Yazılım Mühendisliği

Yazılım Mühendisliği → İlk fikirden yazılımın ölümüne kadar her yönüyle ilgilenen mühendislik dalı

İyi Yazılım

- + Fonksiyonellik
- + Performans
- + Bakım kolaylığı
- + Güvenilirlik
- + Kullanılabilirlik
- + Kabul Edilebilirlik
- + Verimlilik

ETİK

- Gizlilik
- Yeterlilik
- Fikri Mülkiyet Hakları
- Bilgisayarın kötüye kullanımı

KALİTE

Soyut, yoruma açık

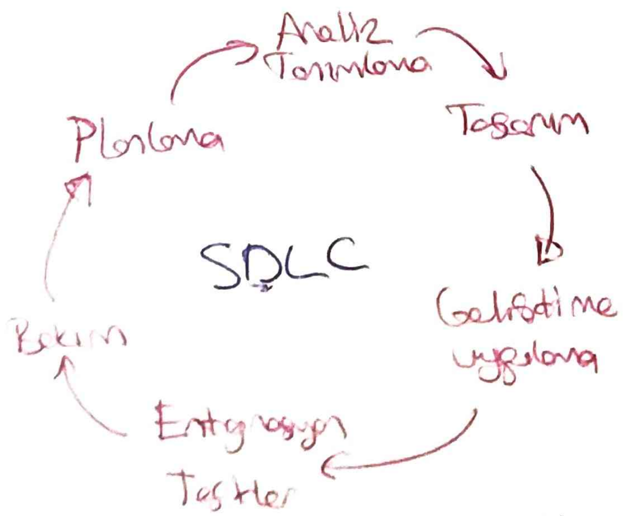
* Gereksinimleri karşılama başarısıdır

YAZILIM YAŞAM DÖNGÜSÜ

Fikirden → ölümüne kadar olan zaman

Yazılım Mühendisliği Destek Süreçleri

- Proje yönetimi
- Kalite yönetimi
- Geçerleme Doğrulama
- Formal ve Teknik Gözden geçirme
- Yazılım konfigürasyon Yönetimi
- Yeniden kullanılabilirlik Yönetimi
- Yazılım ölçümü
- Risk Yönetimi



Software Development Life Cycle

KALİTE GEREKSİNİMLERİ

Doğruluk ↔ **Doğruluk**

- gereksinimleri karşılama

Doğruluk
Fiziksel modelin Mantıksal modele uygunluğu

Güvenilirlik
- gerek duyulan işlevleri, fonksiyonları ne hassasiyet ile yerine getirecek

Verimlilik
- İşlevlerin, fonksiyonların için kullanılması gereken kaynakların, zamanın, kodun miktarı

Tamlık
Gereken işlevlerin tam olarak tanımlanma derecesi

Kesinlik
Tasarımda satır sayısının küçüklüğü

Bütünlük
- Yazılımın değişmeden etkilenmeden kullanılabilmesi

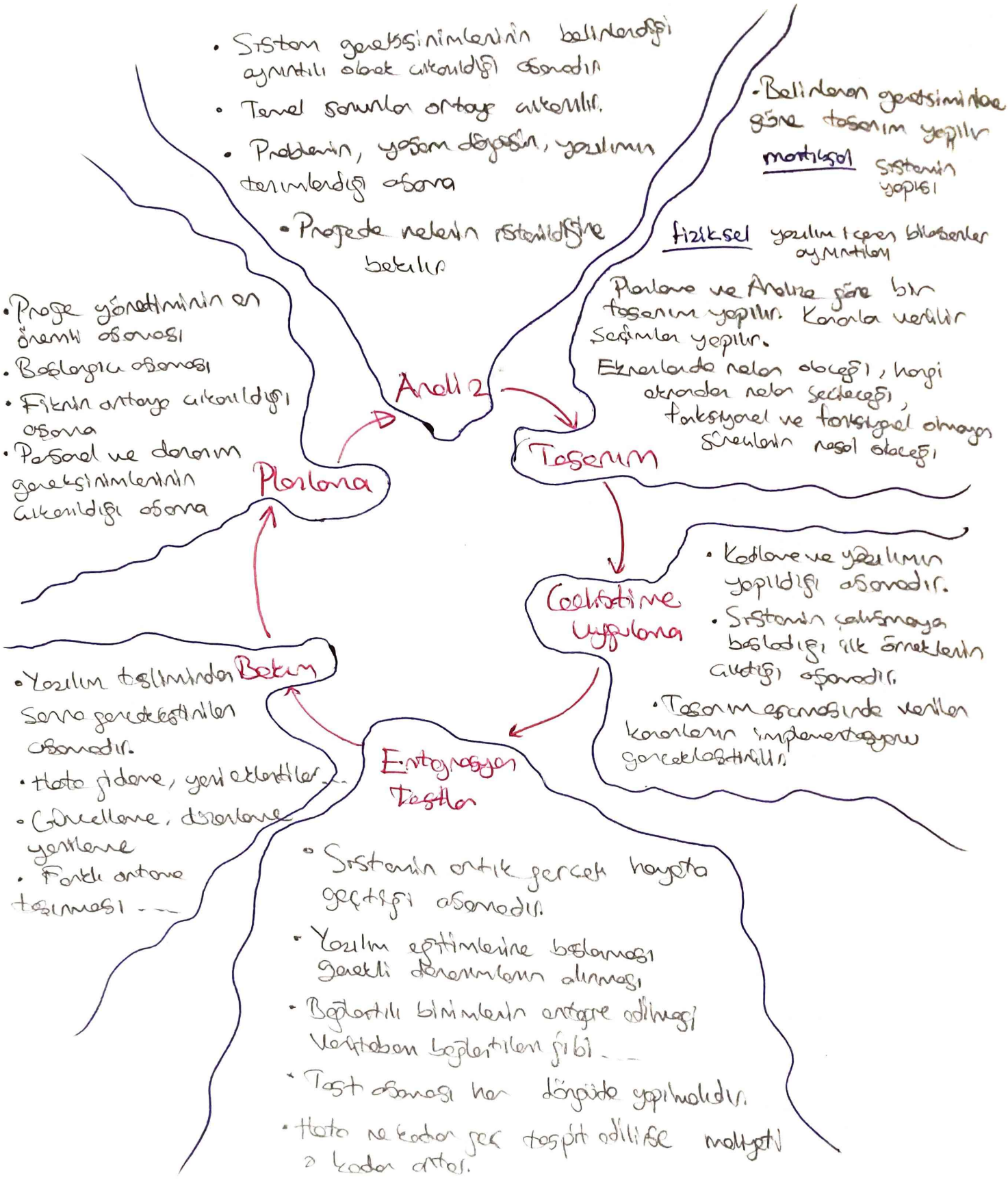
Tutarlılık
Proje sürecinde tasarım ve belgeleme süreçlerinde uygunluğu

