

## Giriş

Jenerik Ürünler: Satış, herkese açık ürünler.

## Yazılım Süreci Faaliyetleri

Dört Temel Aktivite vardır;

Specification: yazılımin özellikleri ve kısıtlamaların getirildiği yer

Development: yazılımin tasarılandığı ve programlandığı.

Software Validation: Mysteriye test etme

Evolution: yazılımin daha da geliştirilmesi

---

Heterojenlik: yazılımın tüm ortamlarda çalışması gerekmekte  
(Android, iOS...)

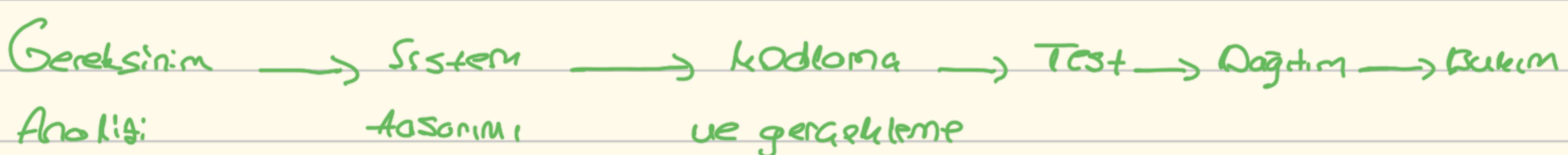
Entegre Servisler: Bir API aracılığıyla diğer hizmetler tarafından erişilebilen hizmetlerdir.

Bağımsız Servisler: Basitçe bir taraficıdan erişilen ve bağımsız çalışan servislerdir.

# Yazılım Süreç Modelleri

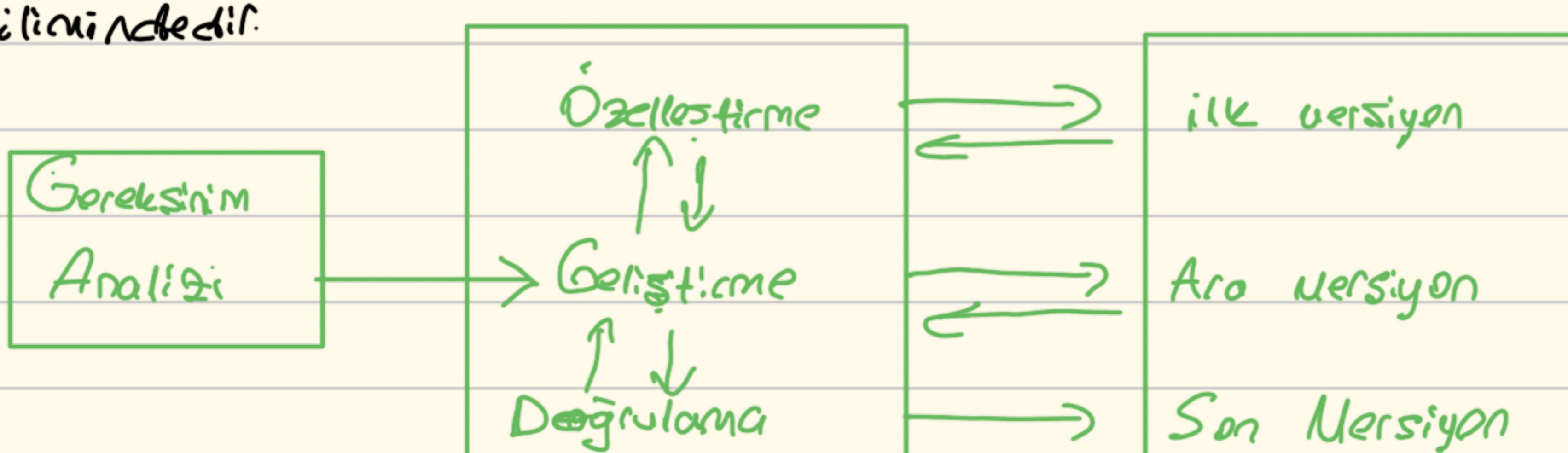
## 1) Selale Modeli:

- \* Plan odaklı modeldir, Spesifikasiyon ve geliştirmenin aynı ve farklı aşamaları vardır.
- \* Dezavantaj: süreç başladıkten sonra süreci değiştirmek zordur. Bir sonraki aşamaya geçmeden önceki aşamaların bitmesi olması gereklidir.
- \* Müşteri gereksinimlerine yanıt vermek zorlaşır.
- \* Çok büyük projelerde kullanılır.



## 2) Artımlı Geliştirme

- \* Değişen müşteriler gereksinimlerine hizmet etmek daha kolaydır.
- \* Yazılımin testimi daha kolay ve hızlıdır.
- \* Süreç gürünür değildir.
- \* Sistem yapısı genel istekler / özellikler eklenerek bozulma eğilimindedir.



### 3) Entegrasyon ve konfigürasyon Modeli:

- \* Sistemin entegre edildiği: yazılımın yeniden kullanımına dayalıdır.
- \* yeniden kullanımın öğeleri kullanımının gerekliliklerine uygunluk  
İN: yapılabilirlik.
- \* geliştirilenin gözlem miktarını azaltma bölgeleri maaşlı ve  
riskleri: azaltma konusunda avantajlıdır, yazılımın hizli teslimini de  
saglar.
- \* Geneksimlerden ötürü nemeli kagınlamasıdır, kullanım  
hizyogorunu koruyamaz.
- \* Yeniden kullanılabilir bileşenlerin yeni sürümleri: ORU kullanım  
muruşsun elinde olmadığından kontrol kısıtlıdır.

### Tasarım Faaliyetleri

**Mimar Tasarımı:** Sistemin genel yapısı, ana bileşenleri, bunların  
ilişkileri ve nasıl doğrudan belirlenir.

**Arayüz Tasarımı:** Bileşenler arası arayüz tasarılanır.

**Bilesen Tasarımı:** Her sistem bileşeninin nasıl çalışacağı.

**Merkaba Tasarımı:**

### Test Aşamaları:

1) Geliştirme veya Bileşen Testi:

2) Sistem Testi:

3) Kullanıcı Testi:

## Gevik Geliştirme

- \* Program Spesifikasyonu; tasarım ve uygulama iç içedir.
- \* Değerlendirme için yeni versiyon sık sık testim edilir.
- \* Minimum dokümantasyon; yazılım kodu açık olmaktadır.

