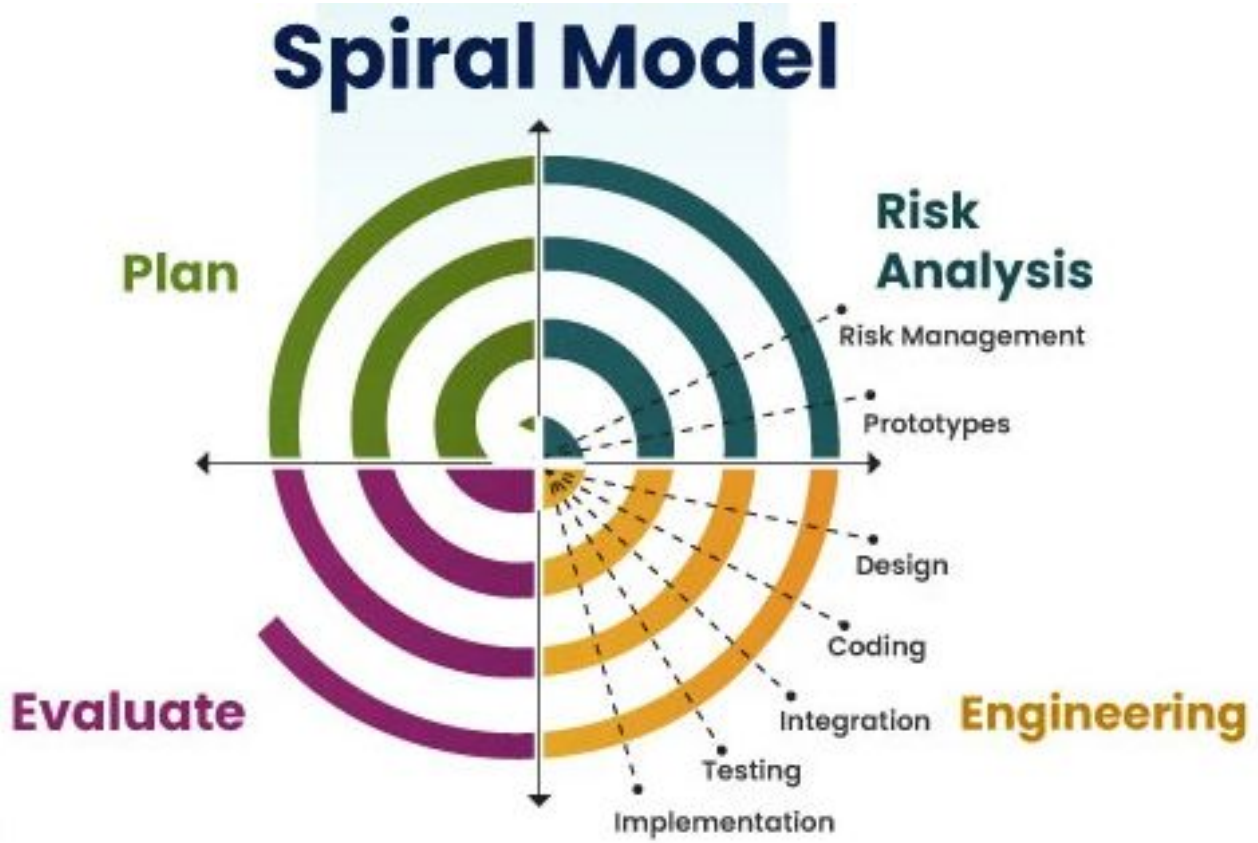
- Bir yazılımın temel işlevlerini içeren ilk sürümü müşteriye sunmak avantajlı mıdır? Neden?
- Artımlı model, risk yönetimi açısından ne gibi kolaylıklar sağlar?

Spiral Model

- Risk odaklı bir modeldir; her döngüde planlama, risk analizi, geliştirme ve değerlendirme yapılır.
- Aşamalar:
  - Planlama
  - Risk Analizi
  - Geliştirme ve Test
  - Müşteri Değerlendirmesi
- Avantajlar:
  - Riskler erken belirlenir ve yönetilir
  - Büyük ve karmaşık projeler için uygundur
  - Müşteri geri bildirimi her döngüde alınır
- Dezavantajlar:
  - Karmaşık ve maliyetlidir
  - Küçük projeler için aşırı ağırdır
- Kullanım Alanları:
  - Büyük ölçekli, yüksek riskli projeler
  - Uzun vadeli yazılım projeleri



