



YMH 311-Yazılım Tasarım ve Mimarisi

Yazılım Mühendisliği'ne Giriş

Prof. Dr. Resul DAŞ

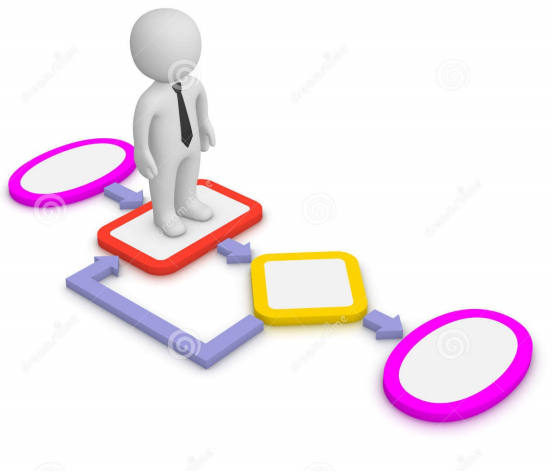
Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Yazılım Nedir?.....	4
Yazılım Mühendisliği.....	15
Yazılımların Sınanması.....	20
Yazılım Maliyetleri.....	22
Yazılım Sistemlerinin Sınıflandırılması.....	23
Yazılım Kalite.....	27

Mantık (Algoritma)

- Mantık, bilginin yapısını incelerken, kesin sonuca ulaşmak için doğru ile yanlış arasındaki akıl yürütme ayrımı yapmaktır.
- Bilgisayarlaştırılmak istenen işin mevcut mantığı, yazılıma yansıtılmak durumundadır.
- Bu nedenle mantık (algoritma) bileşeni yazılımın en önemli bileşenlerinden biridir.



Veri (Bilgi)

- Veri (ing. data), işlenmemiş bilgi veya bilginin ham halidir. Bilgi ise, en basit anlamda verinin işlenmiş şeklidir. Her tür yazılım mutlaka bir veri üzerinde çalışmak durumundadır. Veri dış ortamdan alınabileceği gibi, yazılım içerisinde de üretilebilir.
- Yazılımın temel amacı “veri” yi “bilgi” ye dönüştürmektir.



Belge (Dokümanlar)

- Yazılım üretimi bir mühendislik disiplini gerektirir. Mühendislik çalışmalarında izlenen yol ya da kullanılan yaklaşımlar yazılım üretimi için de geçerlidir.
- Yazılım üretimi sırasında, birçok aşamada yapılan ara üretilere ait bilgiler (planlama, analiz, tasarım, gerçekleştirim, vb. bilgileri) belli bir düzende belgelenmelidirler.



İnsan (Kullanıcı, geliştirici)

- İki boyutludur; yazılımı geliştirenler ve kullananlar.
- Günümüzde artık tek kişi ile yazılım geliştirmekten söz edilmemektedir.
- Yazılım üretimi için bir takım oluşturulmakta ve takımın uyumlu çalışabilmesi için çeşitli yöntemler geliştirilmektedir.

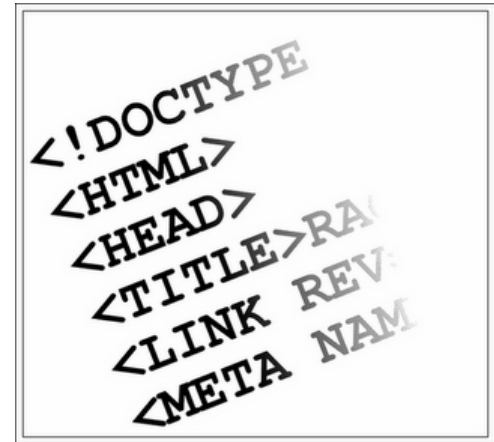


Program (Kod)

- Yazılımın ana çıktısı sonuçta bir bilgisayar programıdır.
- Program işleme alındıktan sonra bakım çalışmaları sürekli olarak gündeme gelir.

Bunun iki temel nedeni:

- Hiç bir program bütünüyle her olasılık göz önüne alınarak test edilemez.
- İşletmeler doğaları gereği dinamik bir yapıya sahiptir ve zaman içerisinde sürekli olarak yeni istek ve gereksinimler ortaya çıkabilmektedir.



Yazılım Donanım Karşılaştırması

- Yazılım geliştirilir vs donanım üretilir. (fabrika ortamında seri üretim)
- Donanım bileşenleri dışarıdan temin edilebilir, ancak yazılımı oluşturan parçalar için bu çoğu zaman mümkün değildir (günümüzde “yeniden kullanılabilir yazılım” %1-2).



Yazılım Donanım Karşılaştırması

- Yazılım eskimez.
- Oysa, her donanımın belli bir ömrü vardır. Ömrünü tamamlayan donanım yenisi ile değiştirilir.
- Yazılımın eskimesi ortaya çıkabilecek yeni ihtiyaçları karşılayamaması, kullandığı teknolojinin eskimesi olarak tanımlanabilir.
- Yeni gereksinimler yazılıma ekler yaparak yansıtılır.

Yazılım Donanım Karşılaştırması

- Yazılım en az donanım kadar önemlidir.
- Diyaliz makinelerinde kullanılan yazılımların 2000 yılı uyumsuzluğundan ötürü, bir çok diyaliz makinesi çalışmamış ve böbrek hastaları zor durumda kalmıştır.
- Japonya'da telefon yazılımında ortaya çıkan bir yazılım hatası onbinlerce abonenin saatlerce telefon konuşması yapamamasına neden olmuştur.

Yazılım Donanım Karşılaştırması

- Yazılım kopyalama ve donanım kopyalama farklıdır.
- Hata toleransı amacıyla, hayati olan bir donanımın sistemde bir kopyası daha bulundurulur ve sistemde biri arızalandığında diğeri çalışmayı devralabilir.
- Oysa, bir yazılımı sistemde iki ayrı bilgisayar üzerine kopyalamak oluşabilecek hatalara çözüm olmayacaktır. Belki, sisteme aynı işi yapan iki farklı eş yazılım yüklenmesi çözüm olabilir (**kritik yazılım sistemleri- uçak avionics**).

Yazılım Üretim Ortamı

- Değişik yetenekte bir çok personel (analist, programcı, test uzmanı, vs.)
- Yazılım çıktısı ile ilgilenen kullanıcılar
- Yeniliğe tepki gösteren kullanıcılar ve yöneticiler
- Yeterince tanımlanmamış kullanıcı beklentileri
- Personel değişim oranının yüksekliği
- Yüksek eğitim maliyetleri
- Dışsal ve içsel kısıtlar (zaman, maliyet, işgücü, vs)
- Standart ve yöntem eksiklikleri
- Verimsiz kaynak kullanımı
- Mevcut yazılımlardaki kalitesizlik
- Yüksek üretim maliyeti



Yazılım Mühendisliği

IEEE Tanımı (1993)

“Yazılım Mühendisliği:

Sistemli, düzenli, ölçülebilir bir yaklaşımın yazılım geliştirmede, yazılımın işlenilmesinde ve bakımında uygulanmasıdır.

Diğer bir deyişle mühendisliğin yazılıma uygulanmasıdır.



Yazılım Mühendisliği

- Yazılım üretiminin mühendislik yöntemleriyle yapılmasını öngören ve bu yönde;
- yöntem,
 - araç
 - teknik ve
 - metodolojiler
- üreten bir disiplindir.



Yazılım Mühendisliği

- Yazılım mühendisliği bir yöntemler, teknikler ve araçlar kümesi olarak değerlendirilebilir.
- Yazılım mühendisliğinin hedefi; yazılım üretimindeki karmaşıklıkları gidermektir.
- Geçmişte kullanılan iş akış şemaları gibi yöntemler günümüzde yetersiz kalmaktadır.
- Ayrıca, yazılım üretimi işi tek kişinin başarabileceği boyuttan çıkmış ve bir takım iş biçimine dönüşmüştür.

Yazılım Mühendisi

- Yazılım Mühendisliği İşini yapan kişidir.
- Temel hedefi; **üretimin en az maliyet ve en yüksek nitelikte yapılmasını sağlamaktır.**
- Programcı değildir. Ancak programcının tüm yeteneklerine sahiptir.
- Yazılımın daha çok mantıksal boyutuyla ilgilenir ve işi insanlarla ilişkiyi gerektirir.
- Sistem analisti de değildir. Farkı; analist sadece sistemin analiz aşaması ile ilgilenirken, yazılım mühendisi tüm aşamaların içindedir.

Yazılım Hataları

- Yazılım hataları, yazılım yaşam döngüsünde çok önemli yer tutan unsurlardan biridir.
- Yazılım geliştirme de karşılaşılan en sıkıcı durumdur.
- Hatalar yüzünden yazılım maliyeti artmaktadır.
- Yazılım geliştirme sürecini uzatmaktadır.
- Çözümü erken bulunmayan hatalar bazen uyulması gereken sistemi zor durumda bırakarak zamansal problemler oluşturmaktadır



Yazılımların Sınanması

- Bir programı tüm ayrıntıları ile test etmek teorik olarak mümkün olmakla birlikte, uygulamada bu mümkün değildir.
- Yazılım ancak sınırlı sayıda veri ile sınanabilir.

Mantıksal Tasarım	%20
İşlevsel Tasarım	%15
Kodlama	%30

Hataların “Yayılma” Özelliği

➤ Yazılımda Hata Düzeltme Maliyetleri:

- Yazılım üretimindeki hatalar yayılma özelliği gösterir.
- Bu nedenle, hata düzeltme maliyetleri ilerleyen aşamalarda giderek artar.

Analiz	1
Tasarım	5
Kodlama	10
Test	25
Kabul Testi	50
İşletim	100



Yazılım Maliyetleri

- Günümüzde yazılım maliyetlerindeki artışlar giderek artmaktadır. "Yazılımımızı alırsanız yanında donanımı ücretsiz olarak sağlayacağız" deyişi zaman içerisinde giderek doğrulanmaktadır.
- Örneğin, günümüzde bir kopyası yüz bin dolar dolayında satılan kurumsal kaynak planlama yazılımlarının bulunduğu gözlemlenmektedir.
- Öte yandan bir kişisel bilgisayar ise 1000 ABD dolarının altında satılmaktadır.
- Yazılımın kopyalanma maliyeti ile donanım kopyalama maliyetinin arasındaki farklılık dikkate alındığında, yazılım maliyetlerinin, donanım maliyetlerine oranla oldukça yüksek olduğu ortaya çıkar.



Yazılım Sistemlerin Sınıflandırılması

- İşlevlerine göre sınıflandırma
- Zamana dayalı özelliklere göre sınıflandırma
- Boyuta göre sınıflandırma



İşlevlerine Göre Sınıflandırma

Hesaplama	Mühendislik Çözümleme
Veri İşleme	Bankacılık
Süreç Temelli	Gömülü Sistemler
Kural Temelli	Robotik, Yapay Zekâ
CAD	Sinyal İşleme

Zamana Dayalı Özelliklere Göre Sınıflandırma

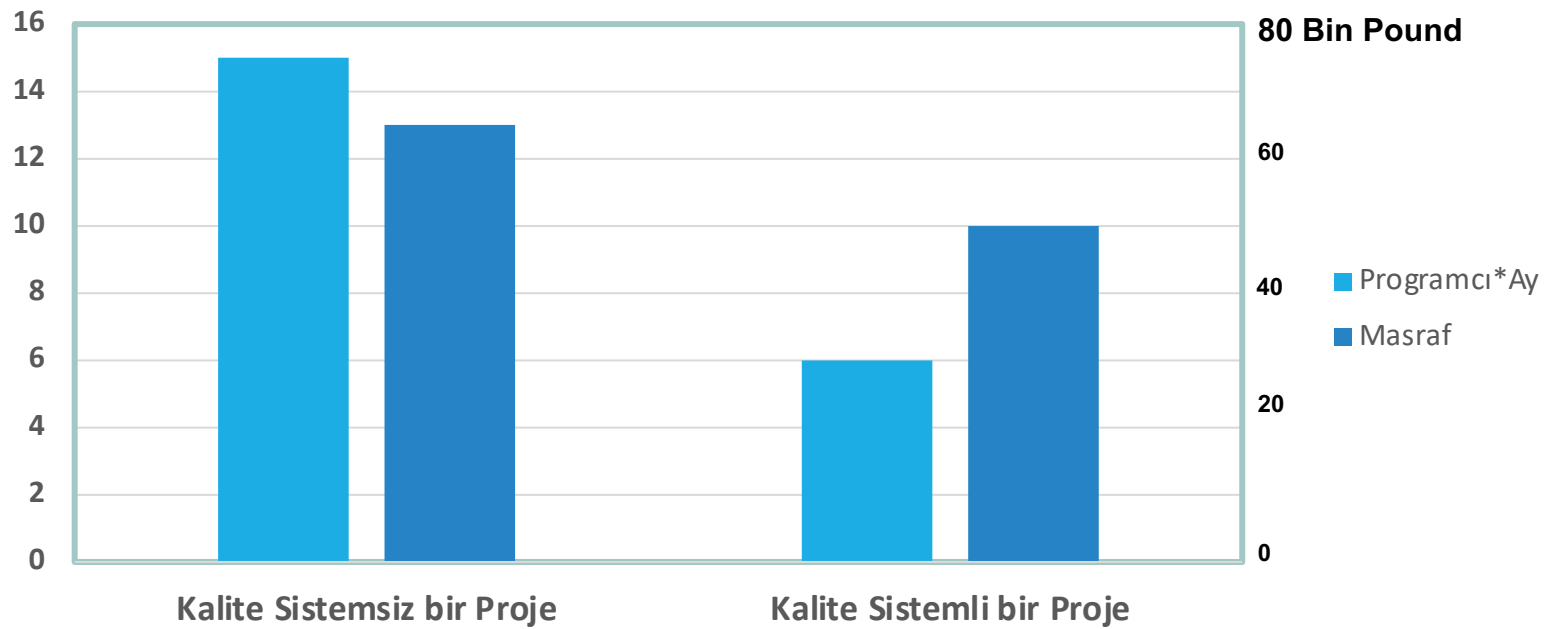
Toplu (Çevrim-Dışı)	Çevrim-İçi
Gerçek Zamanlı	



Boyuta Göre Sınıflandırma

Küçük ($SS < 2000$)	PC Oyunları Öğrenci Projeleri
Orta ($2000 < SS < 100,000$)	CAD BDE Yazılımları
Büyük ($100,000 < SS < 1 \text{ Milyon}$)	İşletim Sistemleri
Çok Büyük ($SS > 1 \text{ Milyon}$)	Komuta Kontrol Sistemleri Hava Tahmini Sistemleri Yıldız Savaşları Sistemleri

Yazılımda Kalite



Yazılımda Kalite

➤ Üretim Süreci Boyunca ara ürünlere ilişkin kalite standartlarının geliştirilmesi ve geliştirme işlemlerinin bu standartlara uygunluğunun denetlenmesidir.

Yazılım kalite sağlama etkinlikleriyle;

- Yazılım maliyetleri düşürülür,
- Yazılım üretiminin yönetimi kolaylaşır,
- Belgeleme ve standart sorunları giderilir.