## Release vs Version

- \* Release bir yazılımin hafka赞叹 doğrudır.
- \* Versiyon yazılımin herhangi bir bileşenini veya varyasyonunu ifade eder

## Plan Odaklı VS Gevik

- |   |   |
|---|---|
| Küreseldeki ilkelerin önceden planlandığı                 | * Spesifikasyon, tasarım, uygulama iç içedir.                                 |
| * Selale modeli şart değildir artımlı geliştirme de olur. | * İlerleme müzakeresi ile gerginleşir.<br>* Amacı genel giderken azaltmakdır. |
| * Bileşenler, etkinlikler içindedir                       | <u>Principler:</u>  |

\* Mütter: katılımlı

\* Artımlı Testimat

\* Sıkıcılar Değil insanlar

\* Değişikliği önemse

\* Sadeliğ: koruyun

## Gevik Yöntemlerin Başarılı Olduğu Yerler

- 1) Product Development
- 2) Custom System Development → müşterinin ürün geliştirme sürecine dahil olduğu

## Gevik Geliştirme Teknikleri:

### 1) Asırı Programlama (Extreme Programming);

\* Yükseltilmiş geliştirmeye asırı bir yaklaşım getirmektedir.

- Giinde birkaç kez yeni versiyon
- Artımlar müşterilere 2 haftada 2 testim edilir
- Her yapı oncaく testlerden geçerse başarılı kabul edilir.

\* Temel XP pratikleri;

- 1) Gereksinim için kullanıcı hikayeleri
- 2) Yeniden Düşünen → Sürekli kod iyileştirmeleri (Refactoring)
- 3) Test Önceliği: Geliştirme
- 4) Esli Programlama

### Esli Programlama;

\* Kullanıcının girişlerini kolide calışıp kod çalıştırmasını sağlar.

\* Kod satırılığını artırır ve bilgiyi takip genelinde yayar.

\* Refactoringe teşvik eder.

\* Aynı bulgusuya boşta otururlar, genel riski de azaltırlar

## 2) Scrum (En yaygın yöntem.) (Geçikle framework)

- \* Geçik pratiklerden ziyade yinelenebilir geliştirmeye odaklanır.
- \* Scrumda üç aşama vardır.
  - 1) Genel hedef ve yazılım mimarisı.
  - 2) Her döngünün sistemin artışı, geliştirilmesi **sprint döngüsü**
    - ↓
    - Sabit uzunluk (2-4 hafta)
  - 3) Proje kapanışı
- \* framework sağlama odaklıdır ve eşli programlamayı zorunlu tutmaz.
- \* Sprint döngüsünün başlangıcı **product backlog** (listesi) dir.
  - ↓
  - Kullanıcı mikyası veya basit bir talimat olabilir.
- \* Segim aşaması, tüm proje ekibini ve müşteriyi içerişir.
- \* Product Owner, döngü boyunca en önemli öğelerin hangileri olduğunu tanımlamak için backlog listesindeki öğelerin öncekile venir.
- \* Yapılacak öğeler belirlendikten sonra önceki sprintlerden takmin yürüterek yeni sprintte backlogların ne kadarının kırılabilirliğini takmin ederler.
- \* Böylece, bir sprint boşluk ve ekip utmın nerde kalışacağına vendi karar verir.

### Sprint Döngüsü

- \* Takım, müşteriden ve organizasyondan izole edilir. Tüm iletişim, "Scrum Master" aracılığıyla kanalize edilir.
- \* Scrum Master, ekibi dış etkenlerden korur.

\* Sprint sonunda, yapılan görüşmeler gözden geçirilir ve bir sonraki sprint daha sonra başlar.

2 Daily Scrum → Ekip her gün aynı saatte 15 dakikalık kısa toplantı yapar.  
Distributed Scrum

\* Scrum takım üyesinin her gün bir araya geldiği şekilde tasarlanmıştır ancak artık geliştiriciler dijitalin farklı yerlerinden bir araya gelmemektedir.

\* offshore geliştirme için ürün sahibi artık dağıtık olabilecek geliştirme ekibinden farklı bir ülkedendir.

## Kanban

\* Toyota tarafından ortaya çıkarılmış, David Anderson tarafından yayılım geliştirmeye uyarlanmıştır

\* Yalın bir metodoloji: yalın, hizmet sunumu, kendini yönetme

\* Hizmet Sunumuna odaklanmıştır.

\* Ekip üyelerine kendilerini yönetme özgünlüğü verir.

1) Kanban Board ile iş akışını görselleştirme

Backlog	Selected	Analysis	Development	Testing	Done

\* Kanban boardin üç adımlı iş akışı vardır;

1) to do → yapılacaklar

2) doing (in progress) → yapılmıyor

3) done → Tamamlandı

2) Herhangi bir zamanda Devam Eden iş Miktarının (WIP) Sınırlanılması:

Geliştinicilerin, başka bir görevin başlamadan önce mevcut görevlerini bitirmesi, önceliğin

3) iş Akışını Yönetme;

4) Süreç Politikaları Açık Hale Getirilir

Örneğin üzerinde çalışılan işe segilir ve "done" tanımlaması için kullanılan kriter yazılıf.

5) Sıkılık iyileştirmeye Dikkatlenmek;

6) Süreç Değişikliklerini iş listesinde görmek.

\* Walking The Board → Son durum? Engel var mı? Ne zaman tamamlanır?

yani amas işlerin durumunu konuşmaktır.

\* Takım üyeleri, board üzerinden takımları eksik ettiğinde belirter ve bunları boarda ekler. yapabildikleri herhangi bir işi done tarafına getirmeye çalışırıor.