Kullanılabilirlik
- Programın öğrenim kolaylığı
girdi-çıktı işlemleri için gereken çaba
hatasız verimli kullanma

Bakım kolaylığı
Hata bulma, onarma için gereken çaba

Esneklik
Programda değişiklik yapabilme kolaylığı

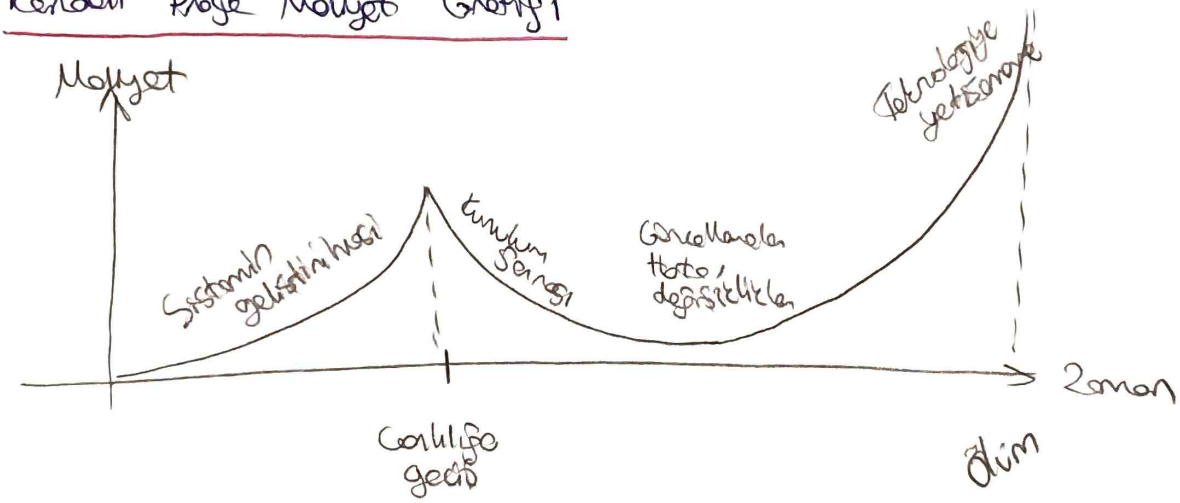
Denetlenebilirlik
Standartlara uygunluğun kontrol edilebilmesi


Analiz
• Sistem gereksinimlerinin belirlendiği ayrıntılı olarak çıkarıldığı aşamadır
• Temel sorunlar ortaya çıkarılır.
• Problemin, yaşam döngüsü, yazılımın tanımlandığı aşama
• Projede nelerin istendiğine bakılır
Tasarım
• Belirlenen gereksinimlere göre tasarım yapılır
mantıksal Sistemin yapısı
fiziksel yazılım içeren bileşenler ayrıntılar
Planlama ve Analize göre bir tasarım yapılır. Kararlar verilir Seçimler yapılır.
Ekranlarda nelar olacağı, hangi ekranlar nelar seçileceği, fonksiyonel ve fonksiyonel olmayan süreçlerin nasıl olacağı
Planlama
• Proje yönetiminin en önemli aşamasıdır
• Başlangıç aşamasıdır
• Fikrin ortaya çıkarıldığı aşama
• Personel ve donanım gereksinimlerinin çıkarıldığı aşama
Geliştirme Uygulama
• Kodlama ve yazılımın yapıldığı aşamadır.
• Sistemin çalışmaya başladığı ilk örneklerin çıktığı aşamadır.
• Tasarım aşamasında verilen kararların implementasyonu gerçekleştirilir.
Entegrasyon Testler
• Sistemin artık gerçek hayata geçtiği aşamadır.
• Yazılım eğitimlerine başlanması Gerekli dönanımların alınması
• Bağlantılı birimlerin entegre edilmesi Veritabanı bağlantılarının gibi...
• Test aşaması her döngüde yapılmalıdır.
• Hata ne kadar geç tespit edilirse maliyeti o kadar artar.
Bakım
• Yazılım teslimatından sonra gerçekleştirilen aşamadır.
• Hata giderme, yeni eklentiler...
• Güncelleme, düzeltme yapılarına
• Farklı ortama taşınması...

Kendall'ın 7 adımı

Problem, Fırsatlar ve hedeflerin belirlenmesi
İnsan sevkiyatı enformasyon ihtiyaçlarının belirlenmesi
Sistem ihtiyaçlarının belirlenmesi
TÜM bu isteklerin birleşiminden bir sistem, yazılım tasarlanması
Yazılım geliştirilmesi ve dökümantasyon
Sistem testleri ve bakım adımlarının belirlenmesi
Sistemin gerçeğe alınması ve değerlendirilmesi

Kendall Proje Maliyet Grafiği



Yazılım Geliştirme Süreç Modelleri

Geleneksel (Ad hoc)
Klasik, doğrusal akış (Waterfall)
V model
Evrimsel Süreç
↳ Artırımlı (İnkremental) Model
↳ Helezon (Spiral) Model
Çevik Modeller (Agile)
↳ Scrum
↳ XP

Geleneksel (Ad hoc)