Yazılımda Kalite

Ekonomi	Tamlık	Yeniden Kullanılabilirlik	Etkinlik	Bütünlük
Güvenirlik	Modülerlik	Belgeleme	Kullanılabilirlik	Temizlik
Değiştirilebilirlik	Geçerlik	Esneklik	Genellik	Sınanabilirlik
Taşınabilirlik	Bakılabilirlik	Anlaşılabilirlik	Birlikte Çalışabilirlik	

Özet

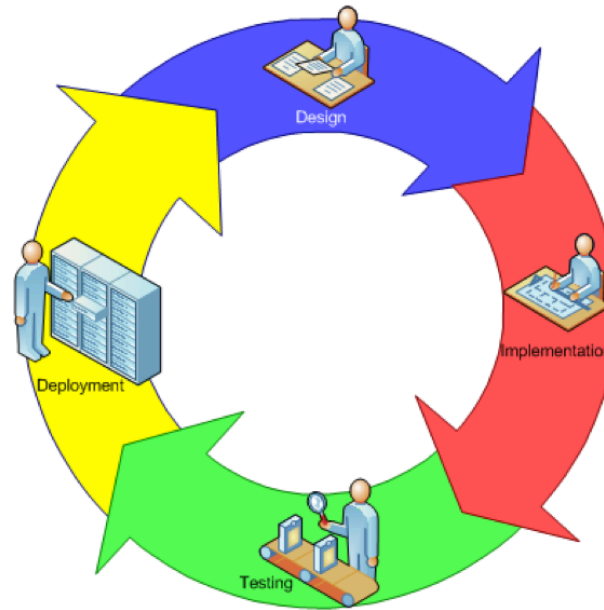
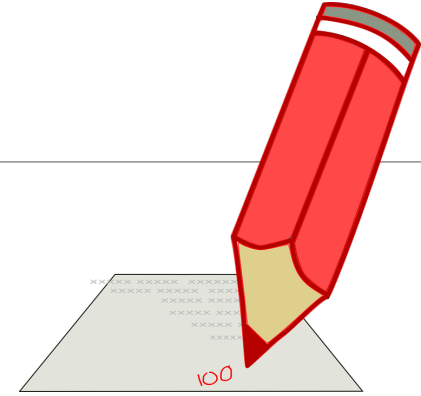
- **Yazılım**= Mantık, veri, belge, insan ve program bileşenlerinin, belirli bir üretim amacına yönelik olarak bir araya getirilmesi, yönetilebilmesi için kullanılabilecek ve üretilen, yöntem, araç, bilgi ve belgelerin tümünü içerir.
- **Yazılım Mühendisliği**: Sistemli, düzenli, ölçülebilir bir yaklaşımın yazılım geliştirmede, yazılımın işlenilmesinde ve bakımında uygulanmasıdır. Diğer bir deyişle mühendisliğin yazılıma uygulanmasıdır.
- **Yazılım mühendisliğinin hedefi**; yazılım üretimindeki karmaşıklıkları gidermektir.
- **Yazılım kalite sağlama etkinlikleriyle**;
 - Yazılım maliyetleri düşürülür,
 - Yazılım üretiminin yönetimi kolaylaşır,
 - Belgeleme ve standart sorunları giderilir.

Sorular

1. Yazılım ile program arasındaki farklılığı belirtiniz.
2. Yazılım ile donanım arasındaki farklılıkları belirtiniz.
3. Neden yazılım giderek pahalılaşırken, donanım ucuzlamaktadır?
4. Yazılım mühendisliği ile diğer mühendislik disiplinlerini karşılaştırınız.
5. Bu bölümde verilen yazılım sınıflandırmasını dikkate alarak, her sınıflandırma için bir yazılım örneği veriniz.
6. Yazılım mühendisi ile programcı arasındaki farklılığı belirtiniz.
7. Yazılım Hatalarında Yayılma neyi ifade etmektedir?
8. Yazılım Maliyetinde hangi Sınıflandırma ağır basmaktadır?
9. İşlevlerine Göre Sınıflandırma da Kural Temelli 'ye Örnek veriniz.
10. Yazılım Kalitesini Tanımlayınız.

Araştırma ödevleri

1. Yazılım Yaşam Döngüsünü Araştırınız.
2. Yazılım Süreç Modellerini Araştırınız.



Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- “Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- “Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Saridoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

<http://blog.alisuleymantopuz.com/2014/08/30/yazilim-mimarisi-ve-tasarimi-nedir/> *

<http://www.akifsahman.com/?p=175>

<https://ece.uwaterloo.ca/~se464/08ST/index.php?src=lecture> * <http://info.psu.edu.sa/psu/cis/azarrad/se505.htm>

<http://www.metinakbulut.com/YAZILIM-MIMARISI/> * http://ceng.gazi.edu.tr/~hkaracan/source/YPY_H3.pdf

<http://iiscs.wssu.edu/drupal/node/3399> * <http://www.cs.toronto.edu/~sme/CSC340F/slides/21-architecture.pdf>

<http://www.users.abo.fi/lpetre/SA10/> * <http://sulc3.com/model.html>

<http://salyangoz.com.tr/blog/2013/11/23/digerleri/yazilim-gelistirme-surec-modelleri-3/>

Sorularınız?



Yazılım Yaşam Döngüsü ve Süreç Modelleri



Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Yazılım Yaşam Döngüsü.....4

Süreç Modelleri.....16

Metodojiler.....71

Amaçlar

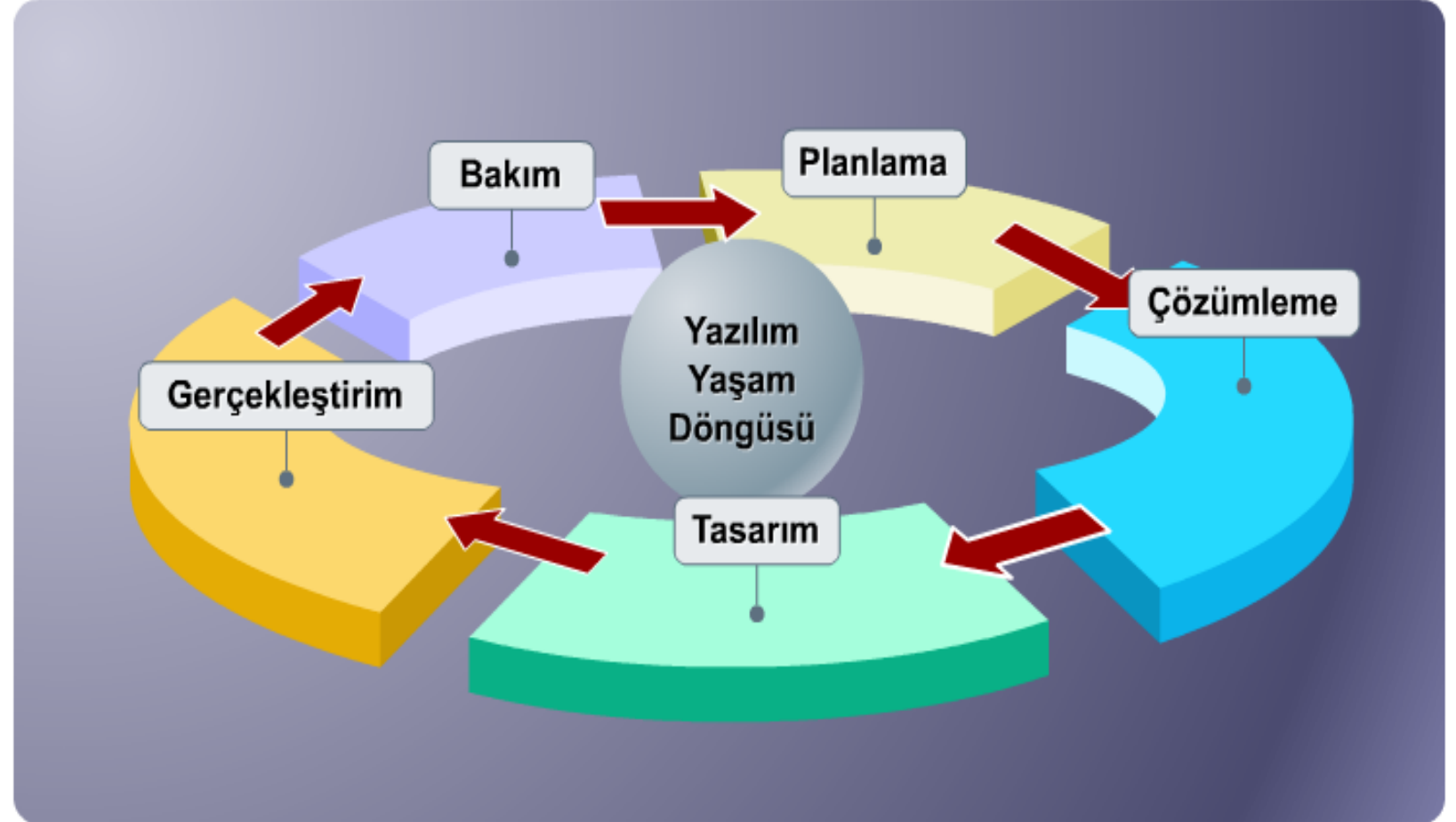
- Yazılım Yaşam Döngüsü 'nün Projelerde ki Önemi Kavramak?
- Yazılım Yaşam Döngüsünde Önemli Kavramlar
- Yazılım Geliştirmede Süreç Modellerinin Önemi?
- Süreç Modellerinin Gelişimi
- Süreç Modeli Çeşitleri ve Uygulanması
- Süreç Modellerinin Avantajları ve Dezavantajları
- Metodoloji Yaklaşımı
- Örnek bir Metodoloji

Yazılım Yaşam Döngüsü Nedir?

- Yazılım yaşam döngüsü, herhangi bir yazılımın, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır.
- Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu aşamalar bir döngü biçiminde ele alınır.
- Döngü içerisinde herhangi bir aşama da geriye dönmek ve tekrar ilerlemek söz konusudur.
- Yazılım yaşam döngüsü tek yönlü ve doğrusal olduğu düşünülmemelidir.

Yazılım Yaşam Döngüsü Temel Adımları

- **Planlama**
- **Çözümleme**
- **Tasarım**
- **Gerçekleştirim**
- **Bakım**



Planlama

- Üretilcek yazılım ile ilgili olarak, personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.

Çözümleme

- Yazılım işlevleri ile gereksinimlerin ayrıntılı olarak çıkarıldığı aşamadır.
- Bu aşamada temel olarak mevcut sistemde var olan işler incelenir, temel sorunlar ortaya çıkarılarak, yazılımın çözümleyebilecekleri vurgulanır.
- Temel amaç, bir yazılım mühendisi gözüyle mevcut yapıdaki işlerin ortaya çıkarılması ve doğru olarak algılanıp algılanmadığının belirlenmesidir.
- Bu aşamada temel UML diyagramlarının çizimine başlanır (Use Case, Activity, Class diagram... vs.)

Tasarım

- Çözümleme aşamasından sonra belirlenen gereksinimlere karşılık verecek yazılım ya da bilgi sisteminin temel yapısının oluşturulması çalışmalarıdır.
- Bu çalışmalar, mantıksal tasarım ve fiziksel tasarım olarak iki gruba ayrılır.
 - **Mantıksal Tasarım:** Mevcut sistem değil önerilen sistemin yapısı anlatılır. Olası örgütsel değişiklikler önerilir.
 - **Fiziksel Tasarım:** Yazılımı içeren bileşenler ve bunların ayrıntıları içerilir.

Gerçekleştirim

- Kodlama
- Test etme
- Kurulum

çalışmalarının yapıldığı aşamadır.

Bakım

- İşletime alınan yazılım ile ilgili olarak, hata giderme ve yeni eklentiler yapma aşamasıdır.
- Bu aşama yazılımın tüm yaşamı boyunca sürer.

Yazılım Yaşam Döngüsü Temel Adımları

- Yazılım yaşam döngüsünün temel adımları çekirdek süreçler (core processes) olarak da adlandırılır.
- Bu süreçlerin gerçekleştirilmesi amacıyla
 - **Belirtim Yöntemleri (Software Specification Methods)** - bir çekirdek sürece ilişkin fonksiyonları yerine getirmek amacıyla kullanılan yöntemler.
 - **Süreç Modelleri (Software Process Models)** - yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlayan modeller kullanılır.

Belirtim Yöntemleri

- **Süreç Akışı İçin Kullanılan Belirtim Yöntemleri**
 - Süreçler arası ilişkilerin ve iletişimin gösterildiği yöntemler (Veri Akış Şemaları, Yapısal Şemalar, Nesne/Sınıf Şemaları).
- **Süreç Tanımlama Yöntemleri**
 - Süreçlerin iç işleyişini göstermek için kullanılan yöntemler (Düz Metin, Algoritma, Karar Tabloları, Karar Ağaçları, Anlatım Dili).
- **Veri Tanımlama Yöntemleri**
 - Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemler (Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü).

Süreç (process) nedir?

- **Süreç** olguların ya da olayların, belli bir taslağa uygun ve belli bir sonuca varacak biçimde düzenlenmesi, *sıralanması*.
- Bir şeyin yapılışını, üretiliş biçimini oluşturan sürekli işlemler, eylemler dizisi (**Kaynak:** <http://tr.wikipedia.org>)
- Aralarında birlik olan veya belli bir düzen veya zaman içinde tekrarlanan, ilerleyen, gelişen olay ve hareketler dizisi, proses

Yazılım süreci nedir?

- Bir *yazılım ürünü*nü üretmeyi sağlayan birbiriyle *tutarlı aktivite grubudur*.



Yazılım süreci nedir?

- Ne yapılmak istendiğini tüm uygulama detaylarına girmeden tanımlar.
- Yazılım süreci bizim yazılım üretme yolumuzdur. *

- *Kaynak: Object-Oriented and Classical SWE, 7thEdition, Stephen R. Schach, p71.

Süreç Modelleri

- **Süreç Modelleri**, Yazılım Yaşam Döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlar.
- Yazılım geliştirmenin bahsedilen zorluklarıyla bahsedebilmek için, geliştirmeyi sistematik hale getirmeyi hedefleyen çeşitli süreç modelleri ortaya çıkmıştır.
 - **Bu modellerin temel hedefi**; proje başarısı için, yazılım geliştirme yaşam döngüsü (“software development life cycle”) boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktır.
- Modellerin ortaya çıkmasında, ilgili dönemin donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır.
- Süreçlerin içsel ayrıntıları ya da süreçler arası ilişkilerle ilgilenmez.
- Özetle yazılım üretim işinin genel yapılaşma düzenine ilişkin rehberler olarak kullanılabilir.
- **Örnek:**
 - Geleneksel modeller (Çağlayan (“waterfall”) , evrimsel, döngüsel ...vb.)
 - Çevik (“Agile”) Modeller (Uçdeğer (“extreme”) programlama modeli – XP)

Süreç Modelleri Neden Önemlidir?

- Endüstri kaliteye önem vermektedir.
 - (örn. performans, üretkenlik)
- Deneyimler göstermektedir ki süreçlerin ürünlerin kalitesine kayda değer etkisi vardır.
 - Ürünlerin istenen kalitede olmasını süreçleri kontrol ederek daha iyi sağlayabiliriz.
- Yönetici ve geliştiricilerin, *yazılım geliştirme sürecinin karışıklığı ile baş etmelerini* sağlarlar .