\* Amas: önce yüksek öncelikli halemler ilerletilmeye çalışılır.

### Kanban

- görevler: görselleştirme

- görevleri: Sıkılık testim eder

- Tanımlanmış Rol yok

### Scrum

- Kiso zaman aralıklarında proje testim etme

- 1-4 haftalık dönenlede parça parça testim eder

- ScrumMaster, product owner developer Team

- proje aksı ortasında iyileştirmeye izin verir.
  - Sprint + Sırasında doğası ile yapılmalıdır.
  - proje percasını bitmede için geçen süreyi kullanarak öncelikle öğrenir.
  - Sprintler ile üretim hizini ölçer.
  - Çok geçici önceliklere sahip projeler.
  - Sabit öncelikli projeler.
  - Kanbanboard
  - product backlog, Sprint backlog, product increments
- DevOps**
- önceliği  
+  
Sprint sonundaki  
yenilikler

\* Geliştirme, IT operasyonları, kalite menedżerliği gibi aynı çalışma roller arasında koordinasyonu sağlar.

1) Continuous Integration: Sürekli entegrasyon, kod değişikliklerini, testlerin yapılması depoda sık sık bulaleştirme. release için gereken süreci azaltmaya yardımcı olur.

2) Continuous Delivery: Sürekli teslimat, kod değişikliklerini test ortamlarına otomatik olarak dağıtmayı, pipeline tamam.

3) Infrastructure as Code (IaC): Altyapının kod ve yazılım kullanılarak takip edilmesi ve yönetilmesi.

4) Continuous Monitoring: DevOps takımlarının tüm aşamalarını izlemesidir.

5) Microservices: Tek bir uygulamanın bir dizi küçük servis şeklinde oluşturduğu bir tasarımdır.

6) Configuration Management: İşlem Sistemi, Sunucu yapılandırması, ve operasyonel görevleri otomatikleştirme

## Güçk Yöntemler: Ölçüleme

\* Güçk yöntemler, kicik - orta ölçekli projelerde basarılı

Scaling up: takımın kapasitesini artırmak demek (kicük takımlar için)

- daha deneyimli geliştiriciler
- daha iyi araçlar
- roller netleştirilip engeller kaldırılır.

Scaling Out: Daha büyük işleri güçk şekilde yönetmek

- Birden fazla takım oluşturma
- Organize etme

## Güçk Yöntemlerin Problemleri:

1) Kayıt dökümü, büyük şirketlerin sözleşmeleri imzalamaası ile gelişir.

2) Güçk yöntemler yeri: yazılım geliştirmeye uygun, yazılım inovasyonuna değil   
 → müsterileri geliştirme sürecine dahil etmek  
 → Original geliştirme ekibi giderse sorun çözebilir  
 → ürün dokümantasyonları

3) Ayn. fiziksel ortamlarla işin tasarılmamıştır. Dünya genelde dagitim takimlar için gök uygun değil

## Gök Takımı Scrum'un Temel Özellikleri

1) Rol Çoğaltma

Her takiminin kendi çalışma kişisini, Scrum Master için bir ürün sahibi vardır.

2) Ürün Mimarları

### 3) Scrum (Release) Ayarlama:

Her takimin release tarihler: ayarlanır.

#### 4) Scrum Scrumları

Her takim temsilcisinin ilerleyen! farzımları ve o gün yapılacak iş planlamak için bir araya geldiği Scrum of Scrum vardır.

Günlük Yazılım Bitti.

#### Gereksinim Mühendisliği

> hizmetler: ve kısıtlamaları bulma, analiz etme, sistemlerin yapma sürecidir.

#### Gereksinim Türleri:

Kullanıcı gereksinimleri → hangi hizmetler: sağlanır, beklenenin ve hangi kısıtlamaların altında çalışmas, gerekliliğinin ifadesidir.

Sistem gereksinimleri → Sistem fonksiyonlarının, hizmetlerinin ve operasyonel kısıtlamalarının açıklamasını ortaya koyan dokümanlardır.

#### paydaş Türleri:

- > Son kullanıcılar
- > Sistem yöneticileri
- > Sistem sahipleri
- > Dış paydaşlar

> Günlük yöntemler aracılık gereksinim mühendisliğini kullanır.

#### Fonksiyonel Gereksinimler

- > Sistemin belirli girdilere ve durumlara nasıl tepki vermesi gerekliliği ile ilgili ifadeler
- > Sistemin yapması gerekenler belirtebilir.

#### Fonksiyonel Olmayan Gereksinimler

- > Sistem tarafından sunulan hizmetler veya işlevler üzerindeki kısıtlamaları.
- > Bütün olası sistemlere uygulanır.

#### Domain Gereksinimler: → Sistem üzerindeki kısıtlamalar.

Fonksiyonel kullanım gereklilikleri → Sistemin ne yapması gerektigine dair üst düzey ifadeler

Fonksiyonel Sistem Gereklilikleri → Sistem hizmetlerini öryntili olarak tanımlar

Gereklilik Belirsizliği → fonksiyonel gereklilikler kesin olarak belirtilmediğinde

Gereklilik Tanı Dulosu → Gerçeklik tüm işlevlerin aksiyonlarını içermelidir.

" Tutarlı Olması → Sistemin işlevlerinin aksiyonlarında herhangi bir çatışma veya çelişki olmamalıdır.