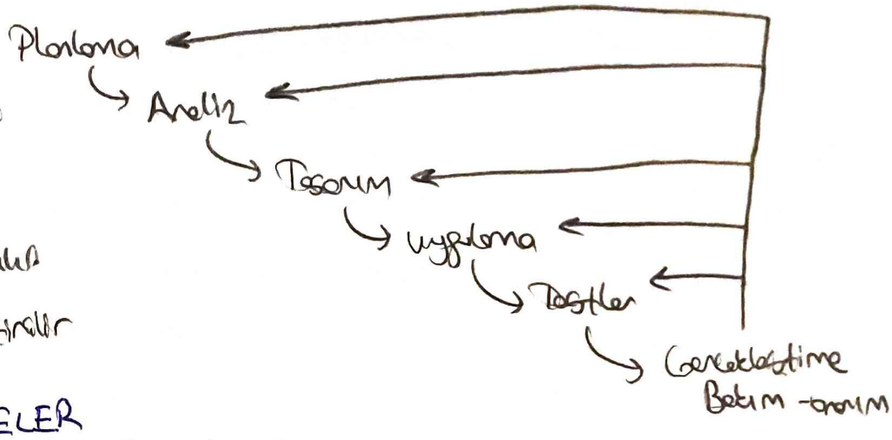
Kaotik, düzensiz ve plansızdır
- Tek kişilik üretim ortamı
- İzlenebilirliği ve bakımı zor
- Basit programlama
- Belirtilmiş

Barok Model

Belgelemeyi ayrı bir süreç olarak ele alır
Aşamalar arası geri dönüş yok

WATERFALL (ŞELALE) MODELİ

- İlk yayınlanan model
- Bir sonraki aşama, bir önceki aşama bitmeden başlamaz
- Dokümantasyon odaklı
- Problemler son aşamaya bırakılır

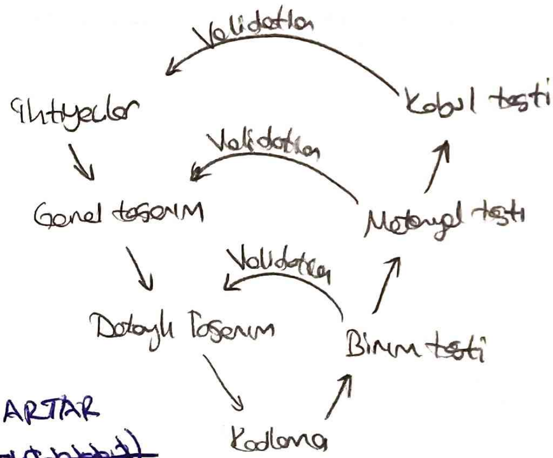


SDLC en az bir kez gerçekleştirilir

* İYİ TANIMLI PROJELER
* AZ ZAMAN GEREKTİREN PROJELER
* GERİ DÖNÜŞE İZİN VERMEZ

V MODELİ

- Üst seviyede daha genel bakıştan tasarım yapılır
- Sol taraf üretim, sağ taraf sınama



* İYİ TANIMLI PROJELER
* KULLANICININ PROJEYE KATKISI ARTAR
~~* KÜÇÜK (<100.000) * ORTA (<500.000)~~

EVRİMSEL SÜREÇ MODELLERİ

- Aşamaların tekrarlanarak yazılım ürünü sürekli geliştirilmektedir
- Küçük parçalar halinde yazılım ortaya çıkmakta, sonradan versiyonları oluşmakta

— HER VERSİYON İÇİN DOKÜMANTASYON MALİYETİ
— SÜREKLİ DEĞİŞİM MALİYETLİ
— ÖZEL ARAÇ VE TEKNİK GEREKLİ, İNSAN YETENEĞİ İHTİYACI
* KÜÇÜK (< 100.000) ORTA (<500.000)

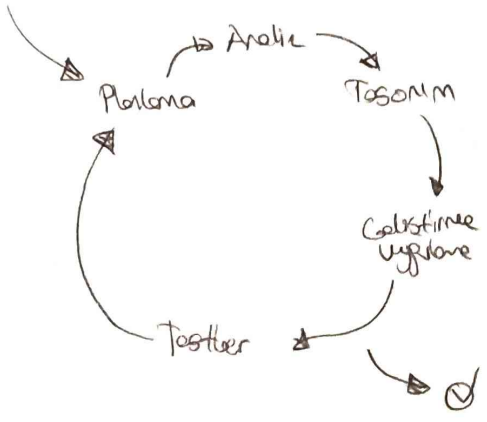
Artımsel (Incremental) Model

Spiral Model

RUP

ARTIMSAL (INCREMENTAL) MODEL

- Sistemi fonksiyonel birimlere ayırıp teslimatları gerçekleştirmek
- Modelde bir döngü var



Analiz → Tasarım → Kod → Test —✓

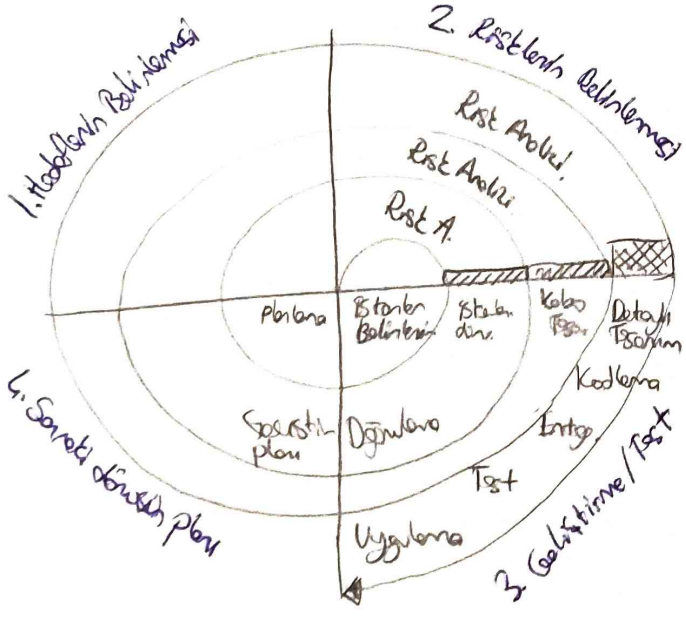
Analiz → Tasarım → Kod → Test —✓

Analiz → Tasarım → Kod → Test —✓

- Risk Az

- Müşteri tüm sistemin bitmesini beklemez
- Müşteri prototipleri kullanarak deneyim kazanır. Gelişimin devamı için ihtiyaçları söyler.
- Müşteri daha çok risk içindedir