Süreç Modelleri

Düzenleyici Süreç Modelleri

- Kodla ve Düzelt (Code and Fix)
- Gelişigüzel Model
- Barok Modeli
- Çağlayan/Şelale Modeli (Waterfall Model)
- V Modeli (V-shaped Model)
- Prototipleme
- Helezonik Model (Spiral Model)
- Evrimsel Geliştirme Modeli (Evolutionary Development)
- Artırımsal Geliştirme Modeli (Incremental Development)
- Araştırma Tabanlı Model (Resource Based Model)
- Formal Sistem Geliştirme (Formal System Development)
- Bileşen Tabanlı Geliştirme (Component Based Development)

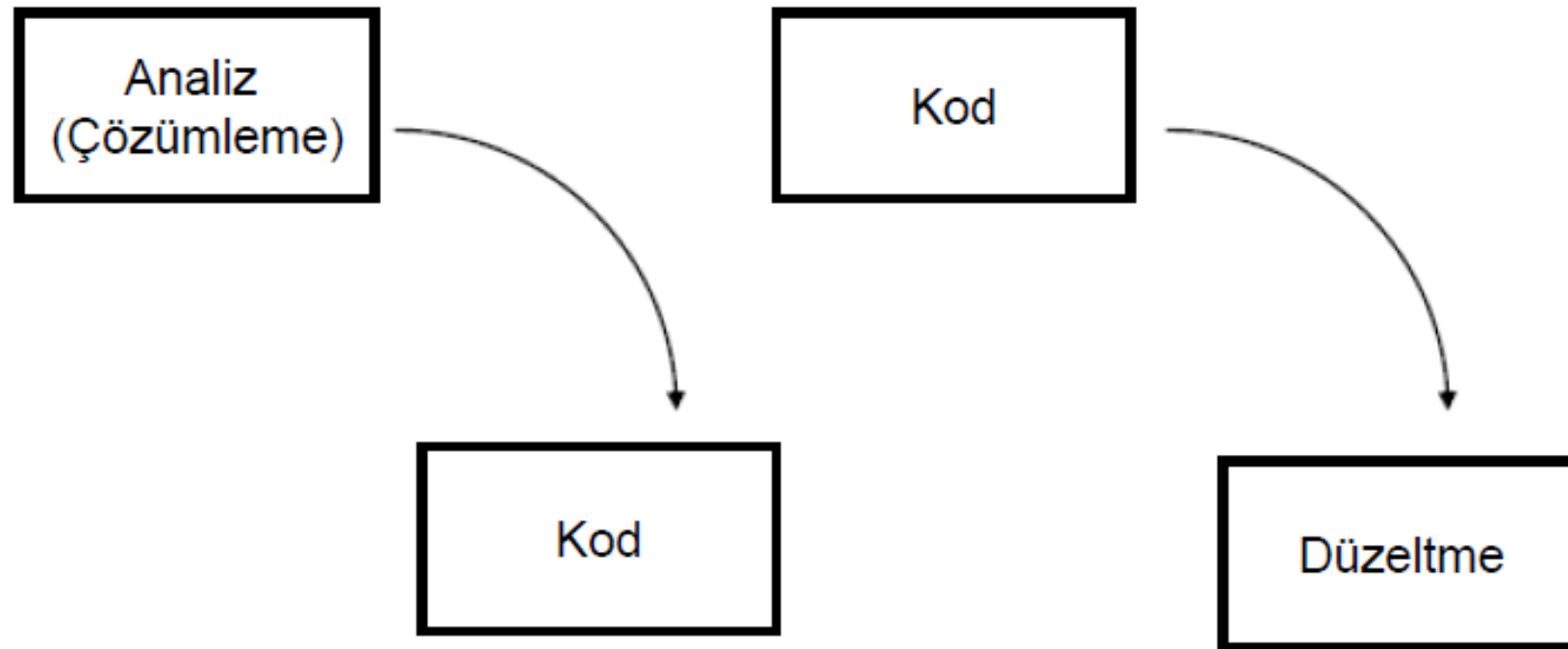
Birleşik Süreç

- Birleşik Sürecin Aşamaları

Çevik Yazılım Süreci

- Uçdeğer Programlama (“Extreme Programming – XP”)
- Scrum
- Özellik Güdümlü Gelistirme (“Feature-Driven Development – FDD”)
- Çevik Tümlesik Süreç (“Agile Unified Process – AUP”)

Kodla ve Düzelt (Code and Fix)



Kodla ve Düzelt - Avantajları

- Tüm gereken yeterli olacak kadar gayrettir.
- Tüm adımlardaki gayret direk olarak ürüne katkı sağladığından çoğu müşteri ödeme yapmaktan mutlu olur.
- Eğer ürün onu yapanlar tarafından kullanılacaksa avantajlıdır.

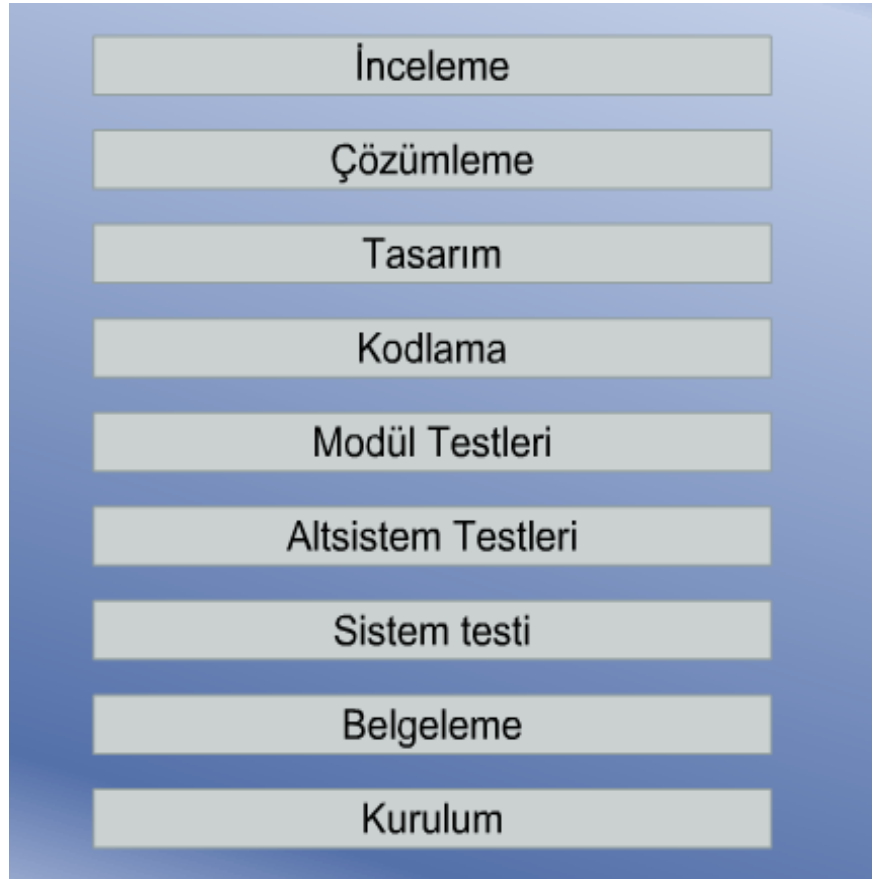
Kodla ve Düzelt - Dezavantajları

- Kodlamaya başlamadan önce değişiklik tahmin edilmediğinden, birbirini izleyen değişikliklerden sonra kod karmakarışık bir hale gelir ve daha sonraki düzeltmeleri yapmak daha da zorlaşır.
- Geliştirilen sistemin boyutunun artması, yapısal olmayan bir şekilde karmaşıklığının yönetilmesini zorlaştırır.
- Müşterinin sürece dahil edilmemesi kullanıcı ihtiyaçlarına uygun olmamasına yol açar.
- Bireysel geliştiriciler için uygundur, takımlar için değil.

Gelişigüzel Model

- Geliştirme ortamında herhangi bir model ya da yöntem kullanılmaz.
- Geliştiren kişiye bağımlı (belli bir süre sonra o kişi bile sistemi anlayamaz ve geliştirme gücünü yaşar).
- İzlenebilirliği ve bakımı oldukça zor.
- 60'lı yıllarda, daha çok tek kişilik üretim ortamlarında kullanılan yöntemlerdir.
- Yani basit programlama yöntemidir..

Barok Modeli



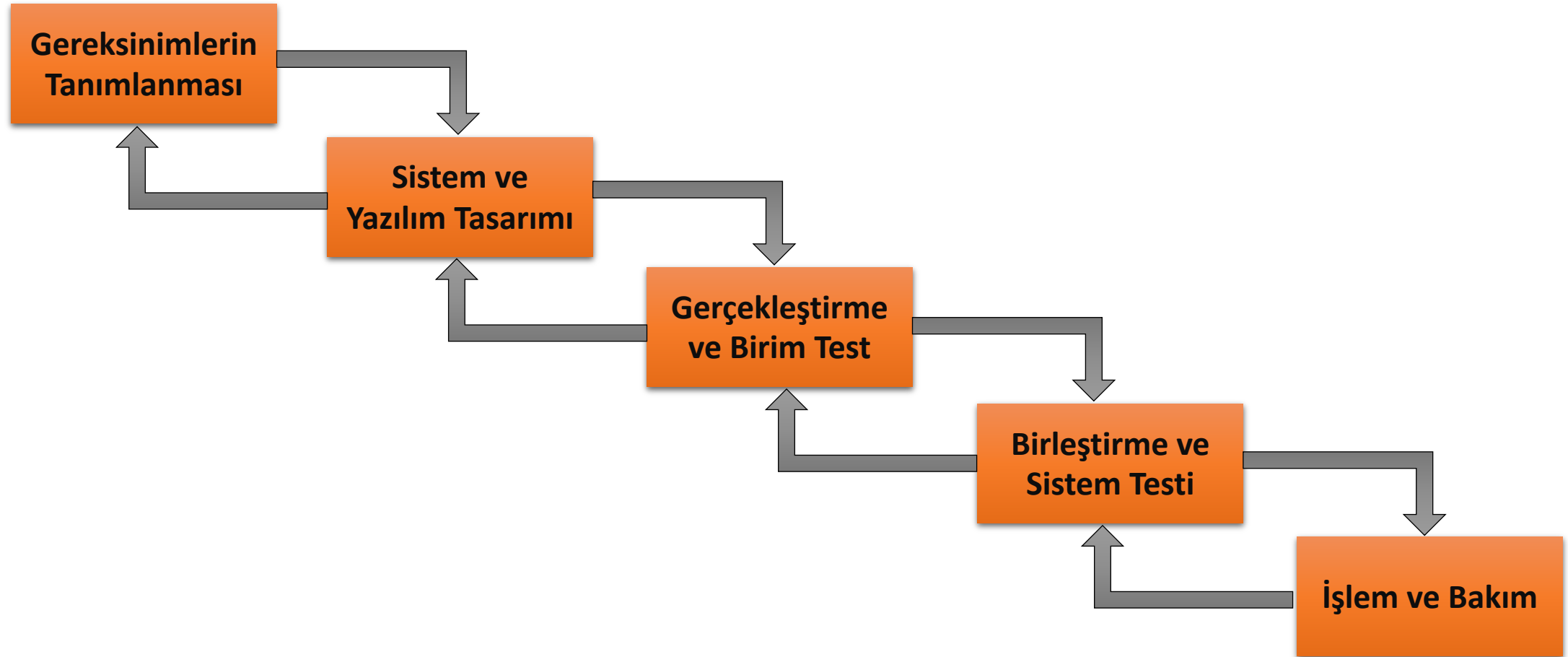
- Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği model.
- Barok modeli 70'li yılların ortalarından başlanarak kullanılmaya başlanmıştır.
- Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.
- Halbuki, günümüzde belgeleme yapılan işin doğal bir ürünü olarak görülmektedir.
- Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değil.

Gerçekleştirim aşamasına daha fazla ağırlık veren bir model olup, günümüzde kullanımı önerilmemektedir.

Çağlayan Modeli

- Yaşam döngüsü temel adımları baştan sona en az bir kez izleyerek gerçekleştirilir.
- İyi tanımlı projeler ve üretimi az zaman gerektiren yazılım projeleri için uygun bir modeldir.
- Geleneksel model olarak da bilinen bu modelin kullanımı günümüzde giderek azalmaktadır.
- Barok modelin aksine belgeleme işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür.
- Barok modele göre geri dönüşler iyi tanımlanmıştır.
- Yazılım tanımlamada belirsizlik yok (ya da az) ise ve yazılım üretimi çok zaman almayacak ise uygun bir süreç modelidir.
- Bir sonraki aşama, önceki aşama tamamlanmadan başlayamaz.
- Her aşamanın sonucu bir ya da birden fazla onaylanan (imzalanan) belgedir.
- Gerekğinde geliştirme aktivitelerinde iterasyonlar (tekrarlamalar) olabilir.

Çağlayan/Şelale Modeli (Waterfall Model)



Çağlayan Modeli - Aşamaları

- **Gereksinim Tanımlama:** Gerçekleştirilecek sistemin gereksinimlerinin belirlenmesi isidir.
 - Müşteri ne istiyor? Ürün ne yapacak, ne işlevsellik gösterecek?
- **Sistem ve Yazılım Tasarımı:** Gereksinimleri belirlenmiş bir sistemin yapısal ve detay tasarımını oluşturma isidir.
 - Ürün, müşterinin beklediği işlevselliği nasıl sağlayacak?
- **Gerçekleştirme ve Birim Test:** Tasarımı yapılmış bir yazılım sisteminin kodlanarak gerçekleştirilmesi isidir.
 - Yazılım ürünü, tasarımı gerçekleştirecek şekilde kodlandı mı?
- **Birleştirme ve Sistem Testi:** Gerçekleştirilmiş sistemin beklenen işlevselliği gösterip göstermediğini sınaama işlemidir.
 - Ürün, müşterinin beklediği işlevselliği sağlıyor mu?
- **İşlem ve Bakım:** Müşteriye teslim edilmiş ürünü, değişen ihtiyaçlara ve ek müşteri taleplerine göre güncelleme isidir.
 - Ürün müşteri tarafından memnuniyetle kullanılabilir mi?

Çağlayan Modeli - Avantajları

- Müşteriler ve son kullanıcılar tarafından da iyi bilenen anlaşılabilen adımlardan oluşur.
- İterasyonlar (tekrarlamalar) bir sonraki ve bir önceki adımlarla gerçekleşir, daha uzak adımlarla olması nadirdir.
- Değişiklik süreci yönetilebilir birimlere bölünmüştür.
- Gereksinim adımı tamamlandıktan sonra sağlam bir temel oluşur.
- Erken işin miktarını arttırır.

Çağlayan Modeli - Avantajları

- Proje yöneticileri için işin dağılımını yapma açısından kolaydır.
- Aşamaları iyi anlaşılabilir.
- Gereksinimleri iyi anlaşılabilen projelerde iyi çalışır.
- Kalite gereksinimlerinin bütçe ve zaman kısıtlamasında göre çok daha önemli olduğu projelerde iyi çalışır.

Çağlayan Modeli Problemleri

Problem - 1

Test aşaması geliştirme sürecinin en sonunda yapılır. Hatalar önemli yeniden tasarım gerekliliğini oluşturur.

Çözüm

1. Çözümleme aşamasının önüne bir ön-tasarım aşaması eklenir böylece programlama kısıtlamaları önceden anlaşılabilir.
2. Her aşamanın sonunda genişletilmiş belgelendirme yapılır

Neden?

Erken aşamalarda tasarım= belgelendirme

Etkili yeniden tasarıma izin verir

Proje ile ortak anlayış

Çağlayan Modeli Problemleri

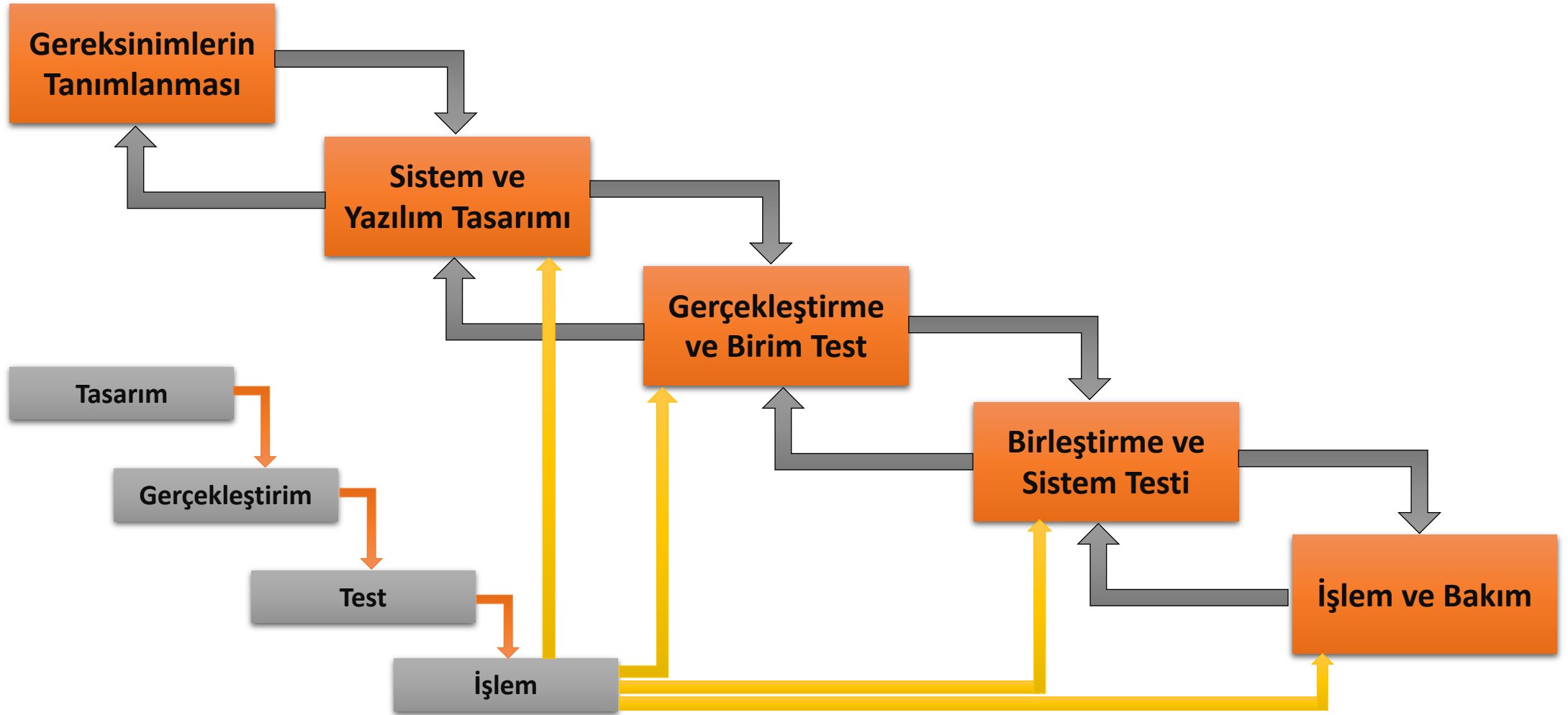
Problem - 2

Eğer ürün tamamıyla orijinal ise, sistemi yapmadan önce biraz deneysel testlerin yapılması gereklidir.