### Fonksiyonel Olmayan Gereklilik Türleri:

Ürün Gereklilikleri → Teslim edilen ürünün belirli bir şekilde durumları gerekliliklerinden

Kuramsal Gereklilikler →

Dış Gereklilikler → Sistem ve geliştirme sürecinin dışındaki faktörlerden kaynaklanan gereklilikler

### Fonksiyonel Olmayan Gereklilikler Belirtmek için Metrikler:

- 1) Hiz
- 2) Boyut
- 3) Kullanim kolaylığı
- 4) Güvenilirlik (Reliability)
- 5) Sağlamlık (Robustness)
- 6) Taşınlabilirlik

### Gereklilik Mühendisliği Süreçleri

Proses

- 1) Gerekliliklerin Ortaya Çıkarılması ve Analizi → -Gereklilik Keşf.
- 2) Gerekliliklerin Spesifikasyonu ↓ -Gereklilik Siniflandırılması
- 3) Gereklilik Doğrulaması (Validation) - Gözleme - Önceliklendirme ve Mütakaree
- 4) Gereklilik Yönetimi - Gözlem Uyga etnografiya - Gerekliliklerin Spesifikasyonu

**Senaryolar** → Örnek kullanıcı etkileşimi: Oturumlarının açılmasıdır.

Kullanıcı hikayeleri: bir sistem hakkında aynılıkla girmeler.

git init → yeni bir repo başlatır

git clone → uzak bir depoya yerel bilgiyararına indirir

git status → çalışma alanındaki dosya durumunu gösterir

git add → değişiklikler sahneye alır

git commit -m " " → Değişikliklerin birinci olarak kaydedilir

git remote add origin → yerel repoyu uzak depoya bağlar

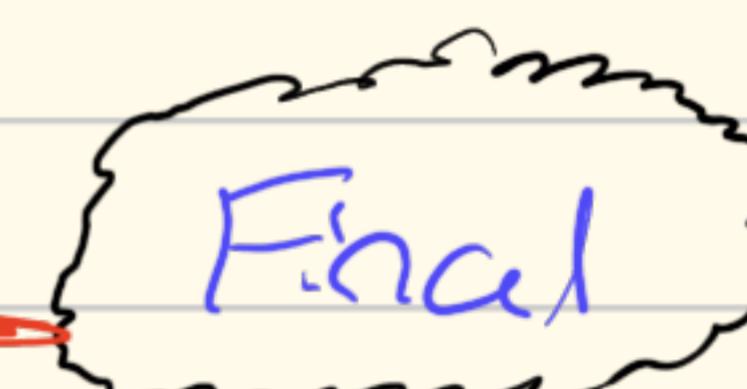
git push → Değişiklikler uzak depoya gönderilir

git pull → uzaktaki değişiklikler yerel depoya çekilir, birleşenin

git fetch → " " " " " , birleşmesi

git branch → Tüm dalları listeler

git branch -d <ism> → belirtilen dallı siler



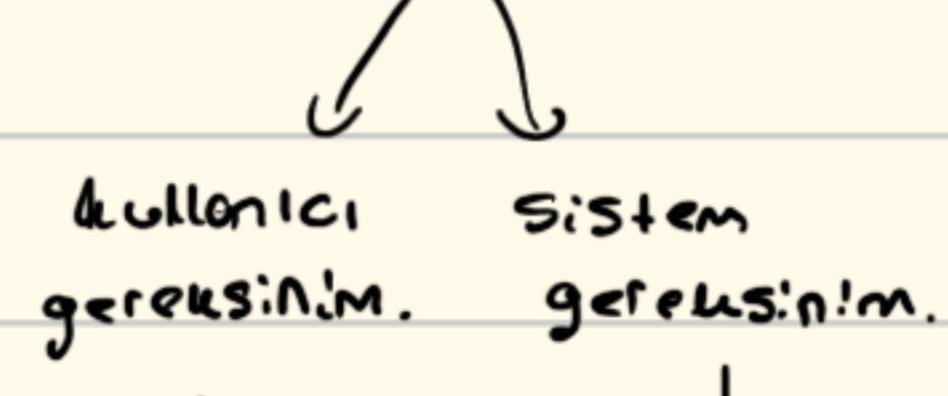
**Gerçekşim Mühendisliği Süreçleri** → sistem ne yapması gerektiği ve kısıtlamalar tanımlar

1- Gereksinimin ortaya çıkarılması (elicitation) ve analizi

2- Gereksinimlerin spesifikasyonu → Kullanıcı ve sistem gereksinimlerinin dokümana yapılması

3- Gereksinimlerin doğrulanması (validation)

4- Gereksinim yönetimi



**Gereksinimlerin Ortaya Çıkartılması Aşamaları**

1- Gereksinim keşfi → Domain gereksinimler de bu aşamada keşfedilir

2- Gereksinim sınıflandırması ve organizasyonu →

3- Gereksinim önceliklendirme ve miktarere →

4- Gereksinimlerin spesifikasyonu → Dokümantasyon yapılır ve bir sonraki aşamaya geçirilir

**Gereksinimlerin ortaya çıkartılması teknikleri**

1- insanlarla görüşme

2- insanları gözlemlene

(Kullanım Senaryoları)  
use cases → Kullanıcılar ve sistem arasındaki etkileşimi grafik ve metin kullanarak açıklamak

## Gereksinim Dogrulama Teknikleri:

- 1- Gereksinim incelemeleri
- 2- prototipleme
- 3- test senaryosu oluşturmak

## Proje Planlama

### Planlama aşamaları:

- 1- Teklif aşaması
- 2- Proje başlatma "
- 3- Periyodik aşama → planın proje boyunca güncellenmesi

### Yazılım Fiyatlandırmasını Etkileyen Faktörler

- 1- Sözleşme şartları
- 2- Maliyet tahmini belirsizliği
- 3- Finansal sağlık
- 4- Pazar fırsatı
- 5- Gereksinim Dolgalanması.

### Görevlerde Bulunan Özellikler

- 1- Süre
- 2- efor tahmini
- 3- son tarih (deadline)
- 4- Bitiş noktası (endpoint)

Kilometre taşları (milestones) → projedeki ilerlemeyi değerlendirebileceğimiz projektroniklerindaki noktalardır.