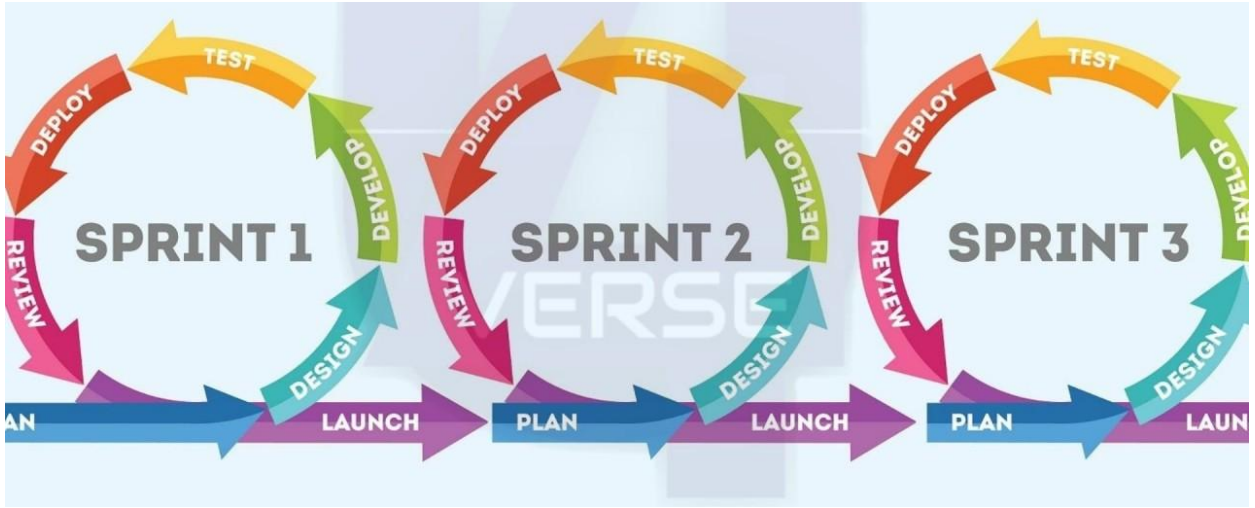
Spiral Model



- Spiral modelin her döngüsünde risk analizi yapılır. Sizce bu yaklaşım **proje maliyetini ve zamanını** nasıl etkiler?
- Büyük ve karmaşık bir proje düşünün: Spiral model neden bu tip projeler için daha uygundur?

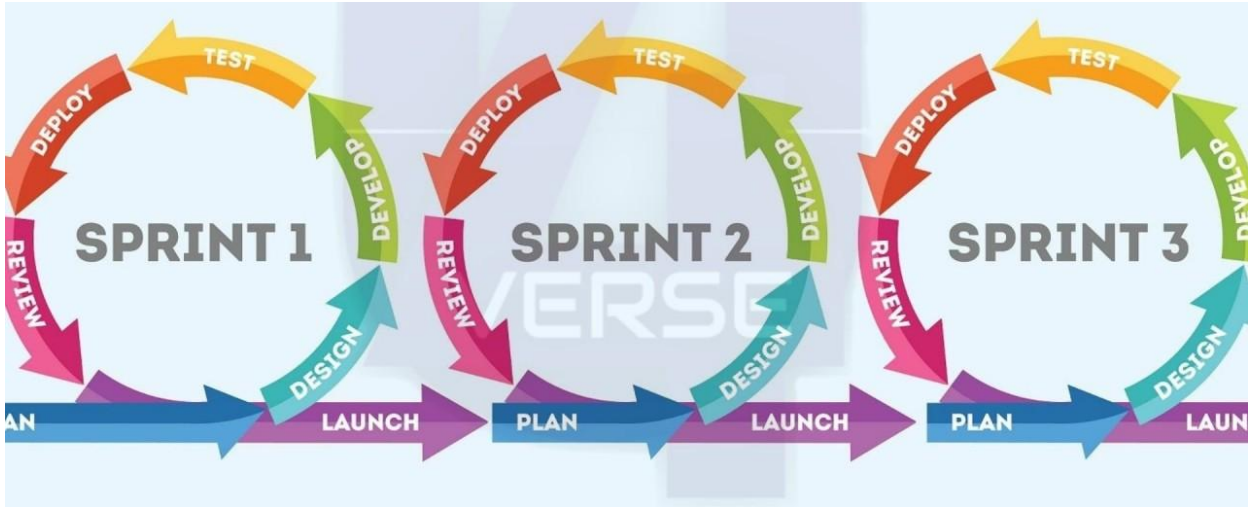
Çevik (Agile) Model

- Esnek, müşteri odaklı ve iteratif bir yaklaşımdır. Yazılım kısa süreli sprint'lerle geliştirilir ve her sprint sonunda çalışan bir ürün sunulur.



- Temel Prensipler:
  - Müşteri ile sürekli iletişim
  - Çalışan yazılım önceliklidir
  - Değişim fırsat olarak görülür
  - Ekip işbirliği önemlidir
- Popüler Çevik Metodolojiler:
  - Scrum: Sprint, Product Backlog, Daily Stand-up
  - Kanban: Görselleştirilmiş iş akışı ve sürekli iyileştirme
  - Müşteri geri bildirimi her döngüde alınır
- Avantajlar:
  - Esnek ve hızlı adaptasyon
  - Erken ve sürekli geri bildirim
  - Müşteri memnuniyeti yüksek
- Dezavantajlar:
  - Dokümantasyon genellikle azdır
  - Büyük ekiplerde yönetim zorlaşabilir

Çevik (Agile) Model



- Sprint sonunda çalışan bir yazılım sunmak müşteriye ve geliştirme ekibine **hangi avantajları** sağlar?
- Agile yöntemlerde dokümantasyon genellikle sınırlıdır. Sizce bu durum projeyi **hangi açılardan etkileyebilir?**

Genel Karşılaştırma

- Eğer gereksinimlerin çok net olmadığı bir proje elinize geçse hangi modeli seçerdiniz ve neden?
- Büyük bir projede, değişen kullanıcı gereksinimlerini yönetmek için hangi modeli tercih edersiniz?
- Tüm bu modeller arasında sizin ideal süreç modeli nasıl olurdu? Neden?