Çözüm

Bazı anahtar hipotezleri sınamak için bir prototip yap.

Prototip yaptıktan sonra



Çağlayan Modeli Problemleri

Problem - 3

Önceden anlaşma sağlansa bile yazılımın ne yapacağı konusu yoruma açıktır.
Kullanıcılar kaliteyi en sondan önce anlayamazlar

Çözüm

Teslim etmeden önce sürece müşteriye de dahil et.
- Gözden geçirmeler

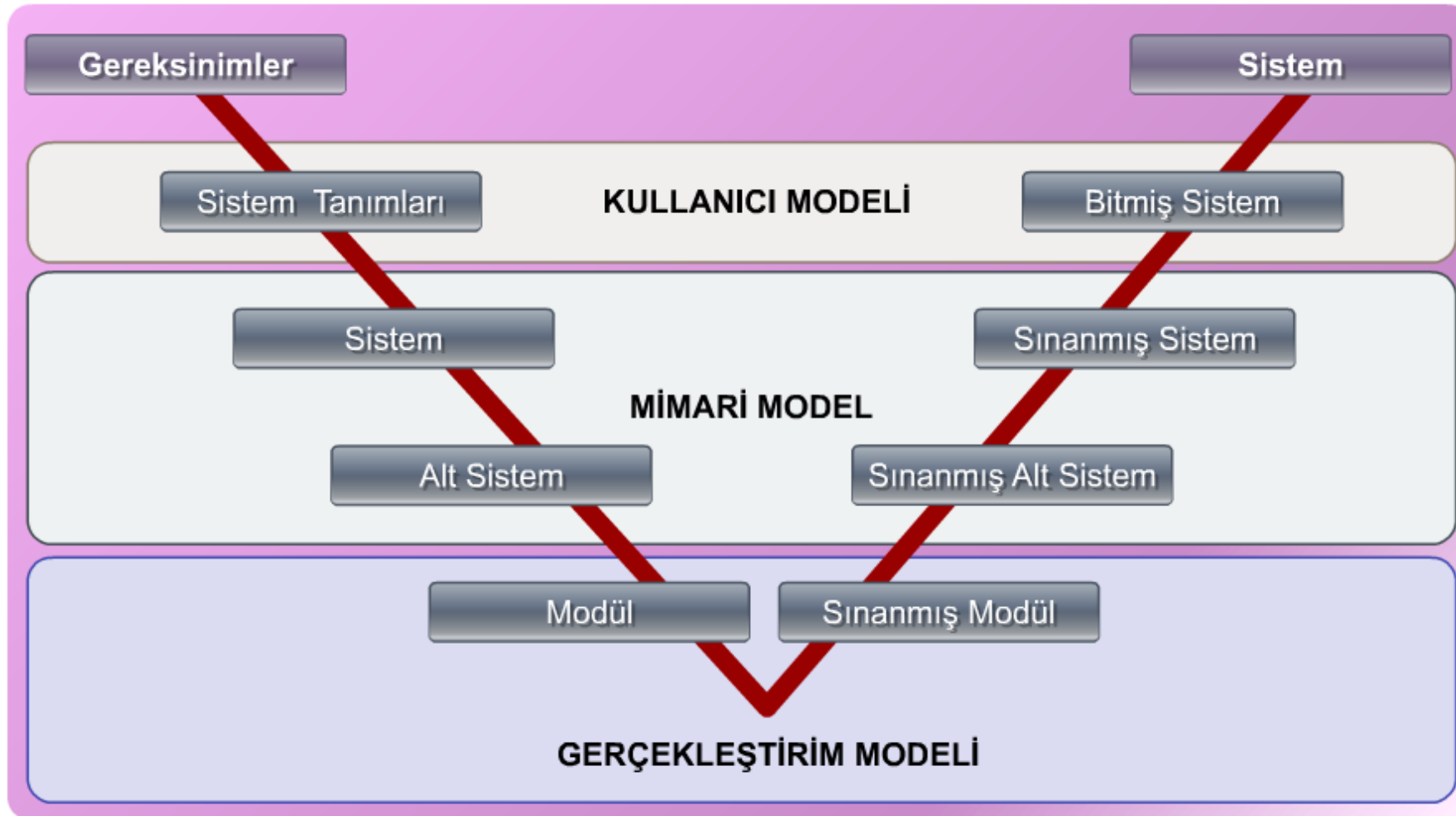
Çağlayan Modeli - Diğer dezavantajları

- Bitirme kriteri olarak belgelendirmeye önem verilmektedir.
 - Bazı alanlar için mümkünken (derleyiciler, işletim sistemleri, vb.) etkileşimli son kullanıcı uygulamaları gibi alanlar için zordur.
- Sistem geliştirilmesi süresince de gereksinimler sıklıkla değişir.
 - Çağlayan modeli gereksinimlerin çok iyi anlaşılabilirdiği durumlarda kullanılmalıdır.
- İki ya da daha önceki fazlara gitmek çok maliyetlidir.
 - bu durumda da gerektiğinde tüm fazı yeniden gerçekleştirmek çok büyük bir iştir.
- Bir faz tamamlanmadan diğerine geçilememesi riski arttırır.

V Modeli (V-shaped Model)

- Proje ve gereksinim planlaması
 - Ürün gereksinimleri ve belirtim çözümlemesi
 - Mimari ve yüksek seviye tasarım
 - Detaylı tasarım
 - Kodlama
 - Birim testi
 - Tümleştirme ve test
 - Sistem ve kabul edilme testleri
- Üretim, işletim ve sürdürülebilirlik

V Modeli



V Modeli

- Sol taraf üretim, sağ taraf sınaama işlemleridir.
- V süreç modelinin temel çıktıları;

Kullanıcı Modeli

Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınaama belirtileri ve planları ortaya çıkarılmaktadır.

Mimari Model

Sistem tasarımı ve oluşacak altsistem ile tüm sistemin sınaama işlemlerine ilişkin işlevler.

Gerçekleştirim Modeli

Yazılım modüllerinin kodlanması ve sınaanmasına ilişkin fonksiyonlar.

V Modeli

- Belirsizliklerin az, **iş tanımlarının belirgin** olduğu BT projeleri için uygun bir modeldir.
- Model, **kullanıcının projeye katkısını** arttırmaktadır.
- BT projesinin **iki aşamalı olarak ihale** edilmesi için oldukça uygundur:
 - İlk ihalede kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta,
 - İkinci ihalede ise ilkinde elde edilmiş olan kullanıcı modeli tasarlanıp, gerçekleştirilmektedir.

V Modeli - Avantajları

- Verification ve validation planları erken aşamalarda vurgulanır.
- Verification ve validation sadece son üründe değil tüm teslim edilebilir ürünlerde uygulanır.
- Proje yönetimi tarafında takibi kolaydır
- Kullanımı kolaydır.

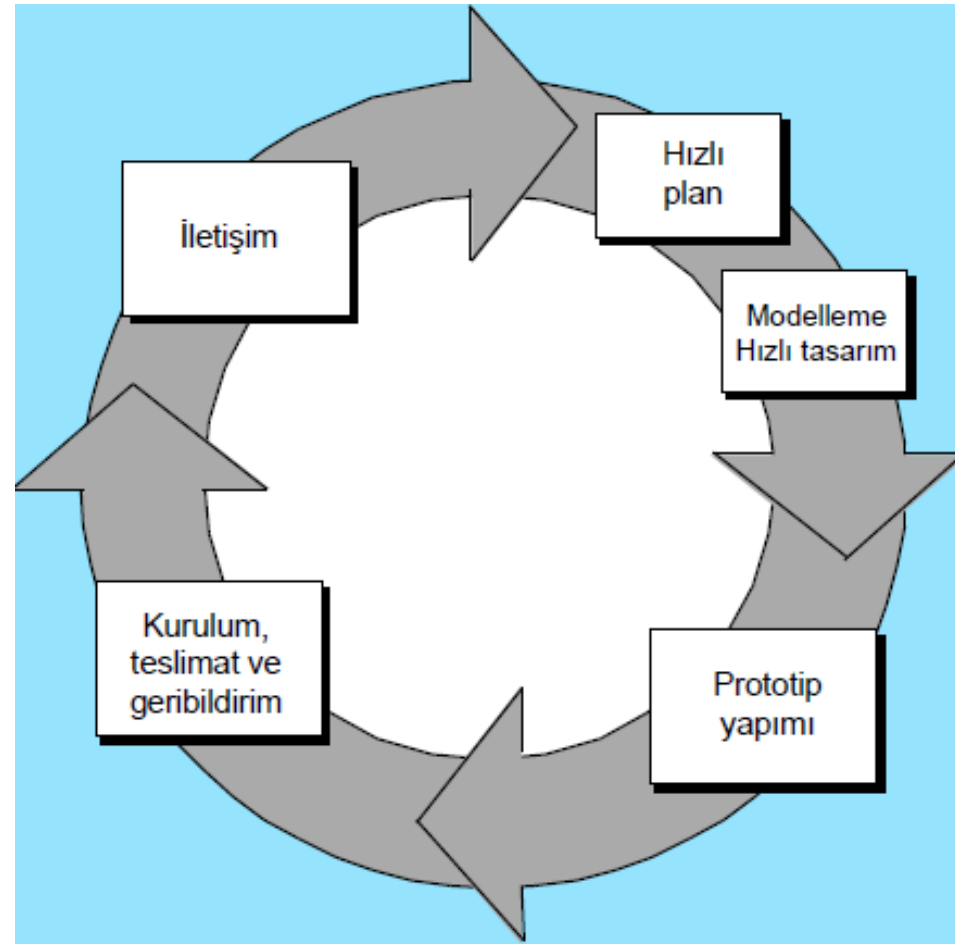
V Modeli - Dezavantajları

- Aynı zamanda gerçekleştirilebilecek olaylara kolay imkan tanımaz.
- Aşamalar arasında tekrarlamaları kullanmaz.
- Risk çözümleme ile ilgili aktiviteleri içermez

Prototipleme

- Gereksinim tanımlama fazında hızlıca yapılan kısmi gerçekleştirme.
- Gereksinimler netleştikçe prototipi düzelt.
- Müşteri memnun olana kadar düzeltmelere devam et.

Prototipleme



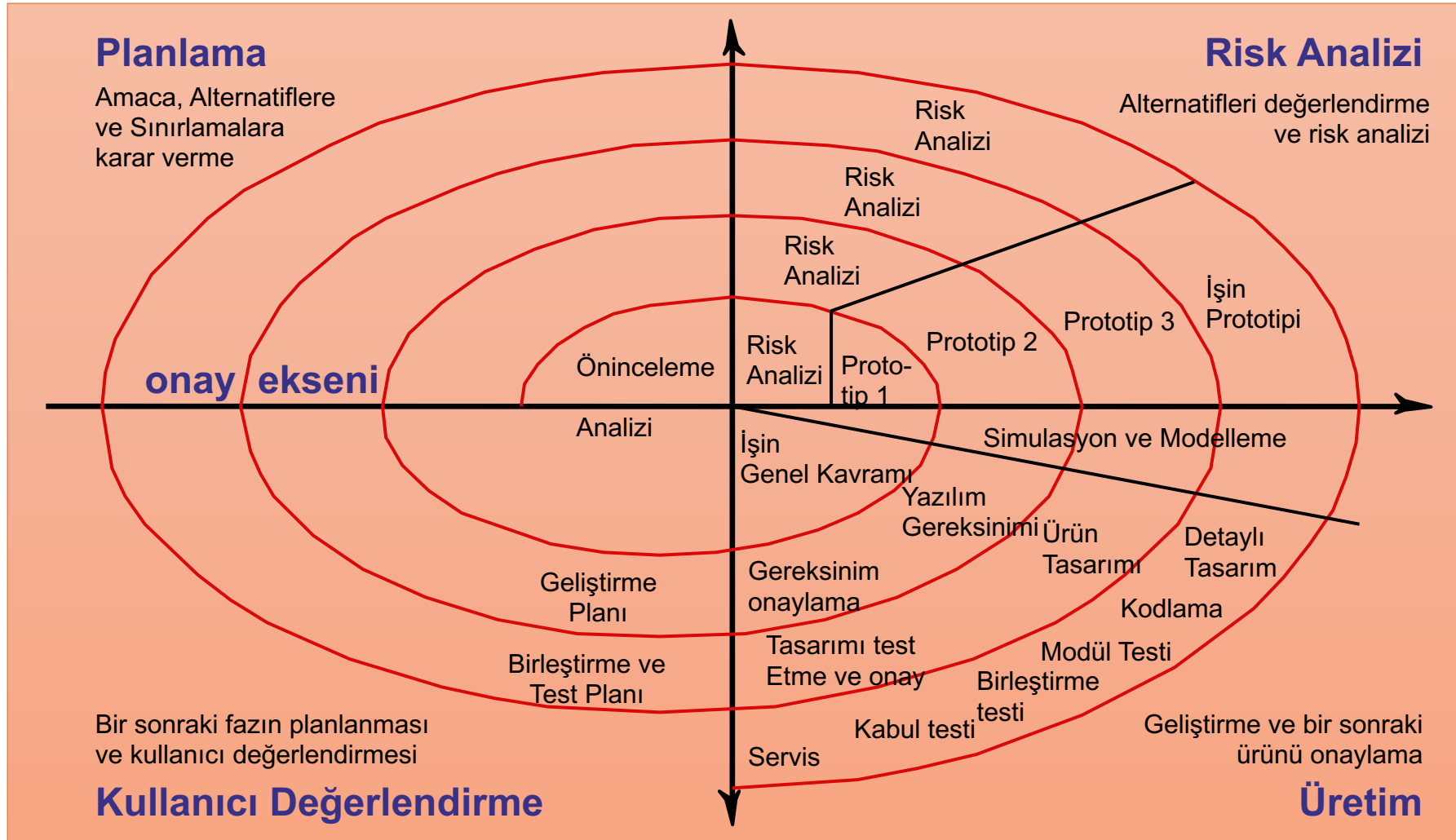
Prototipleme - Avantajları

- Kullanıcı sistem gereksinimlerini görebilir.
- Karmaşa ve yanlış anlaşılımları engeller.
- Yeni ve beklenmeyen gereksinimler netleştirilebilir.
- Risk kontrolü sağlanır.