SPIRAL MODEL



- Süreci tekrar tekrar eder.
- Her turda projenin farklara kaydetmesini hedefler.
  - Hedeflerin Belirlenmesi
  - Risklerin Belirlenmesi
  - Geliştirme ve Testler
  - Sonraki dönüşün planı
- Ürünler en sonu kullanıcıya sunar

RISK ANALİZİ OLGUSU BELİRTİLDİ

ÇEVİK (AGILE) METODLAR

⊕ İNSAN ODAKLI YAKLAŞIM

XP METODU

4-10 kişilik takım
İki haftalık dilimler halinde
Sık gerçekleşmeli
* MÜŞTERİ İLE GÜÇLÜ İLİŞKİ GEREKTİRİR

⊕ İLETİŞİM
⊕ GERİ BİLDİRİM
⊕ BASİTLEŞTİRME
⊕ CESARET

ÇEVİK MODELİN TEMEL İLKELERİ

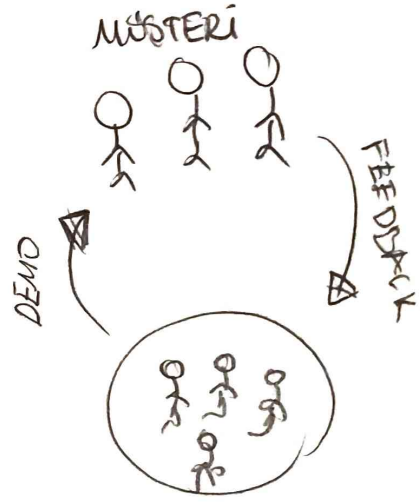
- Hızlı, devamlı, kullanışlı yazılım üretmek
- İnsan ilişkilerini destek etmek
- Çalışan yazılım
- Geç değişiklikleri memnuniyetle karşılamak
- Yüz yüze görüşmek
- Kendi kendini organize takım
- Basitlik

Bireyler ve ilişkileri / Süreç ve araca tercih etmek
Çalışan bir yazılımı / Detaylı dökümantasyona tercih etmek
Müşteri ile işbirliğini / Anlaşma görüşmelerine tercih etmek
Değişiklikleri / Sıkı bir plana tercih etmek

Çok önemli — Az önemli

MANİFESTO

Hızlı Devamlı Kullanışlı yazılım geliştirmek
Değişiklik taleplerinin kolayca kabulü
Sık sık çalışan yazılım teslimi
Geliştiriciler ile alan uzmanları arasında günlük iletişim
Projeye motive bireyler topluluğu
Yüz yüze görüşme ve iletişim
Çalışan yazılım parçaları, gelişim ölçüsüdür.
Sürdürülebilir bir hız sağlamak
Teknik mükemmelliğe katılım ve iyi tasarım
Basitlik önemli
Kendi kendine organize ekip
Ekipler kısa sürede toplantılar ile çalışmayı gözden geçirir



⊕ TAHMİN EDİLEMEYEN DEĞİŞİMLER VARSA/OLACAKSA ÖNCEDEN YOL HARİTASI ÇİZMEK ZOR İSE
⊕ ANALİZ/TEST SÜRELERİNİN UZUNLUĞUNUN BİLİNMEMESİ
⊕ SAĞLAM İLETİŞİM GEREKLİ İSE

AGİLE ÇEVİK KULLANIM

GEVİK MODELİN PROBLEMLERİ

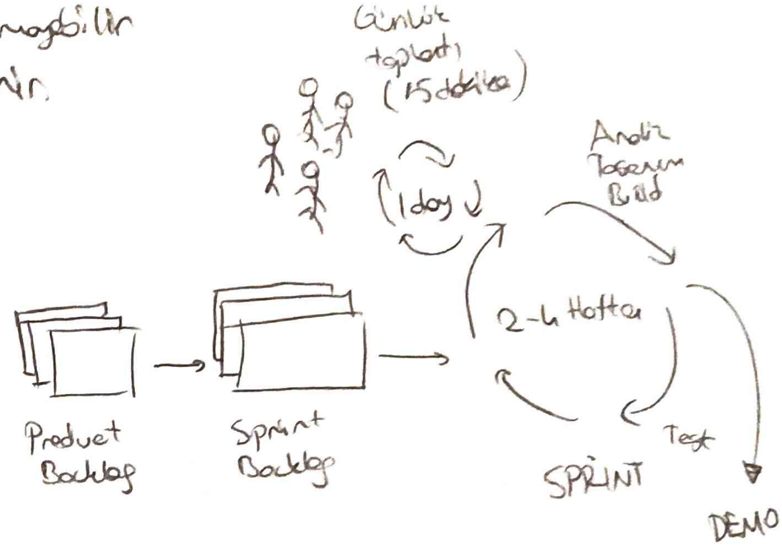
Müşterinin ilgisini kılmak zor

Takımlar yoğun konuşmaya hazır olmayabilir

Sadeliği korumak fazladan iş gerektirir.

SCRUM

- Geliştirme paketlere bölünmüş
- İş "sprintler" halinde gerçekleşir
- Gereksinimlerin "backlog"u oluşur
- Toplantılar kısa, bazen başkansız.
- Müşteriye DEMO'lar sunulur.



BACKLOG

Son kullanıcıdan gelen gereksinimler

Ne yapacağız?