Gereksinim Mühendisliği

- **Gereksinim Mühendisliği**, bir yazılım sisteminin kullanıcılar, müşteriler ve diğer paydaşlar için ne yapması gerektiğini ve hangi kısıtlamalar altında çalışması gerektiğini belirleme, belgeleme ve yönetme sürecidir. Yazılım geliştirme yaşam döngüsünün ilk ve en kritik aşamalarından biridir; başarısız projelerin büyük bir kısmı eksik, yanlış veya sürekli değişen gereksinimlerden kaynaklanır.
- **Gereksinim (Requirement):** Bir ürünün veya hizmetin belirli bir amaca ulaşması için sağlaması gereken bir koşul veya yetenek.
- **Paydaş (Stakeholder):** Yazılımın geliştirilmesinden, kullanılmasından veya sonucundan etkilenen herhangi bir kişi veya kuruluş. (Örn: Müşteri, Son Kullanıcı, Proje Yöneticisi, Geliştirici, Test Uzmanı).
- **Yazılım Gereksinim Belgesi (YGB/SRS - Software Requirements Specification):** Onaylanmış tüm gereksinimleri detaylı ve sistematik bir şekilde içeren resmi dokümandır.

Gereksinim Mühendisliği

- Örnek

ID	UC-001
Creater	B.A.
Created Date	1/1/2000
Updater	
Updated Date	
Actor	Guide
Description	Create Tour - Tur rehberi yeni bir tur planı oluşturur.
Preconditions	Tur rehberi uygulamaya giriş yapmış olmalıdır. Tur rehberi kullanıcısı "Onaylanmış" statüsünde olmalıdır.
Postcondtions	Tur oluşturulur ve tur rehberi 2 saniye sonra ana sayfaya yönlendirilir.
Scenario	1. Tur rehberi anasayfadaki "Tur Oluştur" butonuna basar. 2. Sistem tur rehberi kullanıcısını "Create Tour" sayfasına yönlendirir. 3. Tur rehberi kullanıcısı item1, item2, item3.... alanlarını doldurur. 4. Tur rehber "Yayına Al" butonuna basar. 5. Sistem seçimleri kontrol eder ve işlemi veritabanına kaydeder.
Exceptions	
Comments	

Gereksinim Türleri

- Gereksinimler, doğalarına göre iki ana kategoriye ayrılır:
- **İşlevsel Gereksinimler (Functional Requirements)**
  - Sistemin **ne yapması gerektiğini** tanımlar.
  - Kullanıcıların sistemden beklediği hizmetleri, görevleri ve özellikleri içerir.
  - *Örnekler:*
    - "Sistem, kullanıcıların kredi kartı ile ödeme yapmasına izin vermelidir."
    - "Kullanıcı, şifresini başarıyla sıfırlayabilmelidir."
    - "Sistem, tüm sipariş kayıtlarını veritabanında saklamalıdır."
- **İşlevsel Olmayan Gereksinimler (Non-Functional Requirements - NFR)**
  - Sistemin **nasıl çalışması gerektiğini** veya sistemin kalitesini tanımlar.
  - Performans, güvenlik, kullanılabilirlik, güvenilirlik, sürdürülebilirlik gibi kalite özelliklerini ve tasarım kısıtlarını kapsar.
  - *Örnekler:*
    - **Performans:** "Sayfa yükleme süresi 2 saniyeyi geçmemelidir."
    - **Güvenlik:** "Tüm kullanıcı şifreleri tek yönlü şifreleme ile saklanmalıdır."
    - **Kullanılabilirlik:** "Sistem, yaygın ekran okuyucularla uyumlu olmalıdır."
    - **Güvenilirlik:** "Sistem, %99.9 çalışma süresi (uptime) sağlamalıdır."
    - **Kısıtlama:** "Yazılım, Java 17 ve PostgreSQL veritabanı kullanılarak geliştirilmelidir."

Gereksinim Mühendisliği Süreci

Aşama	Açıklama	Ana Faaliyetler
1. Ortaya Çıkarma (Elicitation)	Paydaşlarla etkileşim kurarak sistemin amaçlarını ve gereksinimlerini keşfetme.	Görüşmeler, Beyin Fırtınası, Senaryo/Kullanım Hikâyeleri, Prototipleme.
2. Analiz ve Müzakere (Analysis & Negotiation)	Ortaya çıkarılan "ham" gereksinimleri inceleme, sınıflandırma, çatışmaları çözme ve detaylandırma.	Modelleme (Use Case Diyagramları), Gereksinimleri Önceliklendirme, Çatışma Çözümü.
3. Belirtim (Specification)	Analiz edilmiş gereksinimleri resmi, net ve sistematik bir belge (YGB/SRS) haline getirme.	Gereksinimleri doğal dilde veya biçimsel gösterimlerle dokümante etme.
4. Doğrulama/Onaylama (Validation)	Geliştirilen gereksinimlerin, paydaşların gerçek ihtiyaçlarını doğru ve eksiksiz olarak karşılayıp karşılamadığını kontrol etme.	Gereksinim İncelemesi (Review), Prototipleme ile Geri Bildirim Alma, Test Edile