Prototipleme - Dezavantajları

- Belgelendirmesi olmayan hızlı ve kirli (quick and dirty) prototipler.
- Prototip hedefleri net değilse kod hackleme ya da jenga başlar.
- Düzeltme aşaması atlanırsa, düşük performansa yol açar.
- Müşteri prototipten de son ürün gibi görünüm ve etki bekler.

Helezonik Model (Spiral Model)



Helezonik Model - Aşamaları

- **Planlama**
 - Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme
- **Risk Analizi**
 - Risk seçeneklerinin araştırılması ve risklerin belirlenmesi
- **Üretim**
 - Ara ürünün üretilmesi
- **Kullanıcı Değerlendirmesi**
 - Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sına ve değerlendirmeler

Helezonik Model

- Risk Analizi Olgusu ön plana çıkmıştır.
- Hedefler, alternatifler ve kısıtlamalar belirlenir.
- Alternatifler değerlendirilir, riskler belirlenip çözülür.
- Aşamanın ürünü geliştirilir.
- Sonraki aşama planlanır.
- Her döngü bir aşamayı ifade eder. Doğrudan tanımlama, tasarım,... vs. gibi bir aşama yoktur.
- Yinelemeli artımsal bir yaklaşım vardır.
- Prototip yaklaşımı vardır.

Helezonik Geliştirme

- Süreç arka arkaya devam eden sıralı aktiviteler şeklinde gösterilmek yerine spiral şekilde gösterilir.
- Spiral üzerindeki her bir halka bir fazı gösterir.
- Belirtim, tasarım gibi kesin fazlar yoktur – spiral deki halkalar neye ihtiyaç varsa onu gerçekleştirmek için seçilir.
- Süreç boyunca risklerin değerlendirilmesi ve çözümü açık olarak yapılır.

Helezonik Model - Avantajları

- Kullanıcılar sistemi erken görebilirler.
- Geliştirmeyi küçük parçalara böler . En riskli kısımlar önce gerçekleştirilir.
- Pek çok yazılım modelini içinde bulundurur.
- Riske duyarlı yaklaşımı potansiyel zorlukları engeller.
- Seçeneklere erken dikkate odaklanır.
- Hataları erken gidermeye odaklanır.
- Yazılım-donanım sistemi geliştirme için bir çerçeve sağlar.

Helezonik Model - Problemler

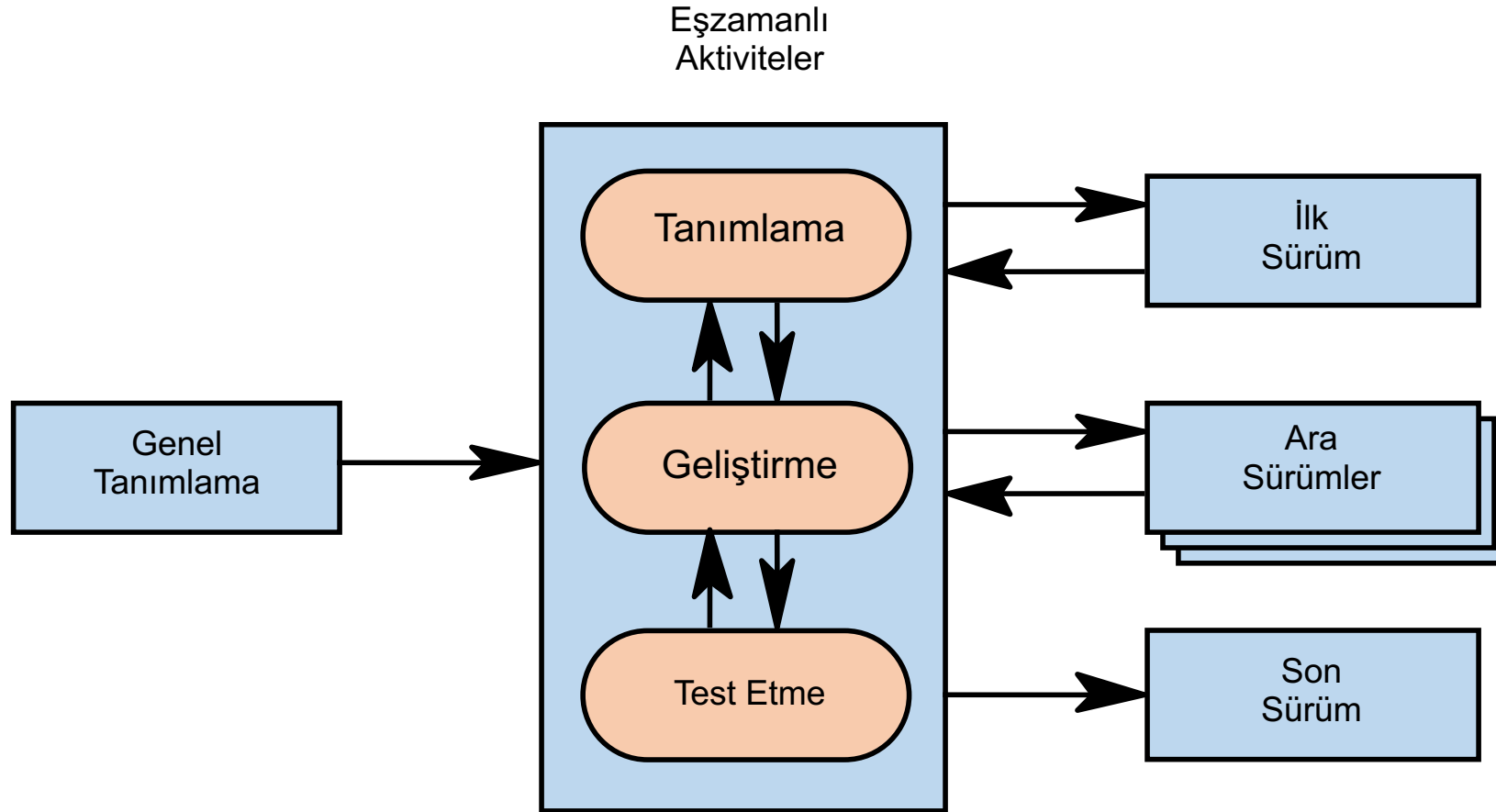
- Küçük ve düşük riskli projeler için pahalı bir yöntemdir.
- Komplekstir (karmaşık).
- Spiral sonsuza gidebilir.
- Ara adımların fazlalığı nedeniyle çok fazla dokümantasyon gerektirir.
- Büyük ölçekte projeler
- Kontrat tabanlı yazılıma uymaz.
 - Yazılımın içten geliştirileceğini varsayar.
 - Kontrat tabanlı yazılımlar adım adım anlaşma esnekliğini sağlamaz.
- Özel risk değerlendirme deneyimine dayanır.
 - Yüksek riskli öğelere yoğunlaşmak, yüksek riskli öğelerin doğru belirlenmesini gerektirir.

Evrimsel Geliştirme Modeli

(Evolutionary Development Model)

- İlk tam ölçekli modeldir.
- Anahtar gereksinimleri ile başlangıç sistemi geliştirilir.
- Müşteri geribildirimi ile sistem pek çok versiyonla yavaş yavaş geliştirilir.
- Belirtim (specification), geliştirme ve geçerleme (validation) aktivitleri koşut zamanlı yürütülür.
- Coğrafi olarak geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir (banka uygulamaları).
- Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedirler.
- Pilot uygulama kullan, test et, güncelle diğer birimlere taşı.
- Modelin başarısı ilk evrimin başarısına bağlıdır.

Evrimsel Geliştirme Modeli



Evrimsel Geliştirme Modeli

- **İki çeşit evrimsel geliştirme vardır:**

- Keşifçi geliştirme (exploratory development)

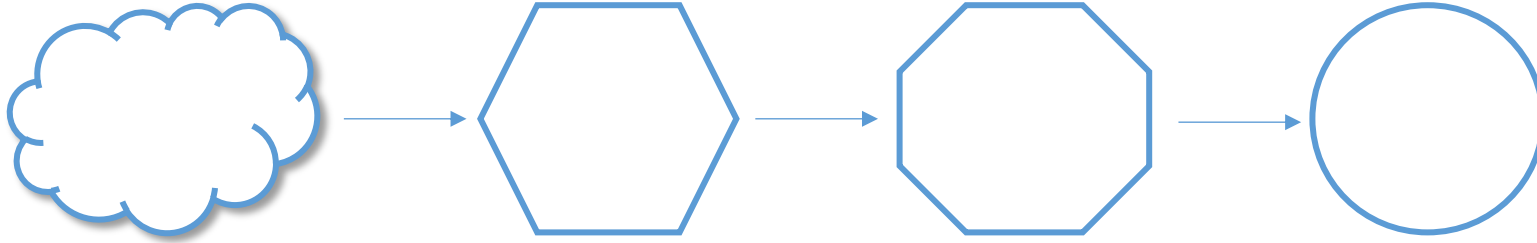
- Hedef: Müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmek
 - İyi anlaşılan gereksinimlerle başlanmalıdır.

“ Ne istediğimi sana söyleyemem ama onu gördüğümde bilirim”

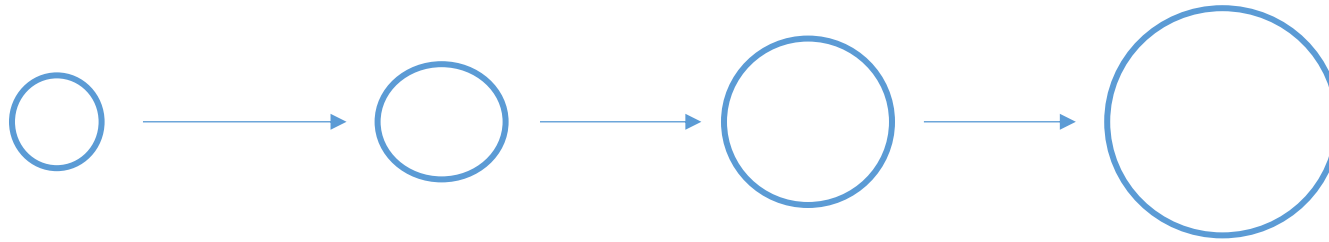
- Atılacak prototipleme (throw-away prototyping)

- Hedef: Sistem gereksinimlerini anlamak
 - Tam anlaşılmamış gereksinimlerle başlar

Karşılaştırma



Çağlayan Modeli



Evrimsel Geliştirme

Zaman

Evrimsel Geliştirme Modeli - Avantajlar

- Kullanıcıların kendi gereksinimlerini daha iyi anlamalarını sağlar.
- Sürekli değerlendirme erken aşamalardaki geliştirme risklerini azaltır.
- Hatalar azalır.

Evrimsel Geliştirme Modeli - Problemler

- Sürecin görünürlüğü azdır (düzenli teslim edilebilir ürün yoktur).
- Sistemler sıklıkla iyi yapılandırılmaz (sürekli değişiklik yazılımın yapısına zarar verir).
- Bakımı zordur.
- Yazılım gereksinimini yenilemek gerekebilir.

Evrimsel Geliştirme - Uygulanabilirliği

- Küçük ve orta boyutlu etkileşimli sistemler (500.000 LOC dan daha az olan).
- Büyük bir sistemin parçaları (ör. Kullanıcı ara yüzleri).
- Kısa süreli kullanılacak sistemler.

Örnek

- Çok birimli banka uygulamaları.
- Önce sistem geliştirilir ve Şube-1'e yüklenir.
- Daha sonra aksaklıklar giderilerek geliştirilen sistem Şube-2'ye yüklenir.
- Daha sonra geliştirilen sistem Şube-3'e,.... yüklenir.
- Belirli aralıklarla eski şubelerdeki güncellemeler yapılır.

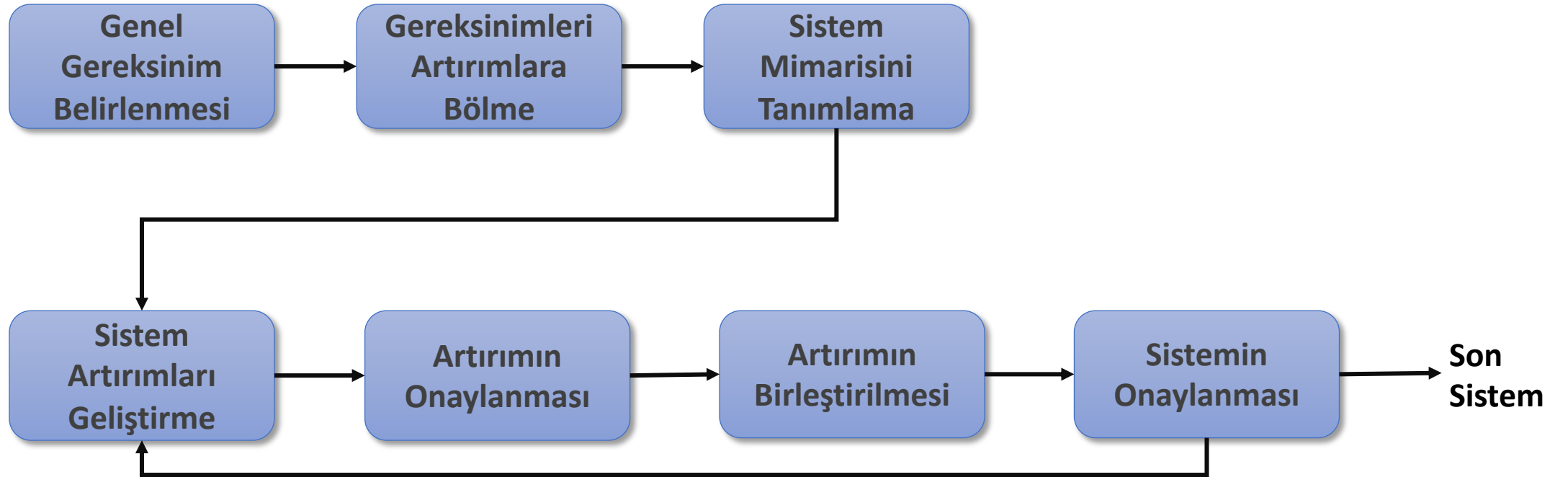
Yazılım Süreç Modellerinde Süreç Tekrarı (“Process Iteration”)

- Yazılım süreç modelleri tek bir defada uygulanmak yerine, birkaç tekrarda uygulanabilir.
 - Örneğin, geniş kapsamlı 5 alt sistemden oluşan bir sistemin; ilk alt sistemi için çağılayan modeli uygulandıktan sonra, geri kalanı için çağılayan modeli tekrar uygulanabilir.
 - Bu şekilde geliştirme riskleri en aza indirilerek ilk tekrarda kazanılan deneyimden, sistemin geri kalanı geliştirilirken faydalanılabilir.
- Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje başında karar verilir.
- Süreç tekrarıyla yakından ilişkili iki geleneksel model vardır:
 - Artırımsal (“incremental”) model
 - Döngüsel (“spiral”) model

Artırımsal Geliştirme Modeli (Incremental Development Model)

- Üretilen **her yazılım sürümü birbirini kapsayacak** ve giderek artan sayıda işlev içerecek şekilde geliştirilir.
- Öğrencilerin bir dönem boyunca geliştirmeleri gereken bir programlama ödevinin 2 haftada bir gelişiminin izlenmesi (bitirme tezleri).
- Uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler bu modele uygun olabilir.
- **Bir taraftan kullanım, diğer taraftan üretim yapılır.**