Herkese açık, herkes değiştirebilir

USER STORY

SPRINT

Belirli süresi var

Sonunda çıktı olması gerek

Toplantılar içerir

Şeffaflık + Denetim + Uyum + SCRUM Takımı + Product owner + Sprint Backlog + Product Backlog + SCRUM Master

= DEMO

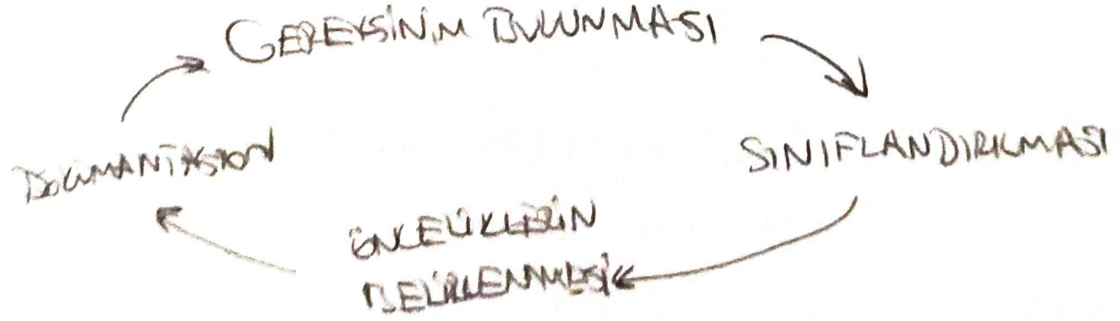
GEREKSİNİM MÜHENDİSİ

+ Kullanıcı gereksinimleri → Kullanıcıya sunacağı servisler, kullanıcı sınırları
+ Sistem gereksinimleri → yazılım gereksinimleri

İşe başlamadan önce gereksinimler → kullanıcı gereksinimleri ve ilk prototiplemeye geçiş → sistem gereksinimlerinin belirlenmeye başlama

– Paydaşlar genelde gereksinim belirlerken zorlanır, kendi terimlerini kullanırlar

Farklı paydaşlar farklı söyler



Gereksinim Sepesifikasyon

Kullanıcı ve sistem gereksinimlerinin dökümantasyonu.

- Doğal Dil ile yazılması
- yapısal doğal dil kullanımı
- grafik gösterimi
- Matematiksel gösterimler

İSTERLERİN BELİRLENMESİ – GEREKSİNİM ANALİZİ

Yazılımın gereksinimleri ne kadar iyi tespit edilirse uygulaması ve hatasızlığı o kadar iyi olur.

- Müşteri ne istiyor ayrıntılı belirlenir

SENARYO ve USE CASE (KULLANIM DURUMLARI) müşterilerle geliştiriciler arasında iletişimi kolaylaştırır.

Senaryo sınıfını tanımlayan Soyutlama (abstraction)'dır.

Sistem kullanımına örnek sistem ve kullanıcı arasındaki etkileşimler

Gereksinim tanımı

- Sistemin sahip olması gereken bir özellik
- İsteneni tarafından kısıtlanması gereken kısıtlamalar

* MÜŞTERİ – GELİŞTİRİCİ – KULLANICI İLETİŞİMİ ÇOK ÖNEMLİ
* DOĞRUDAN KULLANICI TARAFINDAN GÖRÜLMEYEN UNSURLAR İSTER (GEREKSİNİM) DEĞİLDİR

⊕ Aktörlerin belirlenmesi → Farklı kullanıcı türleri belirlenir

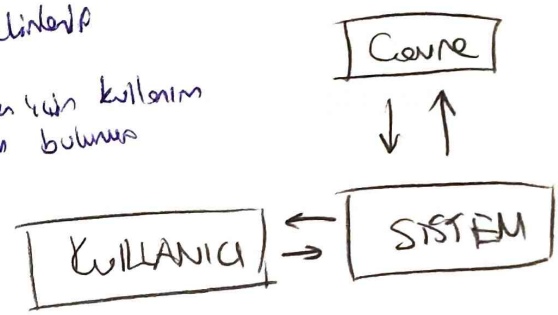
⊕ Senaryoların belirlenmesi → Tipik istekler için kullanım senaryoları bulunur

⊕ Kullanım durumlarının belirlenmesi

⊕ " " iyileştirilmesi

⊕ " " arasındaki ilişkiler

⊕ Fonksiyonel olmayan isterlerin belir.



MÜŞTERİ İLE TAKIM ARASINDAKİ İLİŞKİYE YÖNELİK

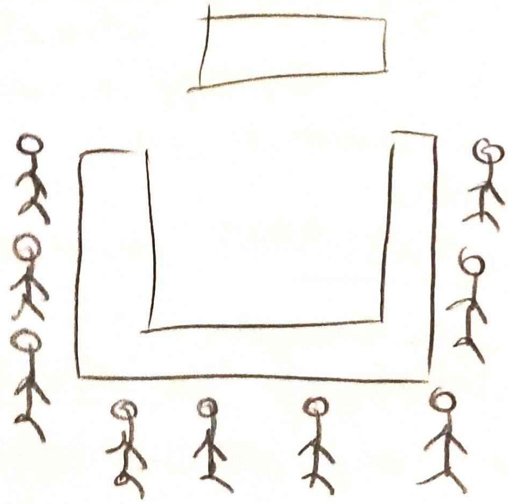
Joint Application Design (JAD)

Knowledge Analysis of Task (KAT)

Usability Test

JAD (Joint Application Design)

Fikir birliği ile kabul edilmiş sistem gereksinimlerini ortaya çıkarmak için düzenlenen toplantılardır.



Katılımcılar :

- ⊕ Sponsor
- ⊕ Toplantı yöneticisi
- ⊕ Dökümantör (Yazıcı)
- ⊕ Business Bilirkişisi
- ⊕ Teknik Bilirkişi
- ⊕ Gözlemciler

KAD (Knowledge Analysis of Task)

Geliştiricilerin, gözlemler sonrası bilgi edinmeleridir.