Teknik	Açıklama	Kullanım Alanı
Görüşmeler (Interview)	Paydaşlarla bire bir veya grup halinde konuşarak gereksinimleri öğrenme.	En yaygın ve etkili yöntem. Bilgi sahibi paydaşlar için ideal.
Senaryolar / Kullanım Hikâyeleri	Kullanıcının sistemle nasıl etkileşimde bulunacağını adım adım tanımlama. (Örn: "Bir kullanıcı olarak, bir ürünü sepete ekleyebilmek istiyorum.")	Kullanıcı merkezli geliştirme, Çevik (Agile) projeler.
Gözlem (Etnografi)	Kullanıcıların mevcut sistemleri gerçek ortamlarında nasıl kullandığını izleme.	Yeni bir sistem geliştirilirken mevcut iş akışını anlama.
Prototipleme	Sistemin çalışan (ancak eksik) bir modelini oluşturarak erken geri bildirim alma.	Kullanılabilirlik ve kullanıcı arayüzü (UI) gereksinimlerini netleştirme.
Beyin Fırtınası (Brainstorming)	Bir grup paydaşın serbestçe fikir üretmesi.	Yeni ve yenilikçi çözümler aranan projeler.

Gereksinim Analizi ve Modelleme

- Analiz aşamasında, toplanan gereksinimler netleştirilir ve tutarsızlıklar giderilir. **UML (Birleşik Modelleme Dili)** diyagramları bu aşamada sıklıkla kullanılır:
- **Kullanım Senaryosu (Use Case) Diyagramları:** Sistem sınırlarını, aktörleri (kullanıcılar/diğer sistemler) ve aktörlerin sistemle gerçekleştirdiği işlevleri (kullanım senaryoları) gösterir. **Temel İşlevsel gereksinimlerin** haritasını çıkarır.
- **Sınıf Diyagramları:** Sistemdeki veri ve nesnelerin yapısını ve ilişkilerini (analiz objeleri) tanımlar.
- **Durum Diyagramları:** Bir nesnenin farklı durumlarını ve bu durumlar arasındaki geçişleri gösterir.

UML (Unified Modeling Language - Birleşik Modelleme Dili)

- Yazılım Gereksinim Analizi (Software Requirements Analysis) sürecinde **UML (Unified Modeling Language - Birleşik Modelleme Dili)**, gereksinimleri **görselleştirmek, analiz etmek, kesinleştirmek ve standartlaştırmak** için kullanılan endüstri standardı bir araçtır.
- UML'nin temel amacı, karmaşık yazılım sistemlerinin yapısını ve davranışını grafiksel gösterimlerle belgeleyerek geliştirme ekibi, müşteri ve diğer paydaşlar arasında **ortak ve net bir anlayış** sağlamaktır. Doğal dille yazılmış gereksinimlerin yol açabileceği belirsizliği (muğlaklığı) ortadan kaldırmada kritik rol oynar.
- **Gereksinim Analizi İçin UML'nin Rolü :** UML, gereksinim analizinde özellikle iki ana gereksinim türünü (İşlevsel ve İşlevsel Olmayan) modellemek için kullanılır:
  - **1. İşlevsel Gereksinimlerin Modellenmesi (Sistemin Ne Yapacağı) :** Bu aşamada, sistemin kullanıcılar için ne tür hizmetler sunacağı ve bu hizmetlerin nasıl işleyeceği görselleştirilir.
  - **2. Yapısal Gereksinimlerin Modellenmesi (Sistemin Kimler ve Hangi Verilerle Çalışacağı) :** Sistemin statik yapısını ve kullanacağı temel veri yapısını tanımlar.

UML (Unified Modeling Language - Birleşik Modelleme Dili)

- ## 1. İşlevsel Gereksinimlerin Modellenmesi (Sistemin Ne Yapacağı)

UML Diyagramı	Amacı ve Gereksinim Analizindeki Rolü
Kullanım Senaryosu (Use Case) Diyagramı	**İşlevsel Gereksinimlerin en önemli aracıdır.** Kullanıcıların (Aktörler) sistemle gerçekleştirdiği tüm ana işlevleri (Kullanım Senaryoları) ve sistemin sınırlarını tanımlar. Kullanıcı bakış açısıyla sistemin kapsamını çizer.
Aktivite Diyagramı (Activity Diagram)	Bir iş sürecinin veya bir kullanım senaryosunun içindeki **adım adım akışı (iş akışını)** görselleştirir. Koşulları, döngüleri ve paralel adımları göstererek gereksinimlerin mantıksal sırasını netleştirir.
Sıra Diyagramı (Sequence Diagram)	Belirli bir görev (örneğin "Giriş Yapma") sırasında nesnelerin **hangi sırayla ve birbirleriyle nasıl etkileşime girdiğini** (mesaj gönderip aldığını) zaman bazlı olarak gösterir. Bir kullanım senaryosunun detaylı davranışını analiz etmek için kullanılır.