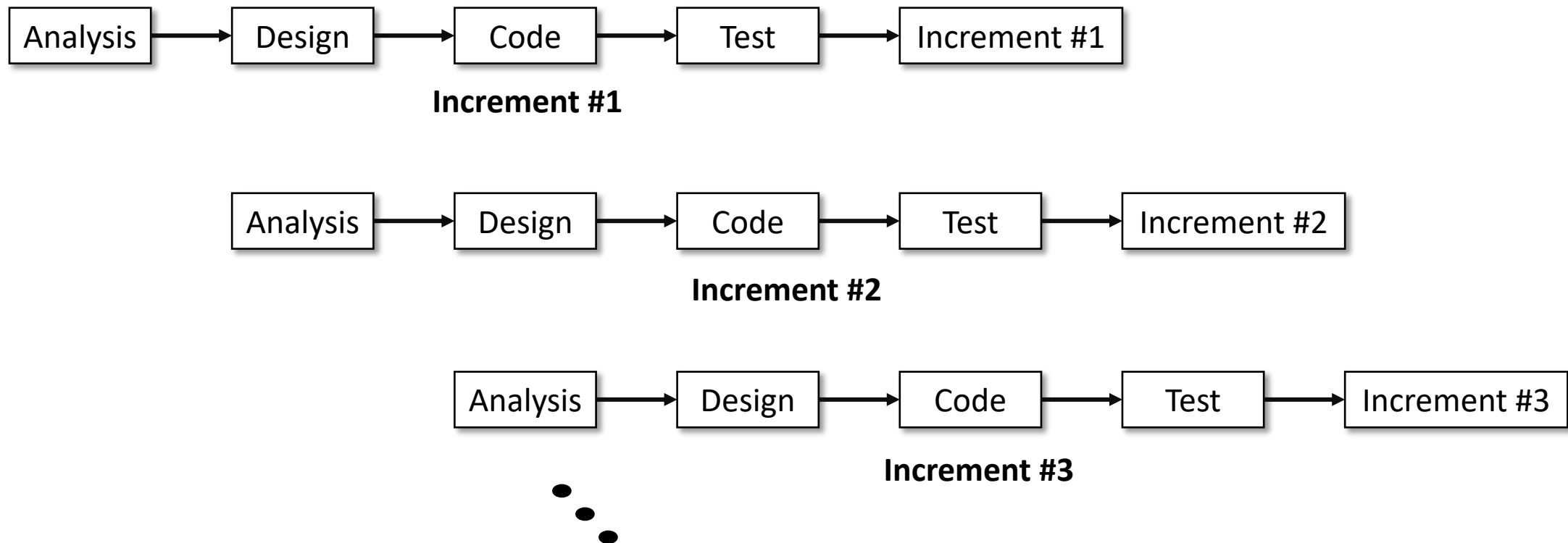
Artırımsal Geliştirme Modeli



Tamamlanmamış Sistem

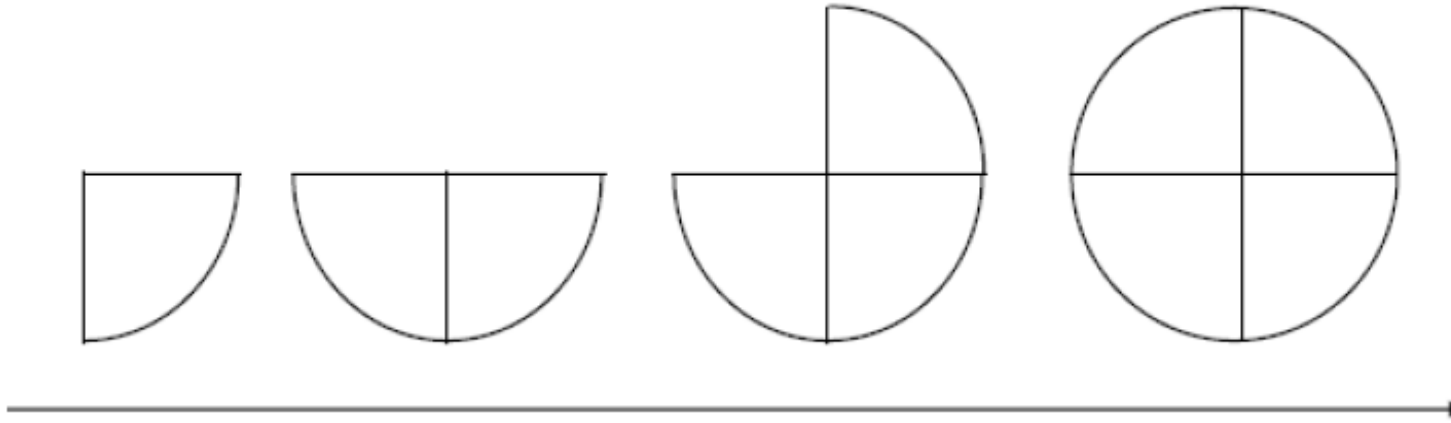
Arttırımsal Geliştirme Modeli

- Aslında Çağlayan modelinin örtüşen şekilde uygulanmasıdır.



Artırımsal Geliştirme Modeli

- Çağlayan modeli ve Evrimsel geliştirme arası bir model:



Artırımsal Geliştirme Modeli - Avantajları

- Sistem için gerekli olan gereksinimler müşterilerle belirlenir
- Gereksinimlerin önemine göre teslim edilecek artımlar belirlenir
- Öncelikle en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir.
- Erken artımlar prototip gibi davranarak, gereksinimlerin daha iyi anlaşılmasını sağlar
- Tüm projenin başarısız olma riskini azaltır
- En önemli sistem özellikleri daha fazla sınanma (test edilme) imkanı bulmuş olur.
- Divide and Conquer (Böl ve Yönet) yaklaşımıdır

Artırımsal Geliştirme Modeli - Dezavantajları

- Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç vardır.
- Gereksinimleri doğru boyuttaki artımlara atamak bazen zor olabilir.
- Deneyimli personel gerektirir.
- Artımların kendi içlerinde tekrarlamalara izin vermez.

Araştırma Tabanlı Model

- **Yap-at** prototipi olarak ta bilinir.
- Araştırma ortamları **bütünüyle belirsizlik** üzerine çalışan ortamlardır.
- Yapılan işlerden edinilecek sonuçlar belirgin değildir.
- Geliştirilen **yazılımlar genellikle sınırlı sayıda kullanılır** ve kullanım bittikten sonra işe yaramaz hale gelir ve atılır.
- Model-zaman-fiya kestirimi olmadığı için **sabit fiyat sözleşmelerinde uygun değildir.**

Örnek

- En Hızlı Çalışan asal sayı test programı!
- En Büyük asal sayıyı bulma programı!
- Satranç programı!

Formal Sistem Geliştirme (Formal System Development)

- Cleanroom yazılım geliştirme
- Matematiksel belirtimin farklı gösterim şekilleri ile çalıştırılabilir programa dönüştürülmesine dayalıdır.
- Formal belirtim, tasarım ve geçerleme kullanarak yazılımda doğruluğun geliştirilmesini vurgular.
- Yazılım artımlarla geliştirilir.
- Sürekli tümleştirme vardır ve fonksiyonellik tümleştirilen yazılım artımları ile artar.
- Felsefesi pahalı hata ayıklama işlemini engellemek için kodu ilk yazarken doğru yazmak ve test aşamasından doğruluğunu sağlamak
 - Formal yöntemler
 - Z dili

Formal Sistem Geliştirme

- **Problemleri**

- Teknikleri uygulayabilmek için eğitim ve özel beceriler gerekmektedir
- Kullanıcı arayüzü gibi sistemin bazı kısımlarını formal olarak belirtmek zordur.

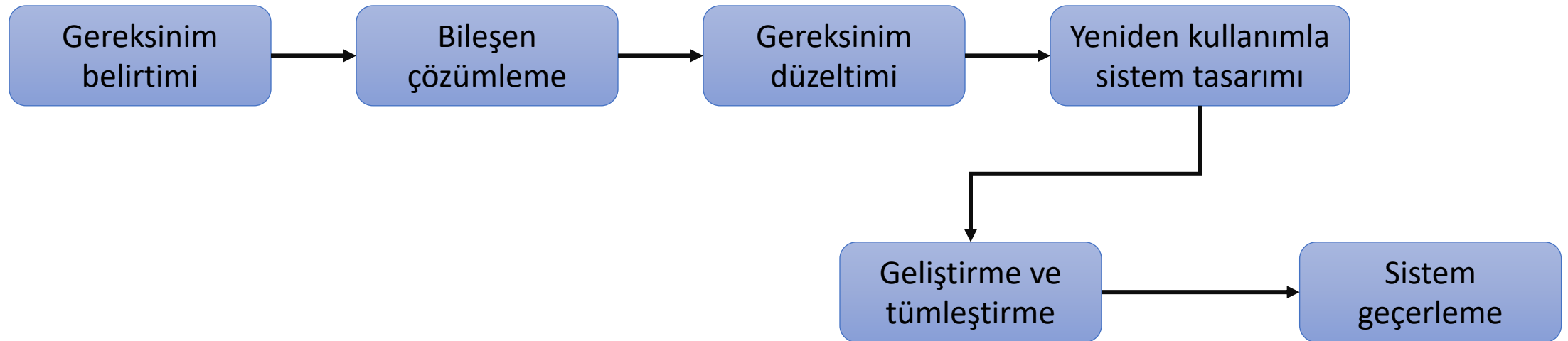
- **Uygulanabilirliği**

- Sistem kullanıma konmadan emniyet ve güvenlik durumlarını sağlanması gereken kritik sistemler.

Bileşen-Tabanlı Model (Component-Based Model)

- Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır.
- **Süreç adımları:**
 - Bileşen analizi
 - Gereksinim günleme
 - Bileşenler kullanarak sistem tasarımı
 - Geliştirme ve tümleştirme
- Bu yaklaşım, bileşen standartlarındaki gelişmeler ilerledikçe daha yaygın olarak kullanılmaya başlanmıştır. ama halen kullanımı limitlidir.

Bileşen-Tabanlı Model - Adımlar



Metodolojiler

- **Metodoloji:** Bir BT projesi ya da yazılım yaşam döngüsü aşamaları boyunca kullanılacak ve birbirleriyle uyumlu yöntemler bütünü.
- Bir metodoloji,
 - bir süreç modelini ve
 - belirli sayıda belirtim yöntemini içerir
- Günümüzdeki metodolojiler genelde Çağlayan ya da Helezonik modeli temel almaktadır

Bir Metodolojide Bulunması Gereken Temel Bileşenler (Özellikler)

- Ayrıntılandırılmış bir süreç modeli
- Ayrıntılı süreç tanımları
- İyi tanımlı üretim yöntemleri
- Süreçlerarası arayüz tanımları
- Ayrıntılı girdi tanımları
- Ayrıntılı çıktı tanımları
- Proje yönetim modeli
- Konfigürasyon yönetim modeli
- Maliyet yönetim modeli
- Kalite yönetim modeli
- Risk yönetim modeli
- Değişiklik yönetim modeli
- Kullanıcı arayüz ve ilişki modeli
- Standartlar

Bir Metodoloji Örneği

- Yourdan Yapısal Sistem Tasarımı Metodolojisi.
- Kolay uygulanabilir bir model olup, günümüzde oldukça yaygın olarak kullanılmaktadır.
- Çağlayan modelini temel almaktadır.
- Bir çok CASE aracı tarafından doğrudan desteklenmektedir.

Yourdon Yapısal Sistem Tasarım Metodolojisi

Aşama	Kullanılan Yöntem ve Araçlar	Ne için Kullanıldığı	Çıktı
Planlama	Veri Akış Şemaları, Süreç Belirtilimleri, Görüşme, Maliyet Kestirim Yöntemi, Proje Yönetim Araçları	Süreç İnceleme Kaynak Kestirimi Proje Yönetimi	Proje Planı
Analiz	Veri Akış Şemaları, Süreç Belirtilimleri, Görüşme, Nesne ilişki şemaları Veri	Süreç Analizi Veri Analizi	Sistem Analiz Raporu
Analizden Tasarıma Geçiş	Akışa Dayalı Analiz, Süreç belirtilimlerinin program tasarım diline dönüştürülmesi, Nesne ilişki şemalarının veri tablosuna dönüştürülmesi	Başlangıç Tasarımı Ayrıntılı Tasarım Başlangıç Veri tasarımı	Başlangıç Tasarım Raporu
Tasarım	Yapısal Şemalar, Program Tasarım Dili, Veri Tabanı Tabloları	Genel Tasarım Ayrıntılı Tasarım Veri Tasarımı	Sistem Tasarım Raporu

Özet

- Yazılım yaşam döngüsü, herhangi bir yazılımın, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır.
- Yazılım Yaşam Döngüsü temel adımları
 - Planlama
 - Çözümleme
 - Tasarım
 - Gerçekleştirim
 - Bakım
- Belirtim Yöntemleri (Software Specification Methods) - bir çekirdek sürece ilişkin fonksiyonları yerine getirmek amacıyla kullanılan yöntemler.
- Süreç Modelleri (Software Process Models) - yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlayan modeller kullanılır.
- **Süreç Modelleri**, Yazılım Yaşam Döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlar.
- Barok Modeli: Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.

Özet

- Çağlayan Modeli Aşamaları:
 - Gereksinim Tanımlama
 - Sistem ve Yazılım Tasarımı
 - Gerçekleştirme ve Birim Test
 - Birleştirme ve Sistem Testi
 - İşlem ve Bakım
- V süreç modelinin temel çıktıları;
 - Kullanıcı Model
 - Mimari Model
 - Gerçekleştirim Modeli
- Helezonik Model Aşamaları:
 - Planlama
 - Risk Analizi
 - Üretim
 - Kullanıcı Değerlendirmesi
- İki çeşit evrimsel geliştirme vardır:
 - Keşifçi geliştirme (exploratory development)
 - Atılacak prototipleme (throw-away prototyping)

Özet

- Artırımsal Geliştirme Modeli-Öğrencilerin bir dönem boyunca geliştirmeleri gereken bir programlama ödevinin 2 haftada bir gelişiminin izlenmesi (bitirme tezleri).
- Formal Sistem Geliştirme: Matematiksel belirtimin farklı gösterim şekilleri ile çalıştırılabilir programa dönüştürülmesine dayalıdır.
- Bileşen-Tabanlı Model : Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır.
- Bileşen-Tabanlı Model – Adımlar
 1. Gereksinim belirtimi
 2. Bileşen çözümleme
 3. Gereksinim düzeltimi
 4. Yeniden kullanımla sistem tasarımı
 5. Geliştirme ve tümleştirme
 6. Sistem geçerleme

Sorular

1. Yazılım yaşam döngüsünün temel adımlarını açıklayınız.
2. Süreç modelleri ve belirtim yöntemlerinin önemini belirtiniz.
3. Süreç modelleri ile belirtim yöntemleri arasındaki farklılıkları belirtiniz.
4. Barok modeli tanımlayınız, yararlarını ve aksak yönlerini belirtiniz.
5. Çağlayan modeli tanımlayınız, yararlarını ve aksak yönlerini belirtiniz.
6. Helezonik modeli tanımlayınız, ayırıcı özelliklerini belirtiniz. Yararlarını ve aksak yönlerini açıklayınız.
7. V model kullanılarak geliştirilecek örnek bir proje tanımı yapınız.
8. VP süreç modeli ile geliştirilecek bir projede uygulanabilecek üç prototipleme örneği veriniz.
9. Evrimsel Geliştirme süreç modelinde Konfigürasyon yönetimi ve değişiklik denetimi neden sorundur?
10. Artımsal geliştirme süreç modelini tanımlayınız, yararlı ve aksak yönlerini belirtiniz.
11. Artımsal geliştirme süreç modeli kullanılarak geliştirilecek bir proje örneği veriniz. Gerekçenizi açıklayınız.
12. Araştırma tabanlı süreç modeli için uygun proje örnekleri veriniz.
13. Metodolojiyi tanımlayınız. Bildiğiniz metodoloji örneklerini listeleyiniz.
14. Yourdon Yapısal Sistem Geliştirme Metodolojisini tamamlayınız.
15. Süreç modelleri, belirtim yöntemleri ile metodolojiler arasındaki ilişkiyi belirtiniz.

Ödevler

- Çevik Yazılım Hakkında Araştırma Yapınız.
- Çevik Yazılım Geliştirmenin projelerdeki başarısı hakkında edindiğiniz bilgileri rapor haline getirin.

Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- “Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- “Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN
- <http://blog.alisuleymantopuz.com/2014/08/30/yazilim-mimarisi-ve-tasarimi-nedir/>
- <http://www.akifsahman.com/?p=175>
- <https://ece.uwaterloo.ca/~se464/08ST/index.php?src=lecture>
- <http://info.psu.edu.sa/psu/cis/azarrad/se505.htm>
- <http://www.metinakbulut.com/YAZILIM-MIMARISI/>
- http://ceng.gazi.edu.tr/~hkaracan/source/YPY_H3.pdf
- <http://iiscs.wssu.edu/drupal/node/3399>
- <http://www.cs.toronto.edu/~sme/CSC340F/slides/21-architecture.pdf>
- <http://www.users.abo.fi/lpetre/SA10/>
- <http://sulc3.com/model.html>
- <http://salyangoz.com.tr/blog/2013/11/23/digerleri/yazilim-gelistirme-surec-modelleri-3/>



YMH311 Yazılım Tasarım ve Mimarisi

Birleşik Süreç ve Çevik (Agile) Yazılım Süreç Modelleri

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Birleşik Süreç ('Unified Process').....4

Çevik ('Agile') Yazılım Süreç Modelleri.....10

Scrum Süreci40

Yazılım Süreçleri – IEEE/IEA 12207.....44

Amaçlar



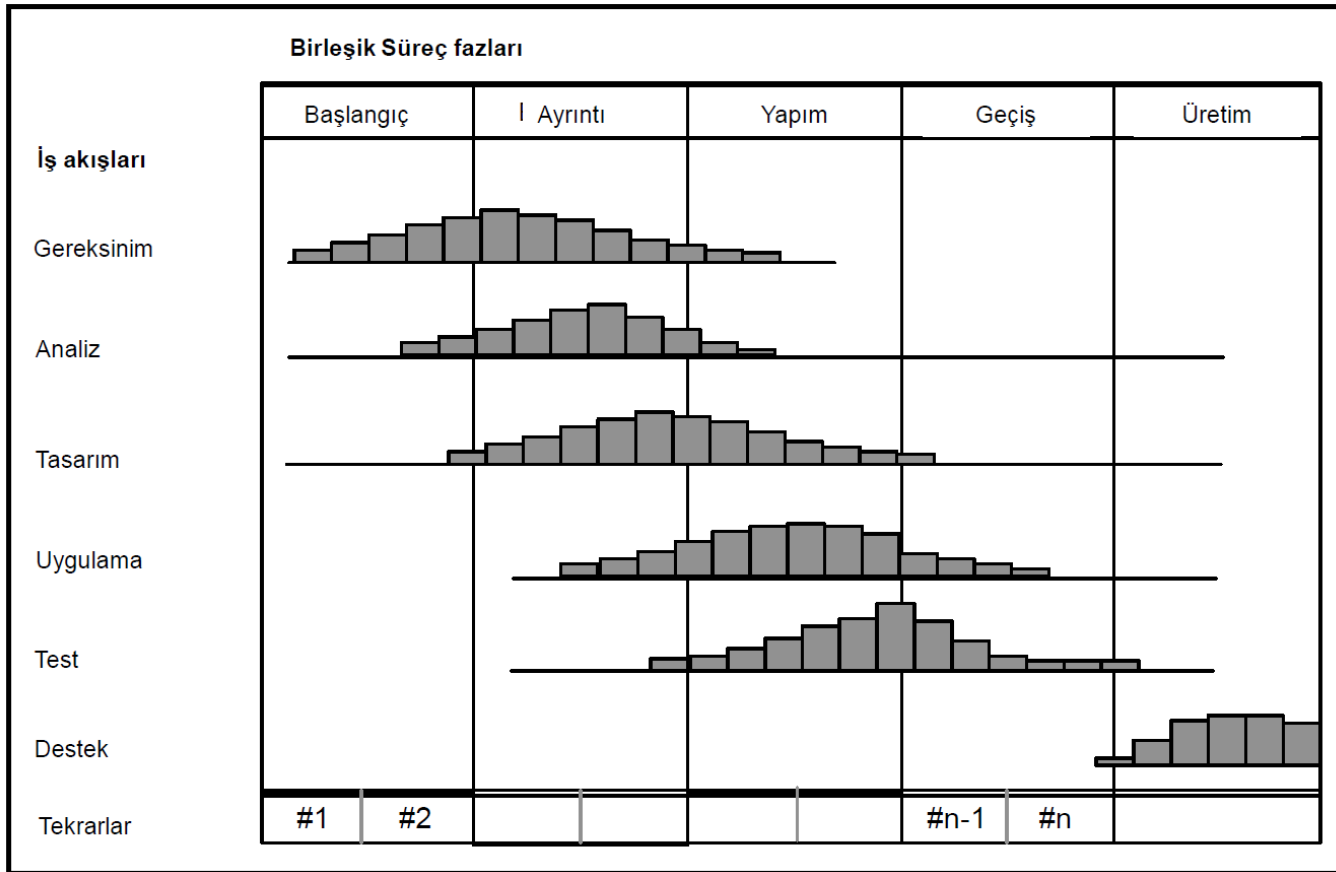
- Birleşik Süreç ve Fazlarını kavramak
- Çevik (Agile) yaklaşım.
- Çevik (Agile) Yazılım Süreç Modellerini Anlamak
- Çevik (Agile) Yazılım Süreçlerinin diğer süreçlere göre üstünlükleri.
- Çevik (Agile) Yazılım Süreç Modellerinin kullanım amaçları.
- Yazılım Süreç Modelleri seçimi.
- Yazılım Süreçleri – IEEE/IEA 12207 kavramı.

Birleşik Süreç (Unified Process)

➤ Nesneye dayalı yazılım geliştirmek için var olan yöntemlerin deneyimler sonucu kabul gören en iyi özellikleri bir araya getirilerek tümleştirilmiş yazılım geliştirme süreci (The Unified Process - UP) oluşturulmuştur.

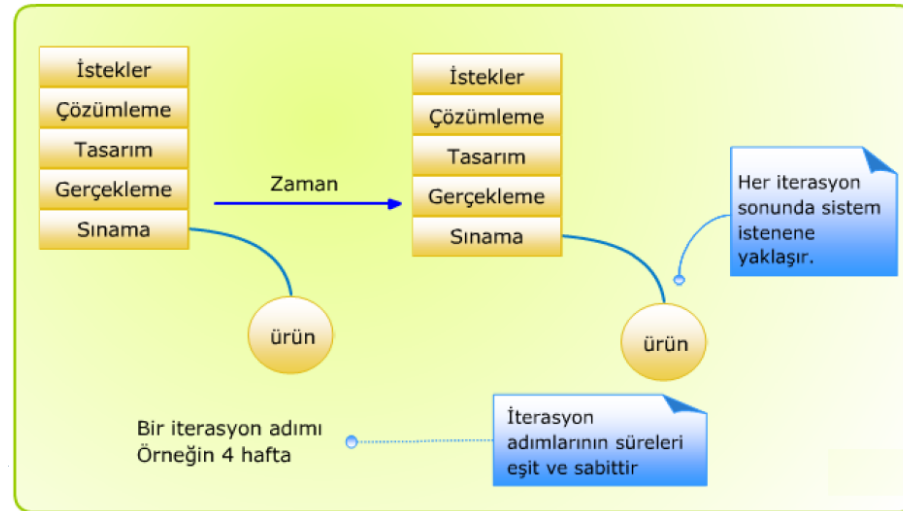
- **Yinelemeli (iterative):** Problemdaki istekler (requirements) bir bütün olarak yerine getirilmeye çalışılmaz. Önce problemin bir kısmı ele alınır. Problemin bu kısmı bağımsız bir proje olarak ele alınır ve hedeflenen istekleri yerine getiren sınanmış tam bir ürün ortaya çıkartılır. Ardından bir sonraki yinelemeye geçilir ve yeni istekler ele alınır. Her iterasyon sonunda hedefe daha yakın bir ürün elde edilir.
- **Arttırmalı ve evrimsel (incremental, evolutionary):** Her iterasyon adımıda yeni istekler ele alındığında iterasyonlar sonucunda elde edilen ürünlerin özellikleri artar ve hedeflenen yazılım ürününe yaklaşırlar.
- **Risk güdümlü (Risk-driven):** ilk iterasyonlarda en riskli kısımlar gerçekleşmelidir. Böylece daha projenin ilk aşamalarında ortaya çıkabilecek problemler görülebilir ve gerekli önlemler alınabilir. Örneğin zaman planı gözden geçirilir, ekibe yeni elamanlar alınabilir, bütçe güncellenebilir.

Birleşik Süreç Fazları



UP: Yinelemeli ve evrimsel yazılım geliştirme

Her yineleme (iterasyon) adımında bütün bir yazılım projesi varmış gibi davranılır. Gerekli tüm aşamalardan geçilerek sınanmış, çalışır bir ürün elde edilir.



Yinelemeli Sürecin Yararları

- **Değişen isteklere uyum:** Yazılım geliştirmedeki en büyük problemlerden biri isteklerin değişmesidir. Yinelemeli çalışma sayesinde istekler değişirse bir sonraki yineleme adımıyla değişikliklere uyum sağlanabilir.
- **Erken geri besleme:** Her iterasyondan sonra müşteriden (yazılımın kullanıcısı) geri besleme alınarak beklentilerin ne ölçüde karşılandığı sınanmış olur. Eğer bir uyumsuzluk varsa erkenden önlem alınır.
- **Büyük sistemlerde çözümleme kolaylığı:** Büyük ve karmaşık bir sistemi bir bütün olarak tek adımda çözümlemek (anlamak) zor olacaktır. Bu nedenle istekleri parça parça ele almak sistemi anlamak ve tasarlamak açısından kolaylık sağlar.
- **Her iterasyonda deneyim kazanılması:** Her yineleme adında yazılım ekibi o konu ile ilgili deneyim kazanır ve elemanlar birbirini daha iyi tanır.
- **Risklerin erken giderilmesi (eğer mümkünse):** Riskli kısımlar ilk iterasyonlarda ele alındığından eğer bir problem varsa bu problemlere karşı önlemler mümkün olduğu kadarıyla erken alınmış olur.
- **Erken ürün elde etme, takımda moral yükselmesi:** Her iterasyon sonunda bir ürün elde edildiğinden zaman içinde hedefe yaklaşıldığı görülür ve yazılım ekibi çalışmalarının sonuçlarını görerek moral kazanır.

UP'nin Kullanılmasına Yönelik Öneriler

2 - 6 haftalık sabit süreli iterasyonlar uygulanmalı

- Deneyemlere göre bir iterasyonun süresinin 3 ya da 4 hafta olarak seçilmesi iyi sonuçlar vermektedir.
- İterasyon 2 haftadan kısa olursa bu sürede bir ürün çıkarmak mümkün olmaz.
- Altı haftadan daha uzun süreli iterasyonlarda ise erken geri besleme alma olanağı ortadan kalkar ve çok fazla sayıda istek ile uğraşmak gerekir.

Yüksek risk taşıyan kısımlar ilk iterasyonlarda gerçekleşmeli

- Daha önce de açıklandığı gibi ortaya çıkabilecek problemleri mümkün olduğu kadar erken fark edip bunlara karşı önlem alabilmek için zorlu görünen istekler önce ele alınmalı.

Temel oluşturan yapılar (çekirdek) önce gerçekleşmeli

- Yüksek riskli kısımlardan başka önce ele alınması gereken modüller sistemin temelini (iskeletini) oluşturan yapılardır.

UP'nin Kullanılmasına Yönelik Öneriler

- Sürekli kullanıcılardan geri besleme alınmalı, isteklere uyulmaya dikkat edilmeli
- Her iterasyondan sonra ürün tam olarak sınanmalı
 - Her iterasyonda bir ürün oluşturulduğu unutulmamalı ve bu ürün tam olarak sınanmalıdır.
 - Aksi durumda hatalar geç fark edilir ve bunları düzeltme maliyeti yüksek olur.
- Kullanım senaryoları yöntemi (use case) uygulanmalı
- Görsel modelleme (UML) kullanılmalı
 - UML tasarımların ifade edilmesini ve takım elemanları arasında iletişimi kolaylaştırır.
- Bir iterasyonda elde edilen deneyim diğer iterasyonda kullanılmalı

Çevik (Agile) Yaklaşım

Bir yazılım projesi, 6 ay içerisinde tamamlanmazsa ve projeye müşterinizi dahil etmezseniz, başarıya ulaşma ihtimaliniz zayıftır.



Çevik (Agile) Yaklaşım



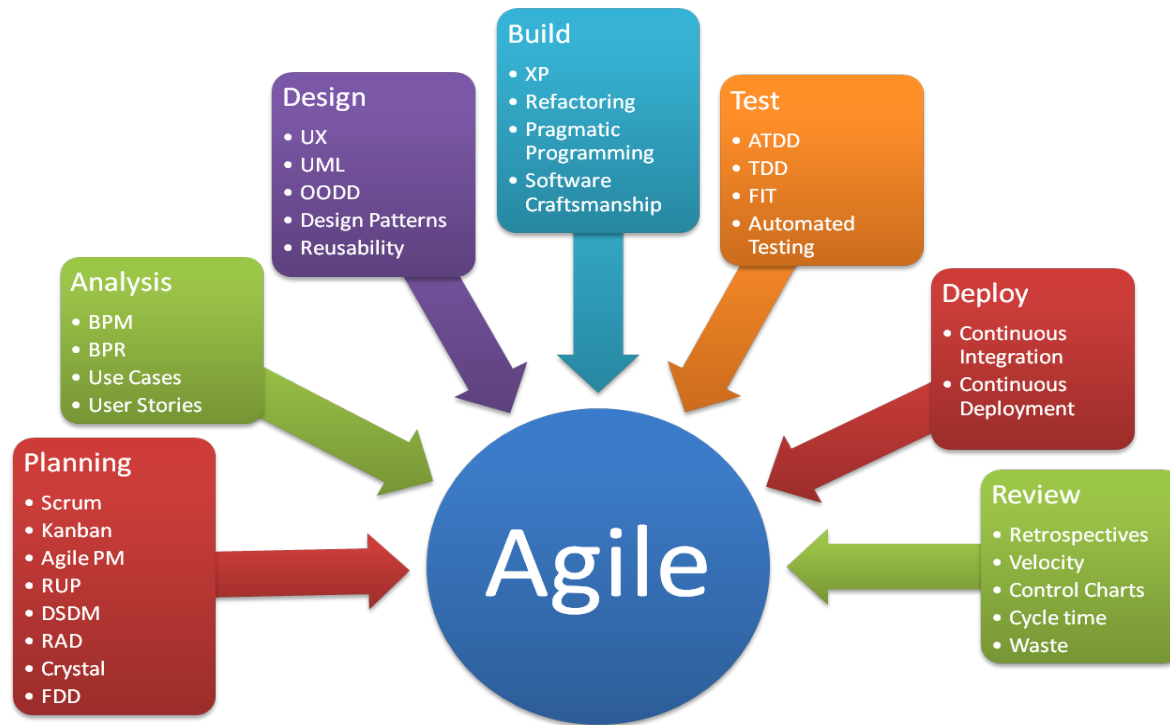
Çevik (Agile) Modelleme

- Scott Ambler tarafından önerilmiştir.
- Çevik model prensiplerinin bir kümesini önerir.
 - Bir amaç ile modelle.
 - Çoklu model kullan.
 - Travel light.
 - İçerik gösterimden daha önemlidir.
 - İçeriği oluşturmada kullandığın model ve araçları bilmek.
 - Adapt locally

Çevik (Agile) Yazılım Süreç Modelleri

- Çevik modeller, mevcut geleneksel modellere alternatif olarak, 1990'larda ortaya çıkmaya başlamıştır.
 - **Çevik:** Kolaylık ve çabuklukla davranan, tetik, atik. [TDK]
 - 1950'lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımların, yazılım sektöründe bir uzantısı olarak ortaya çıkmıştır.
- Çevik modeller kapsamında; yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve belgelendirmeye yönelik, pratiğe dayalı yöntemler yer alır.
- Bu modelleme biçiminin; kapsadığı değerler, prensipler ve pratikler sayesinde geleneksel modellemelere metotlarına göre yazılımlara daha esnek ve kullanışlı biçimde uygulanabileceği savunulmaktadır.