Usability Testing (Kullanılabilirlik Testleri)

• Modelin çeşitli metodlar ile sınanmasıdır

Test Türleri
- Süreç içi
- Süreç Sonu

Test Yaklaşımları
- Tasarım Rehberleri Temelleri
- Uzman Temelli
- Kullanıcı Temelli
- Model Temelli

GEREKSİNİM KAVRAMLARI

Fonksiyonel gereksinimler

Sistemin sunması gereken servislerin, girdilere vereceği cevapların sistemin nasıl davranması gerektiğini anlatan ifadeler

Fonksiyonel Olmayan gereksinimler

Sistemin sunduğu servisler ve fonksiyonlar üzerindeki kısıtlamalardır.
- Zaman kısıtlamaları
- Geliştirme kısıtlamaları
- Standartlaştırma kısıtları

Gereksinimlerin Doğrulanması
- + Doğruluk → Doğru fonksiyon tanımlama
- + Tamlık → Tüm olası koşulları içeriyor mu?
- + Tutarlılık → Çelişen gereksinimler var mı?
- + Açıklık → Gereksinimler açık ve basit ifadeli
- + Gerçekçilik → Gereksinimler yapılabilir mi?
- + İzlenebilirlik

Greenfield Eng. → Mevcut bilinen sistem yok, sistem baştan geliştirilecek

Re Eng. → Mevcut sistemi yeni teknoloji ile birleştirmek

İnterface Eng. → Mevcut sistemin yeni ortama taşınması

Fonksiyonel Gereksinimler

Sistemin ne yapacağını tanımlar

Fonksiyonel Olmayan Gereksinimler

Sistem kısıtları, teknik ortam, dışarıdan kullanıcı odaklı, güvenlik, güvenilirlik kalite güvencesini belirleyen gereksinimlerdir.

⊕ Tepki süresi
⊕ Değişiklik Derecesi
⊕ Kısıtlar

⊕ Hız
⊕ Büyüklük
⊕ Kullanım kolaylığı
⊕ Taşınabilirlik
⊕ Taşınabilirlik

AKTÖRLERİN BELİRLENMESİ

Sistemle etkileşimde olan KULLANICILAR yada DİĞER SİSTEMLER

- Her aktörün sistemle etkileşimi belirlenmeli

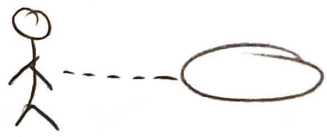
SENARYOLARIN BELİRLENMESİ

- Bir aktörün sistemle olan etkileşimini, adım adım yaptıklarını anlatımıdır.
- Tek bir özelliğin tek bir aktör açısından tanımlanmasıdır.

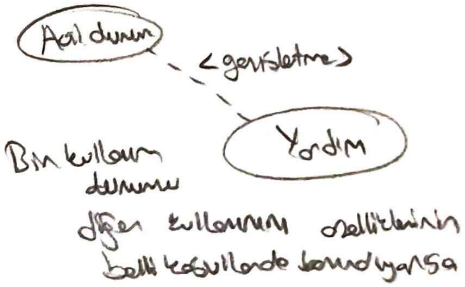
KULLANIM DURUMLARININ BELİRLENMESİ

Bir senaryo bir kullanım durumu örneğidir.
USE CASE tüm senaryoların birleşimidir.

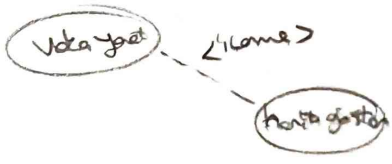
İLETİŞİM



GENİŞLETME



İÇERME



FONKSİYONEL OLMAYAN GEREKSİNİM ANALİZİ

Kullanıcı arayüzü – insan faktörü
Dökümantasyon
Donanım faktörleri
Performans özellikleri
Yasal konular
Kalite kontrolü
Sistem değişiklikleri
Fiziksel Çevre Koşulları
Güvenlik kontrolü
Kaynak kontrolü

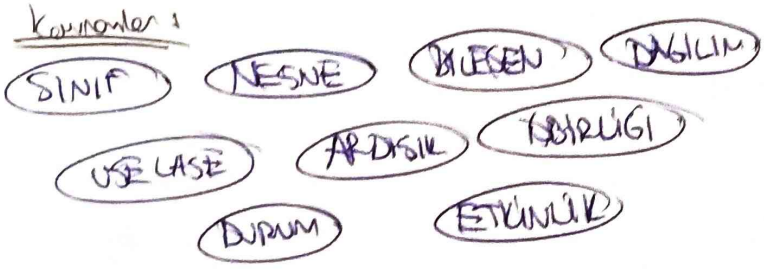
UML (UNIFIED MODELING LANGUAGE)

⊕ Çok karmaşık programların tasarımından gelişmiştir
⊕ Yazılım tasarımı aynı anda düşünülmüştür
⊕ Tek proje birden çok yazılımcı ile yapılacaksa

DİYAGRAM ÇİZME VE İŞLEMSEL MODELLEME DİLİ

YAZILIMIN GENİŞ ANALİZİ TASARIMI YAPILIR
BÜYÜK RESİM İÇİN ÇOK İYİ

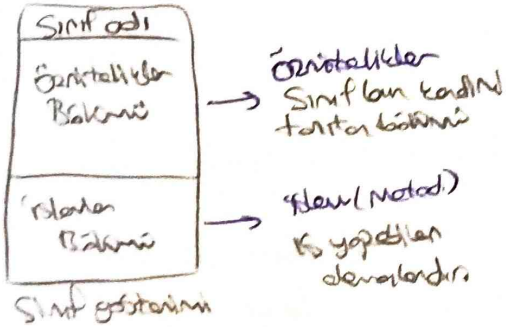
Kavramlar:

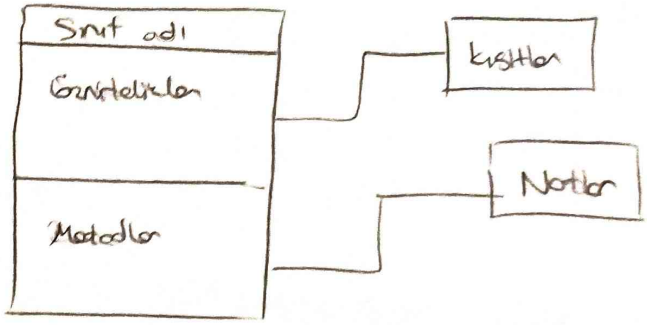


Kullanım Bakışı (User View)
Yapısal Bakış (Structural View)
Davranış Bakışı (Behavior View)
Gerçekleme Bakışı (Implementation View)
Ortam bakışı (Environment View)