- ## 2. Yapısal Gereksinimlerin Modellenmesi (Sistemin Kimler ve Hangi Verilerle Çalışacağı)

UML Diyagramı	Amacı ve Gereksinim Analizindeki Rolü
Sınıf Diyagramı (Class Diagram)	Sistemin temel **veri yapılarını (sınıflarını)**, bu sınıfların sahip olduğu **özellikleri (nitelikleri)** ve sınıflar arasındaki **ilişkileri (bire bir, bire çok vb.)** tanımlar. Sistemde tutulması gereken bilgileri ve bunların bağlantılarını netleştirir.

UML Kullanımının Faydaları

- Yazılım gereksinim analizinde UML kullanmak, projelere önemli avantajlar sağlar:
  - **Belirsizliğin Giderilmesi:** Doğal dildeki ("Sistem hızlı olmalıdır") belirsiz ve yoruma açık ifadeler yerine, görsel ve standart bir gösterim sunarak herkesin aynı şeyi anlamasını sağlar.
  - **İletişim Kolaylığı:** Teknik ekipler (geliştiriciler) ile teknik olmayan paydaşlar (müşteriler) arasında ortak bir dil oluşturur.
  - **Eksiklik ve Çatışma Tespiti:** Diyagramlar, mantık hatalarını, eksik adımları veya birbiriyle çelişen gereksinimleri henüz kodlama başlamadan görsel olarak ortaya çıkarmayı kolaylaştırır.
  - **İzlenebilirlik:** Gereksinimler (Use Case Diyagramları) ile sistemin tasarımı (Sınıf ve Sıra Diyagramları) arasında doğrudan bir bağlantı kurulmasını sağlayarak projenin ilerleyen aşamalarını (Tasarım ve Test) destekler.
  - **Test Edilebilirlik:** Gereksinimlerin akışını ve davranışını netleştirdiği için, test senaryolarının (özellikle Aktivite ve Sıra diyagramlarından) kolayca türetilmesine olanak tanır.

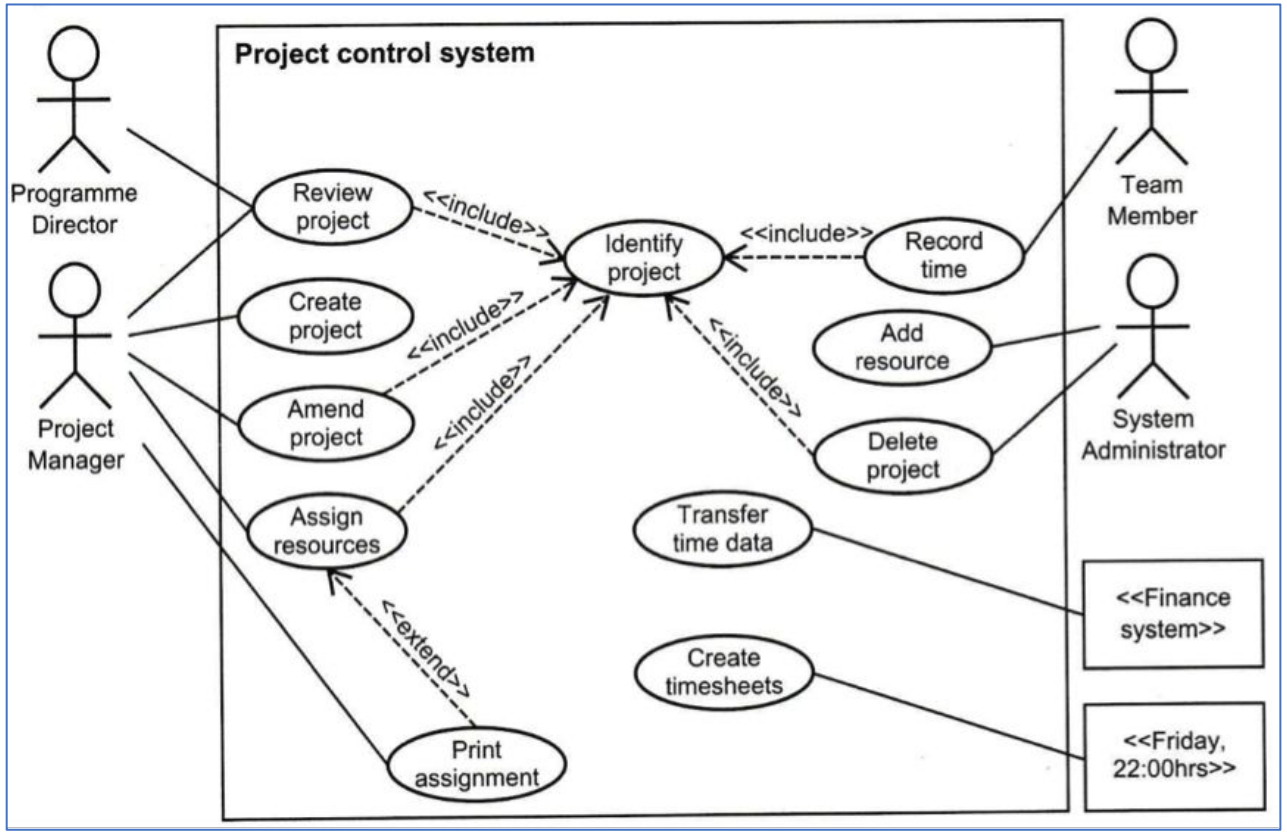
Kullanım Senaryosu Diyagramı (UML Use Case Diagram)