### Plan Oluş. Proje Planlaması:

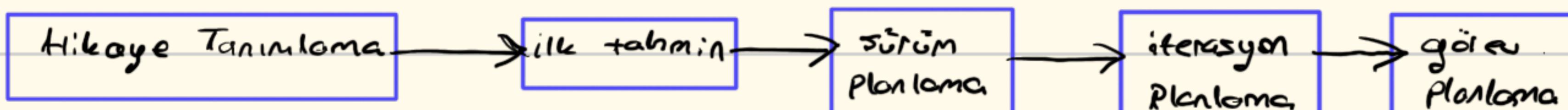
- Giriş
- Proje organizasyonu
- Risk Analizi
- Donanım ve yazılım kaynakları Gereklilikleri

### Geçici Planlama Aşamaları:

- 1- Süre Planlaması
- 2- Yükleme Planlama

- iş döngüsü
- Proje zamanlaması.
- izleme ve raporlama mekanizmaları

### Extreme Programming Planlama Aşamaları:



**COCOMO** → maliyet modellemesidir, bağımsız bir modeldir.

- 1- Uygulama kompozisyon modeli → GUI uygulamaları için uygundur.
- 2- Baştanış tasarım modeli → tasarım net degilken ilk tahminler yapmaya yarar
- 3- Yeniden kullanım modeli → yeniden kullanım bilgilerinin maliyetini hesaplamak için kullanılır.
- 4- Mihari sonrası model → üst terimi ( $B$ ) , çarpanları ( $M$ )

↓	↓
yapılmamış	her belli görevin
bağlantılarının	belirli bir özelliğini
efora etmesi	terstil eder, etrafı doğrudan etkiler

**Sistem Modelleme** → Bir sistemin soyut modelinin geliştirme sürecidir.

### Sistem Perspektifleri:

- 1- Harici perspektif → Sistemin bağlamını veya ortamını modelleme
- 2- Etüdesim " → Sistem ile ortam arasındaki etüdesini modelleme
- 3- Yapısal " → Bir sistemin organizasyonunu modelleme
- 4- Dauranışsal " → Sistemin dinamik davranışını modelleme

### UML Diagram Türleri:

- 1- Aktivite Diagramı → Uer işlevinde yer alan etkinlikleri gösteren
- 2- Kullanım Senaryosu " → Sistem ile ortam arasındaki etüdesini
- 3- Ardişik " → Aktörler ve sistem arasındaki "
- 4- Sınıf " → Sistemdeki sınıfları ve bu sınıflar arası ilişkileri
- 5- Durum " → Sistemin iş ve durum olayları ilişkisi

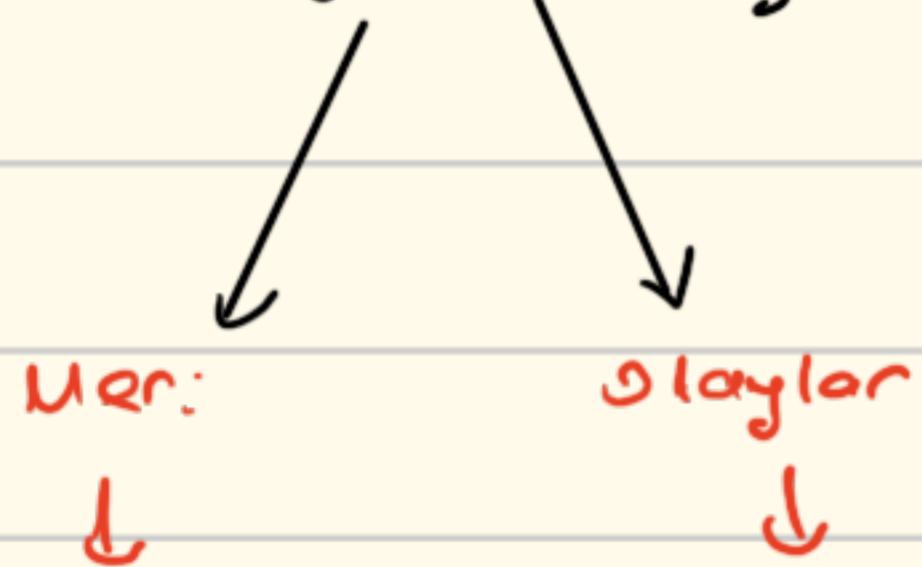
**Bağımlı Modeller:** → Sistemin operasyonel bağlamını göstermek için kullanılır.

**Etkileşim Modeller:** → Kullanıcı gerekliliklerinin belirlenmesine yardımcı olur.

- ↳ Kullanıcı etkileşiminin modellenmesi
- ↳ Sistemden sisteme etkileşimin "
- ↳ Bilesen etkileşiminin "

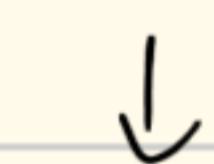
**Düzenlik Modeller:** → Bir sistem yürütülürken dinamik davranışının modellenmesi.

Gereklilikler uygulamalara göre teknikin ne olması gerekligini söyleyler

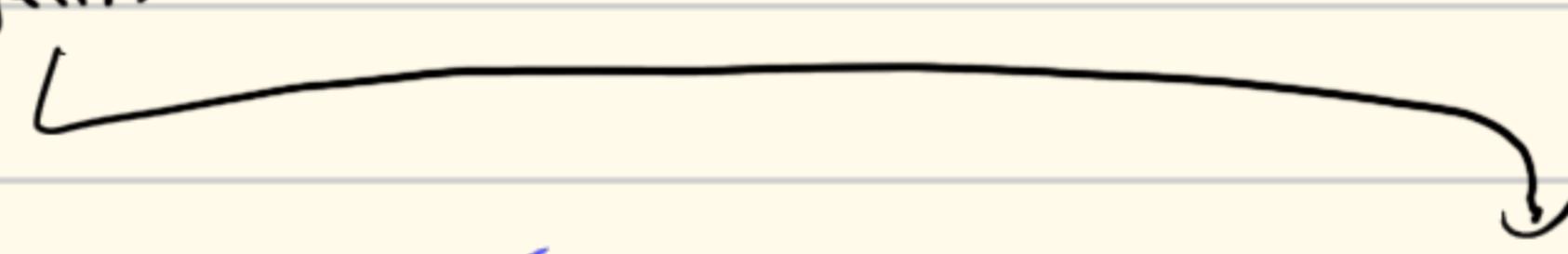


Sistem tarafından sistem işlevini

işlemesi gerekken tetikleyen bazı olaylar  
veriler gelir meydana gelir.