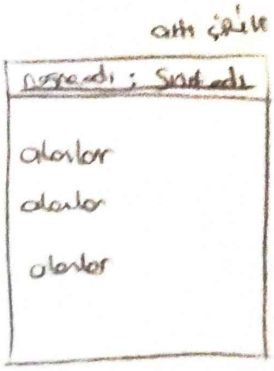
SINIF DİYAGRAMLARI

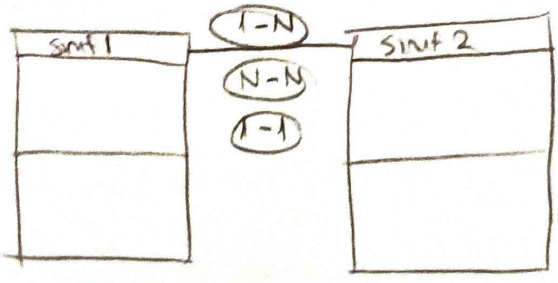
Kavramsal Model
• Eşik kullanılan Diyagram





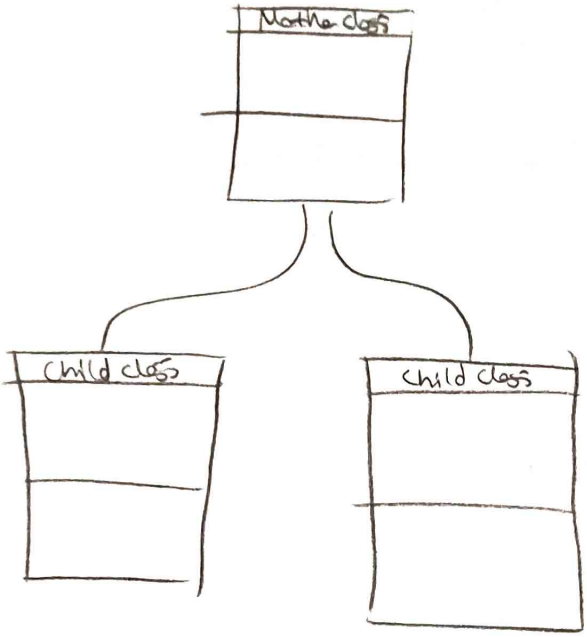
PUBLIC (+)
PROTECTED (#)
PACKAGE (~)
PRIVATE (-)



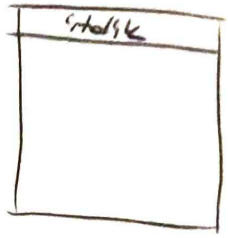


Sınıflar arasındaki ilişkiler çizgi çizerek ve üzerine bilgileri yazarak belirtilir.
Kısıtlar yazılabilir, kodlamada yardımcı