- Bu diyagramın temel amacı, geliştirilecek veya analiz edilen bir sistemin **kapsamını, sınırlarını ve temel işlevselliğini** kullanıcıların bakış açısıyla (aktörler aracılığıyla) göstermektir. Sistemin ne yapması gerektiğini yüksek seviyede ve kolay anlaşılır bir şekilde özetler.
- Bir Kullanım Senaryosu Diyagramı dört temel bileşenden oluşur:
  - **A. Sistem (System Boundary)**
    - **Gösterim:** Dikdörtgen bir kutu.
    - **Açıklama:** Analiz edilen sistemin sınırlarını temsil eder. Dikdörtgenin içindeki her şey sistemin bir parçasıdır (geliştirilecek yazılım), dışındaki her şey ise sistemle etkileşen dış dünya (aktörler) veya diğer sistemlerdir.
  - **B. Aktör (Actor)**
    - **Gösterim:** Çöp Adam (Stick Figure) veya başka bir sistem ise bir kutu.
    - **Açıklama:** Sistemle etkileşimde bulunan, bir değeri olan bir görevi başlatan veya sistemden fayda sağlayan dışsal varlık (kullanıcı, dış sistem, cihaz veya zamanlayıcı) rolüdür.
    - Aktörler bir kişi değil, bir **roldür**. (Örn: "Müşteri", "Yönetici", "Veritabanı Sistemi").
  - **C. Kullanım Senaryosu (Use Case)**
    - **Gösterim:** Sistemin sınırları içindeki elips şekli.
    - **Açıklama:** Sistemin bir aktör için gerçekleştirdiği, ölçülebilir bir değere sahip olan tek bir işlevi veya görevi temsil eder. İşlevsel bir gereksinimi yüksek seviyede tanımlar.
    - **Adlandırma:** Genellikle bir fiil ve bir isimden oluşur (Örn: "Ürün Ara", "Ödeme Yap", "Rapor Oluştur").

Kullanım Senaryosu Diyagramı (UML Use Case Diagram)

- **D. İlişkiler (Relationships)**
  - Bileşenler arasındaki bağlantıları gösterir. Temel olarak üç ana ilişki türü vardır:

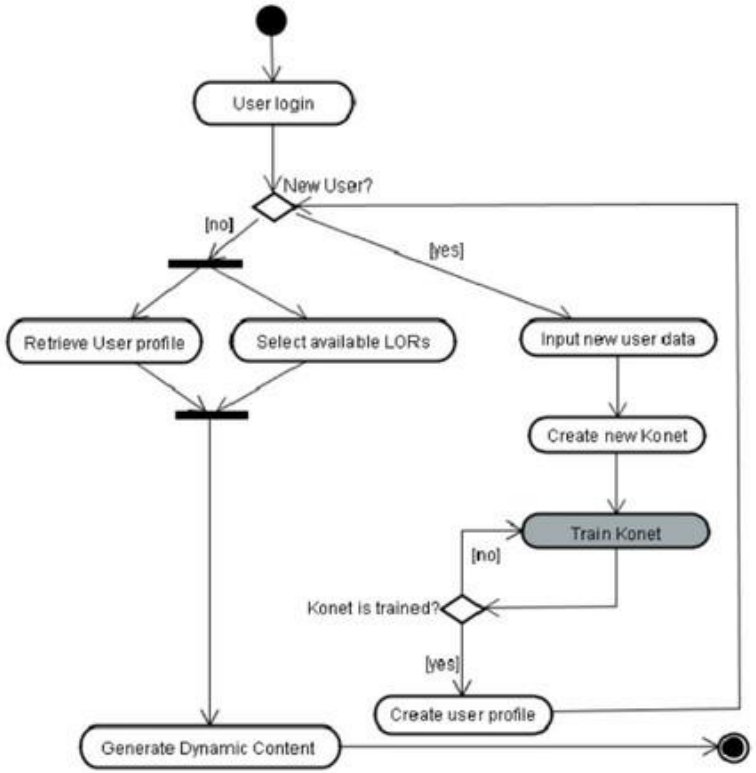
İlişki Türü	Gösterim	Anlamı	Kullanım Alanı
İletişim/İlişki (Association)	Düz çizgi	Bir aktörün belirli bir kullanım senaryosunu başlattığını veya onunla etkileşimde bulunduğunu gösterir.	Aktörler ve Kullanım Senaryoları arasında.
İçerme (≪include≫)	Kesikli ok ve ≪include≫ etiketi	Bir kullanım senaryosunun, başka bir kullanım senaryosunun **zorunlu olarak çalışmasını gerektirdiğini** gösterir. (Tekrarlanan mantık modülleri için kullanılır.)	"Ödeme Yap" ≪include≫ "Kullanıcıyı Doğrula"
Genişletme (≪extend≫)	Kesikli ok ve ≪extend≫ etiketi	Bir kullanım senaryosunun, **belirli bir koşul altında** isteğe bağlı veya alternatif bir işlevi genişlettiğini gösterir.	"Ödeme Yap" ≪extend≫ "Kupon Kullan" (Kupon varsa uygulanır)