User: Oluştu. modelleme



Olay Oluştu. modelleme

- Bilgisi organize etme

- İş akışı optimize etme

- UML class diyagramı

- UML activity diyagramı

- Verilerin tekrarsız ve

- İş süreçlerinin datica iş'

bütünlüğe olması

antlaşılması, öngörmeler ve

üyelleştirilmesi

## Model Oluştu. Mühendisliği

### Avantaj

### Desavantaj

+ Sistemlerin daha büyük soyutlama seviyelerinde değerlendirilmesini sağlar

- Modeler soyutlama içindir ve gerçekleştirmek için mutlaka doğru değildir.

+ Otomatik olarak kod oluşturmak

- Kod üretmekte elde edilen başarısızlığı platformlar için dönüştürme

yeni platformlara uygunlaşmasını kolaylaştırır. geni platformlar için dönüştürme

gesitmenin maliyetlerinden daha ağır busutsuluğu

## Model Türleri

1 - Hesaplımdan bağımsız model (CIM)

2 - Platformdan bağımsız model (PIM)

3 - Platforma özgü modeller (PSM)

MOMI → Modelleme, Organizasyon, metodoloji, işleyişim

↳ Bütinçü yazılım geliştirme yaklaşımı sunar

## Mimari Tasarım

### Mimari Tasarımı

1- Küçük mimari → Bireysel programların mimariyle ilgilenir.

2- Büyük mimari → Diğer sistemleri, programları ve program bileşenlerini içeren karmaşık kurumsal sistemlerin mimariyle ilgilenir.

### Açık Mimarinin Avantajları

1- Paydaş iletişim

2- Sistem analizi

3- Büyük ölçüde yeniden kullanım

Mimari stil ve yapı seçimi sistemin farklı olmayan gerekliliklerine bağlı olmalıdır.

1- performans

2- Güvenlik (Security)

3- Emniyet (Safety)

4- Kullanılabilirlik

5- Sürdürülebilirlik

## Yazılım Mimarisi'nın 4+1 perspektif modeli

1- montiksel perspektif

2- süreç "

3- geliştirme "

4- fiziksel "

## Mimari Desenler (Önemli)

1- Model - view - controller deseni → uygulama bileşenlerini görevlere göre ayırmak

2- katmanlı mimari " → sistemde katmanlarına uygun olarak bağımlılıkları azaltmak

3- Depo " → farklı bileşenlerin ortak bir veri deposu ile iletişim kurmak

4- istemci sunucusu " → görevlendirilen istemci ve sunucu arasında doğrudan açı ile iletişim sağlama

5- boru ve filtre " → veriyi adım adım işlemek için bir zincir oluşturma

## Uygulama mimarilerinin kullanımı

1- Mimari tasarım süreci: için bir koşulsuz yolculuk → mimari model koşulsuzlaşdırılacak

2- Tasarım kontrol listesi → tasarımını diğer mimari modellerle karşılaştırarak eksik ve zayıf yönlerin tespiti

3- Geliştirme evrimini organize etme aracı → her bileşen: bir ekip içinde otayarak paralel geliştirme

4- geninden kullanım değerlendirme aracı → data önceliği bileşenlerin yeniliğinde kullanımını takip etmeyeceğini onlara iğnelemek

5- Ortalık sınırlı kullanımını → ekip içi iletişimini güçlendirme

Slayt 10 Başlıyor

## Nesne Yönelimli Tasarım Süreci Aşamaları:

- 1- Sistemin bağlamının ve dış etkileşiminin tanımlanması
- 2- Sistem mimarisinin tasarılanması
- 3- Ana sistem nesnelerinin tanımlanması
- 4- Tasarım modelinin geliştirilmesi
- 5- Nesne arayüzlerinin belirlenmesi

**Bağlam Modeli:** → geliştirilmekte olan sistemin içerisindeki diğer sistemleri gösteren yapısal bir model

**Etkileşim Modeli:** → Sistemin kulanıldığı外界 ile nesil etkileşimi gösteren dinamik bir model

> modeller, sistem gereklilikleri ile sistemin gereklilikleri arasındaki köprüdür.

> Bir tasarım geliştirmek için UML'yi kullanıldığında üç tür tasarım modeli vardır

Yapısal modeller → nesne sınıfları ve ilişkiler

Dinamik modeller → nesneler arasındaki dinamik etkileşimler

Ayrıca üç UML model türü de aynı zamanda很重要이다.

- Alt sistem modelleri
- Ardişik "
- Durum makinesi "

**Yazılımın yeniden kullanılması (Reuse)** → yazılım geliştirirken mevcut kodlardan

olabileceğinca yararlanılmalı. → Slony 10, sayfa 42 eksek olabilir tam anlamadım

2) konfigürasyon yönetimi

3) Ana Bilgiye - hedef geliştirme (Host - target development)

## Yazılımın Yeniden Kullanım Seviyeleri: (Önemli)

- 1- Soyutlama (Abstraction) seviyesi → tasarımda barındırılan soyutlamaların bilgisel seviyeleri
- 2- Nesne (Object) " → kodu kendisini yazmak yerine kütüphane gibi nesneler kullanılır.
- 3- Bileşen (Component) " → nesnelerin yanında Nesne ve sınıflar kisleştiğinden
- 4- Sistem (System) " → tüm uygulama sistemleri birlikte kullanılır.

## Release

vs

## Version

- Bir yazılım ürününün kullanımına sunulan belirli bir versiyonu
- release taraflı bir versiyon
- test ve geliştirme amacıyla oluşturulmuş olabilir, konuya sunulması zorunda değil.