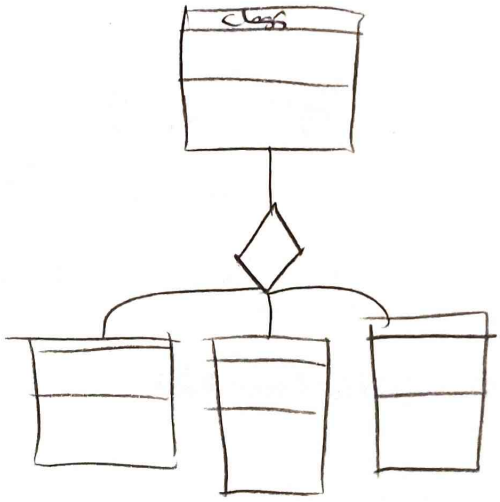
KALITIM



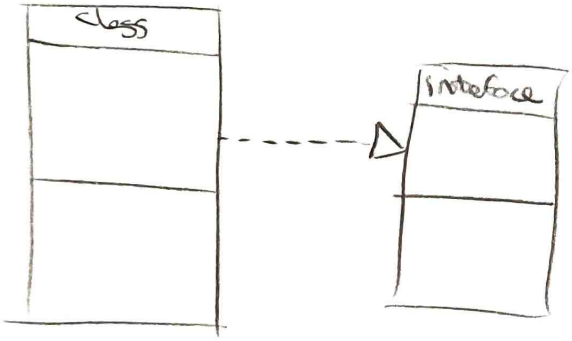
ABSTRACT SINIF



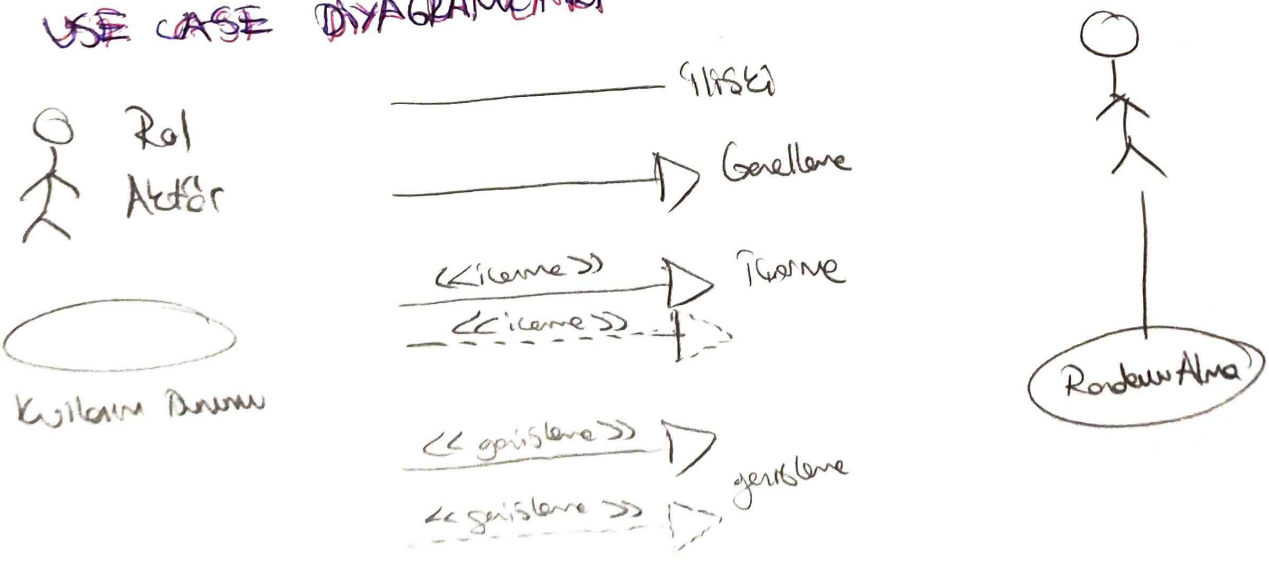
AGGREGATIONS



INTERFACE

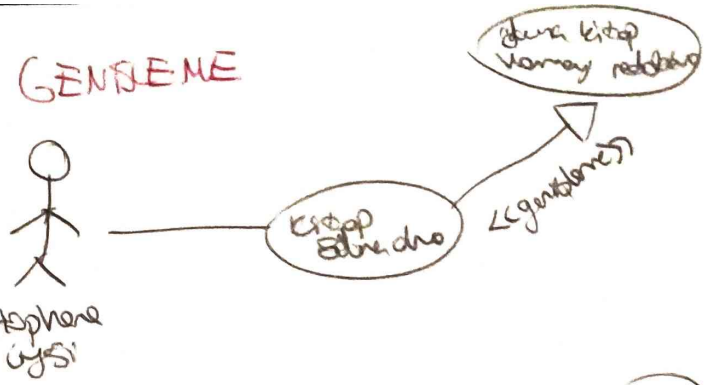


USE CASE DIYAGRAMLARI

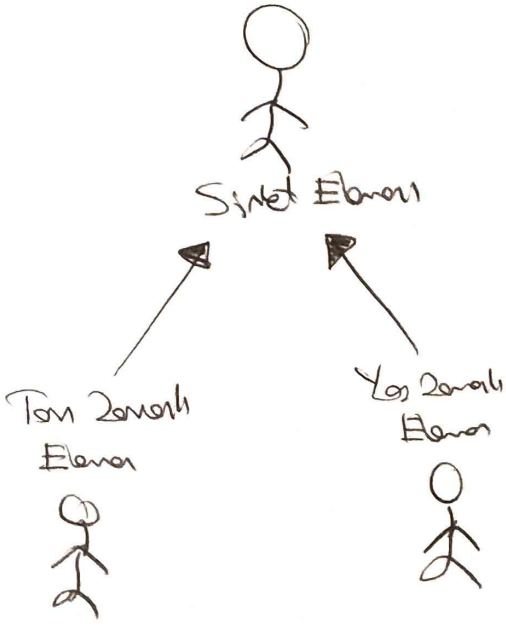


İLİŞKİ

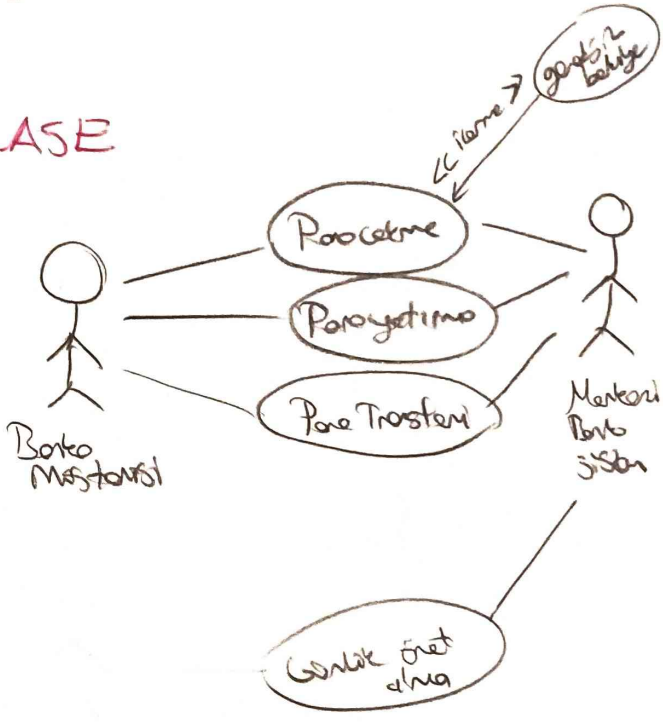
GENİŞLEME



GENELLEME



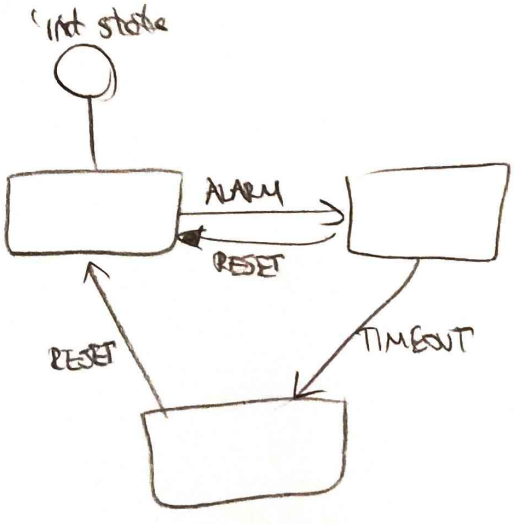
USE CASE



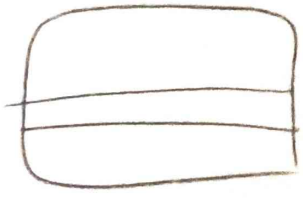
STATE DİYAGRAMLARI

Bir nesnenin belirli durumlarını ve sonuçta geçtiği durumları gösteren modeldir

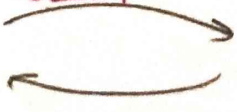
Durum Makinesi
Durum (State)
olay (Event)
Aksiyon (Action)
Geçiş (Transition)
Ön-şart
İlk durum (initial state)
Son durum (Final state)



STATE



TRANSITION
GEÇİŞ



INITIAL STATE

FINAL STATE