Aktivite Diyagramı – Sıra Diyagramı

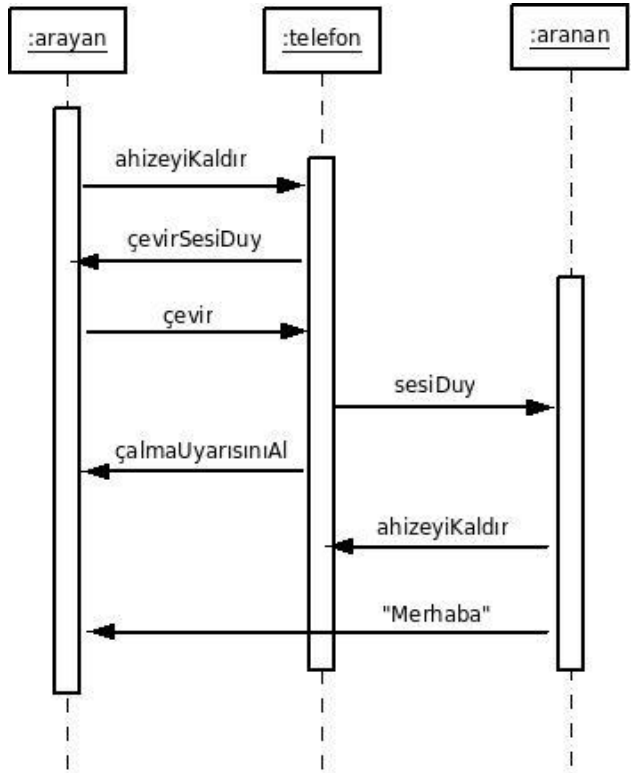
Aktivite Diyagramı

Bir iş sürecinin veya bir kullanım senaryosunun içindeki **adım adım akışı (iş akışını)** görselleştirir. Koşulları, döngüleri ve paralel adımları göstererek gereksinimlerin mantıksal sırasını netleştirir.



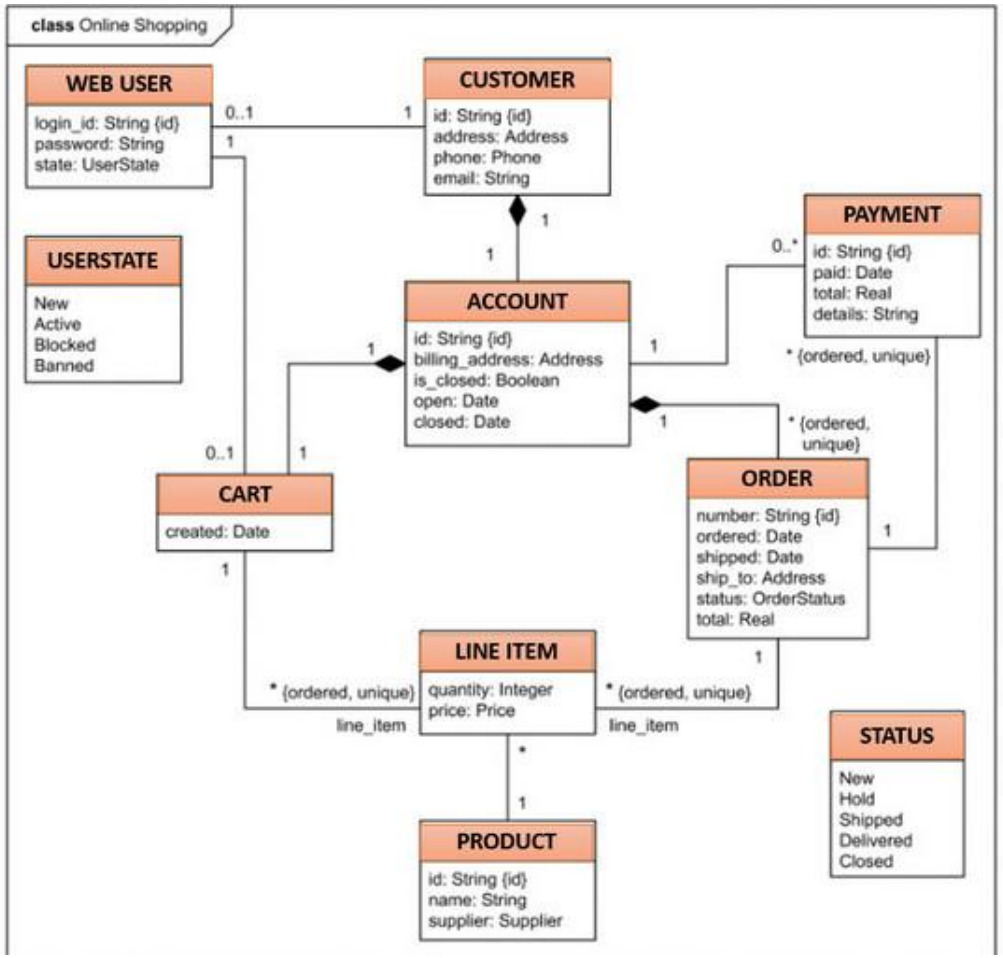
Sıra Diyagramı

Belirli bir görev (örneğin "Giriş Yapma") sırasında nesnelerin hangi sırayla ve birbirleriyle nasıl etkileşime girdiğini (mesaj gönderip aldığını) zaman bazlı olarak gösterir. Bir kullanım senaryosunun detaylı davranışını analiz etmek için kullanılır.