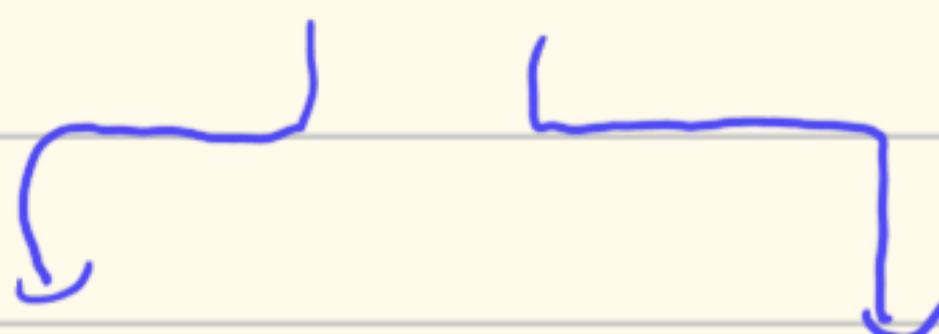
## Konfigürasyon yönetim: featureler:

- 1- versiyon yönetimi
- 2- sistem entegrasyonu
- 3- sonun işleme
- 4- sunum yönetimi

## 2.2.2 Slayt

### Tasarım Kavramları

#### I) Soyutlama (Abstraction)



Prosedürel  
Soyutlama

Ueri  
Soyutlaması



İşlevli tallımlar

Ueri nesnesi!

anlamına gelir

tanımlayan adlandırılmış

Örnek: "create"

bir ueri koleksiyonu

Örnek: create ile

oluşturduğumuz

nesnenin kendisi!

- 2) Mimari (Architecture) → yazılımın genel yapısını ve bu yapının genel bütünlüğünü
- 3) Desenler (Patterns) → tasarım problemlerini çözer
- 4) İşlemlerin Ayrılması (Separation of concerns) → karmasık problemlerin parçaları arasında bölünmesi
- 5) Modülerlik (Modularity) → işlevlerin ayrılması entomenci uygulanır  
Avantajları:
- 1- geliştirmenin daha kolay planlanabilmesi
  - 2- değişiklerin daha kolay uygun sağlayabilmesi
  - 3- test ve debug daha kolay
  - 4- uzun vadeli hizimin kolaylığı
- 6) Bilgi Gizleme (Information Hiding) → modüller, bir modülde bulunan bilgilere diğer modüller tarafından erişilemeyecek ve buna ihtiyaç duymayan şekilde tasarılanmalıdır.
- 7) Fonksiyonel Bağımsızlık (Functional Independence) → işlevlerin ayrılması, modülerlik, soyutlama ve bilgi gizleme konularının sonucudur.  
> bağımsızlığın iki kriteri vardır:  
1) yapısalılık → Bir modülün işlevsel gücü  
2) bağılasm → Bilgi gizleme konusunun bir uygulamasıdır. modüller arası ara bağlantı makasıdır.
- 8) Adım Adım iyileştirme (Stepwise Refinement) → yukarıdan aşağıya bir tasarım stratejisidir. Soyutlama ve adım adım iyileştirme birbirinin tamamlama makasıdır.
- 9) Geniden Düzeneşme (Refactoring) → kodun doğası davranışını değiştirmeyecek olarak iş yapısını iyileştirecek şekilde düzeneşmesidir.  
> genellikle yazılım geliştirme süreçlerinde yaygın kullanılır.
- 10) Tasarım Sınıfları (Design Classes) → Analiz modeli, her dizi analiz sınıfı belirler. Tasarım modeli getirildikçe analiz sınıflarını iyileştiren tasarım sınıflarını oluşturur.

İyi bir tasarımcı sınıfın 4 özelliği vardır;

- 1- Tom yeteri:
- 2- Basitlik
- 3- Yüksek yorumluk
- 4- Düşük koğuşım

### (Önemli)

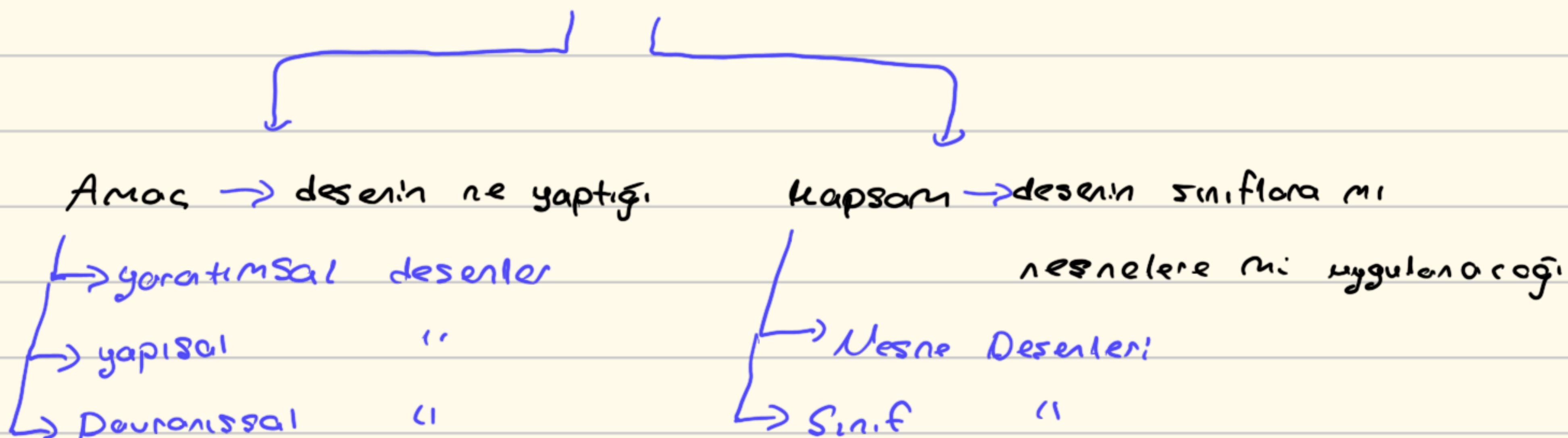
Tasarım Deseni Nedir → diğer tasarımcıların bilgi ve deneyimlerini yeniden kullanmanın bir yol, tekrar eden problemlere geneldeki bir çözüm sunar  
“tekerleği yeniden icat etmek” zorunda kalmayıza

> Bir tasarımcının belirli bir çözümü sunmasına neden olan sistem göçümleridir.

Etki: tasarımcı desen;

- 1- Sınırlı bir probleme özel bir çözüm
- 2- pratikte kontrollü bir çözüm
- 3- tasarım ve mimari ilişkisi tanımlar
- 4- yaklaşımı ve kullanımlığını arızadan zaif tutar

### Tasarım Deseni



Amaclarına göre desenler:

- 1) yaratımsal desenler → nesnelerin yaratılması, kompozisyonu ve temsilii
- 2) yapısal " → data büyük bir yapı oluşturmak için sınıf ve nesnelerin organizasyonu
- 3) Dürounissal " → nesneler arası sorumluluk atanması ve iletişim

Kapsamlarına göre desenler

- 1) Nesne desenleri → Nesneler arasındaki ilişkileri belirtir
- 2) Sınıf " → sınıflar ve alt sınıflar arasındaki ilişkiliyi belirtir.

1) Adapter deseni → sistemimize uyumayan bir kodu adapter deseni sayesinde sisteme uygun hale getirebiliyoruz



2) Singleton Deseni → Bir sınıfın sadece bir nesne oluşturmak için kullanılır. Ancak, oluşturulan nesneye global erişim noktası sağlanır. Oluşturduğumuz nesne tüm isteklere kendisi cevap verir. Sistem girdiği sürece yeni bir nesne oluşturulamaz.

Örnek durumlar → thread pool, önbellekler, iletişim kutuları, log kaynakları, vb.

Singleton multi-thread uygulamalarda → `getInstance()` metodu synchronized yapılır.

iki thread aynı anda `getInstance()` gibi bir singleton erişim metodunu çağırabilir eğer nesne henüz oluşturulmamışsa, ikinci thread için 2. tane nesne oluşturabilir. Bu singleton uygulamaları uygulama çözümü için → Double-Checked Locking kullanılır.

`public static Singleton getInstance()`

`if(instance == null)` { 1. kontrol (synchronized olmadan)

`Synchronized (Singleton.class)` } *yani zaten ilk seferde同步*

`if(instance == null)` { 2. kontrol (kilitli)

`instance = new Singleton();`

}

3

3

3) Observer Deseni → Nesneler arasında bir tür çoğal (one-to-many) bağımlılık türleri, böylece bir nesne durum değişirdiğinde, ona bağlı tüm nesneler otomatik olarak bilgilendirilir ve güncellenir.

Bir de bir sınıf kod uygulamaz ama, singletonun kodlarını sorabiliyor.

4) Factory Method Deseni → Bir nesnenin oluşturulma sürecini alt sınıflara bırakın fasonu desenidir.

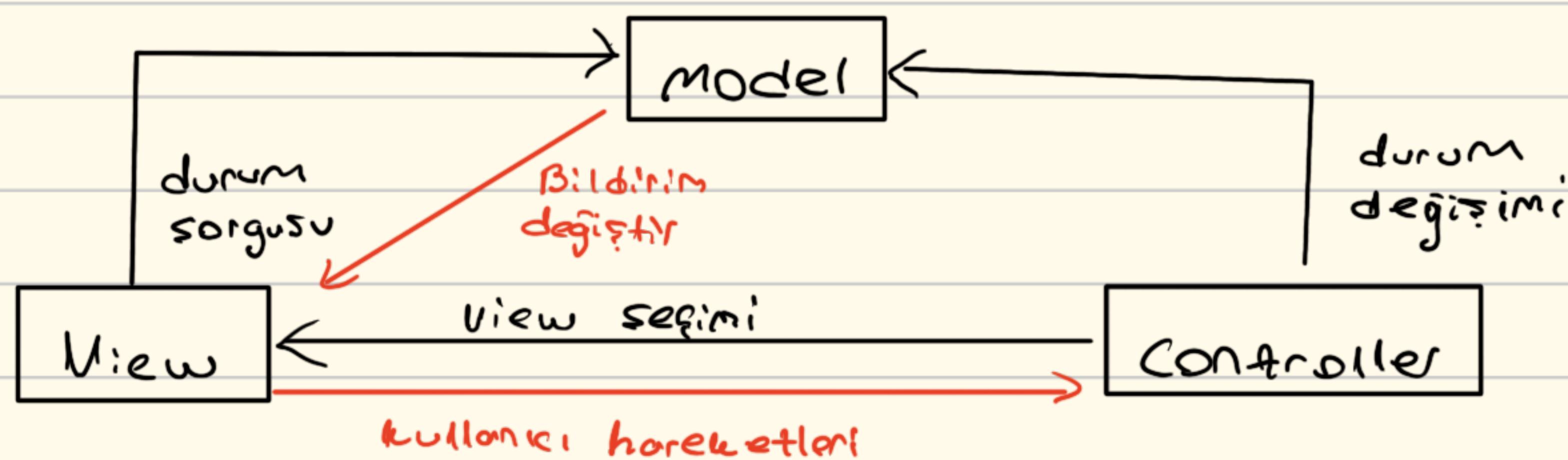
5) Model - View - Controller (MVC) Mimari Deseni →

İç 3' e bölünür:

1- Model → User sınıfı → Ad, şifre

2- View → Ekranındaki label, button → sadece görsellemeye

3- Controller → Tıklamaları algılar, model ile çalışır, view'i günceller



6) Cocoa MVC → Apple iOS / macOS için kullanılan modelidir, temel olarak değişiklik bildirimlerinin controller aracılığıyla view'e iletilmesidir.

Model → User sınıfı → ad, şifre

View + controller → hem görüntü hem etkileşim ile işlenenler → UIView Controller

7) Builder Deseni → Bir nesnenin nasıl oluşturulacağı soyutlar

— Aynı nesnenin farklı temsillerini üretme

Örneğin → Ev sınıfı → cat, pencere, garaj, kapı

EvBuilder → setCat(), setPencere(), setGaraj(), setKapi()

EvBuilder builder = new EvBuilder(); → Builder.setCat(...)

Şekilde kullanıcıya okunabilirlik sağlıyor