Sınıf Diyagramı (Class Diagram)

- Sistemin temel **veri yapılarını (sınıflarını)**, bu sınıfların sahip olduğu **özellikleri (nitelikleri)** ve sınıflar arasındaki **ilişkileri (bire bir, bire çok vb.)** tanımlar. Sistemde tutulması gereken bilgileri ve bunların bağlantılarını netleştirir.



Yazılım Gereksinim Belgesi (YGB/SRS)

- Yazılımın **ne yapacağını** (nasıl yapılacağını değil) tanımlayan kritik bir referans belgesidir.
- **İyi Bir Gereksinimin Özellikleri**
  - Bir gereksinim aşağıdaki özellikleri taşımalıdır:
  - **Net ve Belirsiz Olmayan (Unambiguous):** Tek bir şekilde yorumlanabilmelidir.
  - **Tam (Complete):** Gereken tüm bilgileri içermelidir.
  - **Tutarlı (Consistent):** Diğer gereksinimlerle veya sistem kısıtlarıyla çelişmemelidir.
  - **Doğrulanabilir/Test Edilebilir (Verifiable):** Bir test veya inceleme ile uygulanıp uygulanmadığı kontrol edilebilmelidir.
  - **İzlenebilir (Traceable):** Kökeni (kim istedi) ve uygulamadaki karşılığı takip edilebilir olmalıdır.
  - **Gereklilik (Necessary):** Gerçekten bir paydaş ihtiyacına cevap vermelidir.

Bölüm No	Bölüm Adı	İçerik
1.0	**Giriş**	Amacı, Proje Kapsamı, Belgeyi Okuyucu Kitle, Tanımlar ve Kısaltmalar.
2.0	**Genel Açıklama**	Ürünün Perspektifi (diğer sistemlerle ilişkisi), Ürün Fonksiyonlarının Özeti, Kullanıcı Karakteristikleri, Genel Kısıtlar ve Varsayımlar/Bağımlılıklar.
3.0	**Özel Gereksinimler**	Sistemin temel gereksinimlerinin detaylı listesi.
3.1	İşlevsel Gereksinimler	Her bir ana işlevin (Use Case) ayrıntılı açıklaması, girdileri, çıktıları ve işlem adımları.
3.2	Harici Arayüz Gereksinimleri	Kullanıcı Arayüzü (UI), Donanım, Yazılım ve İletişim Arayüzleri.
3.3	İşlevsel Olmayan Gereksinimler	Performans, Güvenlik, Güvenilirlik, Sürdürülebilirlik, Taşınabilirlik vb. kısıtlamaların sayısal ve net ifadeleri.
Ekler	Kullanım Senaryosu Diyagramları, Veri Sözlüğü, Teknik Modeller vb.	

Gereksinim Yönetimi

- Gereksinim Yönetimi, yazılımın yaşam döngüsü boyunca gereksinimlerin değişmesini, izlenmesini ve kontrol edilmesini sağlayan disiplindir.
- **Gereksinim Önceliklendirme**
  - Tüm gereksinimler aynı öneme sahip değildir. Önceliklendirme, kaynakların doğru kullanılması için önemlidir. Yaygın teknikler:
  - **MoSCoW:**
    - **M**ust have (Mutlaka olmalı)
    - **S**hould have (Olsa iyi olur)
    - **C**ould have (Olabilir)
    - **W**on't have (Şimdilik olmayacak)
  - **Kano Modeli:** Müşteri memnuniyetine göre sınıflandırma (Temel, Performans, Heyecan Verici).
- **İzlenebilirlik (Traceability)**
  - Gereksinimlerin, kaynağı (paydaş ihtiyacı), tasarımı, uygulaması (kod parçası) ve test senaryoları arasındaki bağlantıların kurulmasıdır.
  - **Gereksinim İzlenebilirlik Matrisi:** Hangi gereksinimin hangi test senaryosu, hangi tasarım öğesi ve hangi üst düzey iş hedefiyle ilişkili olduğunu gösterir.

Gereksinim Yönetimi

- **Değişiklik Yönetimi (Change Management)**
  - Gereksinimlerin kaçınılmaz olarak değişeceği varsayılır. Bu değişikliklerin kontrollü yapılması gerekir.
  - **Değişiklik Talebi (Change Request - CR):** Bir paydaş, gereksinim değişikliği talep eder.
  - **Etki Analizi:** Değişikliğin maliyeti, zaman çizelgesi ve diğer gereksinimler/tasarım üzerindeki etkisi değerlendirilir.
  - **Onay/Red:** Bir **Değişiklik Kontrol Kurulu (Change Control Board - CCB)** tarafından değişiklik talebi onaylanır veya reddedilir.
  - **Uygulama ve Güncelleme:** Değişiklik onaylanırsa, SRS ve ilgili tüm belgeler (tasarım, test senaryoları) güncellenir.

İSTANBUL MEDENİYET ÜNİVERSİTESİ

KATILIMINIZ İÇİN TEŞEKKÜR EDERİM.

YOKLAMA İÇİN İMZANIZI ATMAYI UNUTMAYINIZ.
CLASSROOM ÜZERİNDEN SINIFA DAHİL OLMAYI UNUTMAYINIZ

Sınıf Kodu : pua2tnoe

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