Çevik (Agile) Yazılım Süreç Modelleri



Çevik Yazılım Geliştirme Manifestosu

➤ 2001 yılında, dünyanın önde gelen çevik modellerinin temsilcileri, ortak bir zeminde buluşabilmek adına bir araya gelerek “çevik yazılım geliştirme manifestosu” nu yayınlamışlardır. Bu manifestoya göre;

- Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden;
- Çalışan yazılım, detaylı belgelerden;
- Müşteri ile işbirliği, sözleşmedeki kesin kurallardan;
- Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten;

daha önemli ve önceliklidir.

(Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas)

Çevik Yazılım Geliştirme Manifestosu- Prensipler

- En önemli önceliğimiz müşteriye değerli yazılımı erken zamanda ve sürekli bir şekilde sunarak memnun etmektir.
- Geliştirme sürecinde son zamanlarda da gelse değişen gereksinimler hoş karşılanmalıdır. Çevik süreçler değişimi müşterinin rekabetçi avantajı için kullanırlar.
- Çalışan yazılım bir iki haftadan bir iki aya kadar, mümkün olan en kısa sürelerde sunulmalıdır.
- Geliştiriciler ve iş insanları proje boyunca beraber çalışmalıdırlar.

Çevik Yazılım Geliştirme Manifestosu- Prensipler

- Projeleri motivasyonu yüksek bireylerin çevresinde geliştirilmelidir. Onlara ihtiyaçları olan destek ve ortamı sağlayın ve işi yapabileceklerine güvenin.
- Bilginin takımlar içerisinde ya da takımlar arasında en iyi paylaşılması ve anlaşılmasının yolu yüz yüze iletişimidir.
- İlerlemenin en iyi göstergesi çalışan yazılımdır.
- Çevik süreçler süründürülebilir geliştirme sağlar. Sponsorlar, geliştiriciler ve kullanıcılar sabit hızlarını sonsuz şekilde koruyabilirler

Çevik Yazılım Geliştirme Manifestosu- Prensipler

- Teknik mükemmellik ve iyi tasarıma verilen sürekli önem çevikliği arttırır.
- Basitlik önemlidir. (Yapılmaması gereken tüm işleri önlemek).
- En iyi mimari, gereksinimler, ve tasarımlar kendi kendilerine organize olabilen takımlardan çıkar.
- Düzenli aralıklarla, takımlar nasıl daha etkili olacaklarına bakar, davranışlarını düzenler ve ona göre hareket eder.

Çevik Modellemenin Başlıca Özelliği

- Veri modelleri ve ara yüzü modelleri gibi modelleme tekniklerinin neler olduğunu ve bunların ayrıntılarını söylemek yerine bu tekniklerin nasıl uygulanması gerektiğini söylemesidir.
- Mesela; yapılan projelerin test edilmesi gerektiğini belirtse bile nasıl test hazırlanacağına değinmemesi gibi.
- Bu sadece bir yöntemler biçimidir ve bir projenin etkili, hızlı bir şekilde ortaya çıkarılmasında, müşterinin ihtiyaçlarını karşılamasında ve aynı zamanda da her türlü değişikliğe kolayca adapte olabilmesinde geliştiricilere yol gösterir.

Hangi Durumlarda Kullanılabilir?

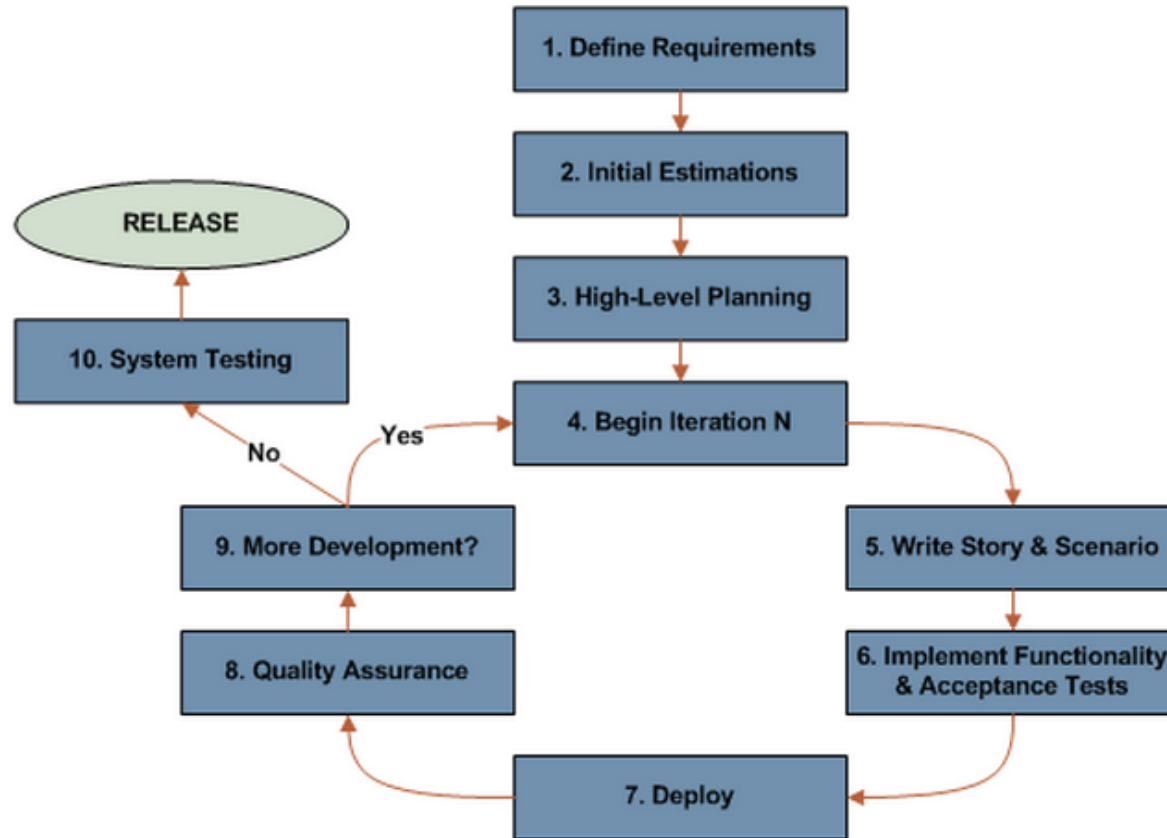
Bu metotların kullanılmasının en uygun olduğu durumlar şunlardır:

- Projenin yazılım evresinde müşteriden gelebilecek talep değişikliklerinin tahmin edilemez olması.
- Projenin parçalarının önce tasarlanıp ardından hemen geliştirilmesinin gerekmesi ve önceden ne yapılacağını, detaylı yol haritasını ve tasarımını tahmin etmenin çok güç olması.
- Analiz, tasarım ve test etme süreçlerinin ne kadar zaman alacağını önceden bilinmemesi.
- Yazılım ekibinin birlikte çalışmak, hiyerarşiye önem vermemek, sağlam iletişim kurmak gibi özelliklere sahip olması.

Çözüm



Çevik Yazılım Geliştirme Süreci



Çevik Yazılım Modelinin Diğer Modellerden Farkı

- Çevik modeller bazen planlı ve disiplinli olmaması ile eleştirilse de bu doğru değildir.
- Açıklamak gerekirse; bir projenin ortaya çıkarılmasında 2 tip model kullanılabilir. **Biri uyarlanabilir olan, bir diğeri de tahmin edilebilir** olandır.
- Değişime tepki verebilecek şekilde tasarlanmıştır. Mesela, projenin gereksinimleri değişirse, projenin takımı da bu değişikliğe ayak uydurur ve değişir.
- Bu takım bize 1 hafta sonra hangi işlerin yapılacağını söyleyebilir ancak 1 ay sonra ne yapacaklarını belirtemez.

Çevik Yazılım Modelinin Diğer Modellerden Farkı

- Yani projede planlanan tarih ne kadar uzaksa o tarihte yapılacak işler de bir o kadar belirsizdir.
- Tahmin edilebilir olan modellerin takımları projenin tamamlanma süresi içindeki tüm değişikliklerin ve gelişmelerin tarihini önceden bilir, bu plan çerçevesinde iş yapar.
- Değişiklik olduğunda tüm plan iptal olur ve yeni bir program yapılır.
- Yeni programda ise sadece en önemli olan değişiklikler seçilip projeye dahil edilir.
- Çevik modeller uyarlanabilir olanlara dahildir.
- Yani değişime açıktır ve uyarlanabilirdir.
- Uyarlanabilir olduğu için bu kadar net bir şekilde planlanmayan proje müşterinin isteklerine öre şekil alır ve istenilen başarıyı gösterir.

Çevik Yazılım Modelinin Diğer Modellerden Farkı

Çevik modellerin bir diğer özelliği de;

- Diğer yinelemeli geliştirme modellerin (iterative development models) zaman dilimi ay olarak alınırken, çevik modellerde bu süre haftaya kadar düşer.
- Bu kısa süre içinde küçük ilerlemeler şeklinde ve her adımda müşteriden geri bildirim alınarak yazılım ortaya çıkarılır.



Çevik Yazılım Geliştirme Modelleri

Çevik modeller tam bir yazılım süreci değildir ama kapsamlı yazılım geliştirme yöntemlerini tamamlayıcı niteliktedir.

Başlıca çevik yazılım geliştirme süreç modelleri:

- Uçdeğer Programlama (“Extreme Programming – XP”)
- Adaptif Yazılım Geliştirme (“Adaptive Software Development -ASD”)
- Dinamik Sistem Geliştirme Metodu (“Dynamic System Development Method”)
- SCRUM
- CRYSTAL
- Özellik Güdümlü Gelistirme (“Feature-Driven Development – FDD”)
- Çevik Tümlsik Süreç (“Agile Unified Process – AUP”)
- Çevik bilgi Metodu (Agile Data Method)

Uçdeğer Programlama (Extreme Programming – XP)

- Uçdeğer programlama, Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır.
- Tüm gereksinimler senaryolar şeklinde oluşturulur. Daha sonra senaryolar işlere bölünür.
- Yazılımcılar çiftler halinde çalışır ve her iş için test de geliştirir. İşleri sonra tümleştirir.
- Sistemin müşterisi de geliştirici takımın devamlı bir parçasıdır.

4 prensip etrafında toplanır:

- Basitlik,
- İletişim,
- Geri bildirim,
- Cesaret



Uçdeğer Programlama

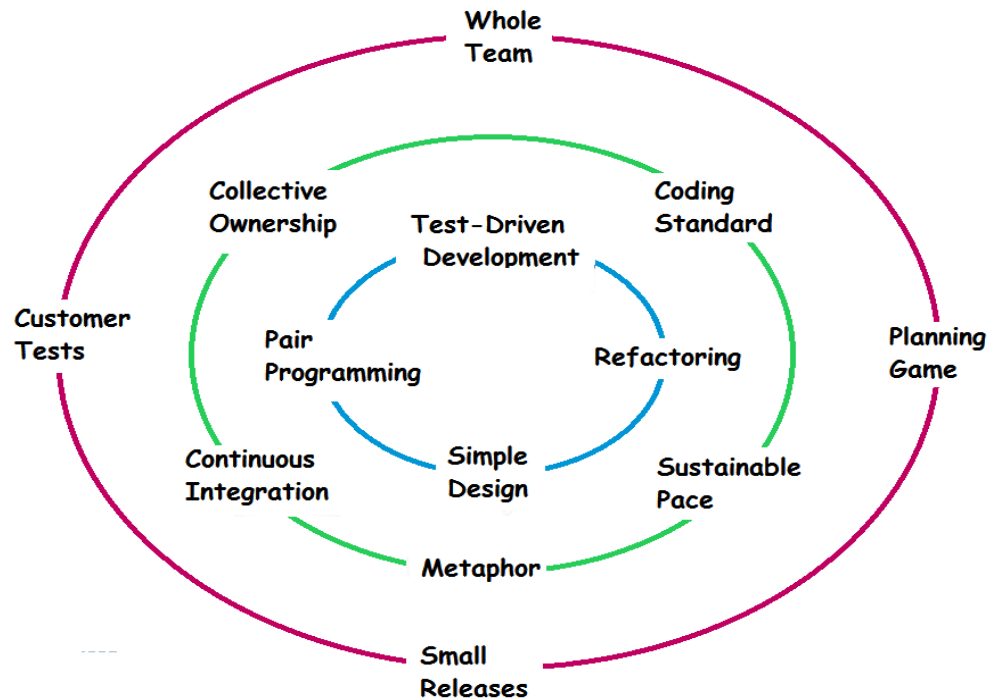
12 temel pratiğin birlikte uygulanmasını gerektirir:

- Planlama oyunu
- Kısa aralıklı sürümler
- Müşteri katılımı
- Yeniden yapılandırma (“refactoring”)
- Önce test (“test first”)
- Ortak kod sahiplenme
- Basit tasarım
- Metafor
- Eşli programlama (“pair programming”)
- Kodlama standardı
- Sürekli entegrasyon (“continuous integration”)
- Haftada 40 saat çalışma

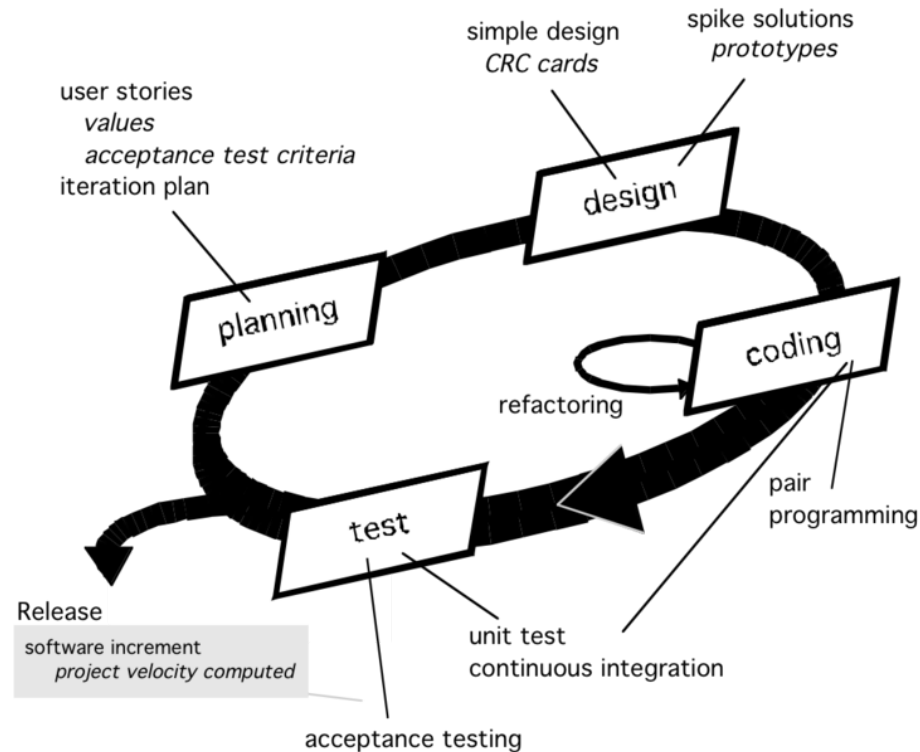


Pratik
Bilgiler

Uçdeğer Programlama



Uçdeğer Programlama



Uçdeğer Programlama

Bazı özellikleri

- Takımın bilgisayarları kübiklerle bölünmüş büyük bir odanın ortasında yer alır.
- Bir müşteri temsilcisi geliştirici takımlarla devamlı beraber çalışır.
- Hiç kimse aynı iş için peşpeşe iki haftadan fazla çalışamaz.
- Takım içerisinde özelleşme yoktur. Herkes belirtim, tasarım, kodlama ve test aşamalarında görev yapar.
- Parçalar geliştirilmeden önce ayrı büyük bir tasarım aşaması yoktur. Onun yerine parçalar geliştirilirken tasarım da değiştirilir.

Küçük ve orta ölçekli projelerde kullanılırlar

Kullanıcının gereksinimleri belirsiz ya da değişkense kullanılırlar.

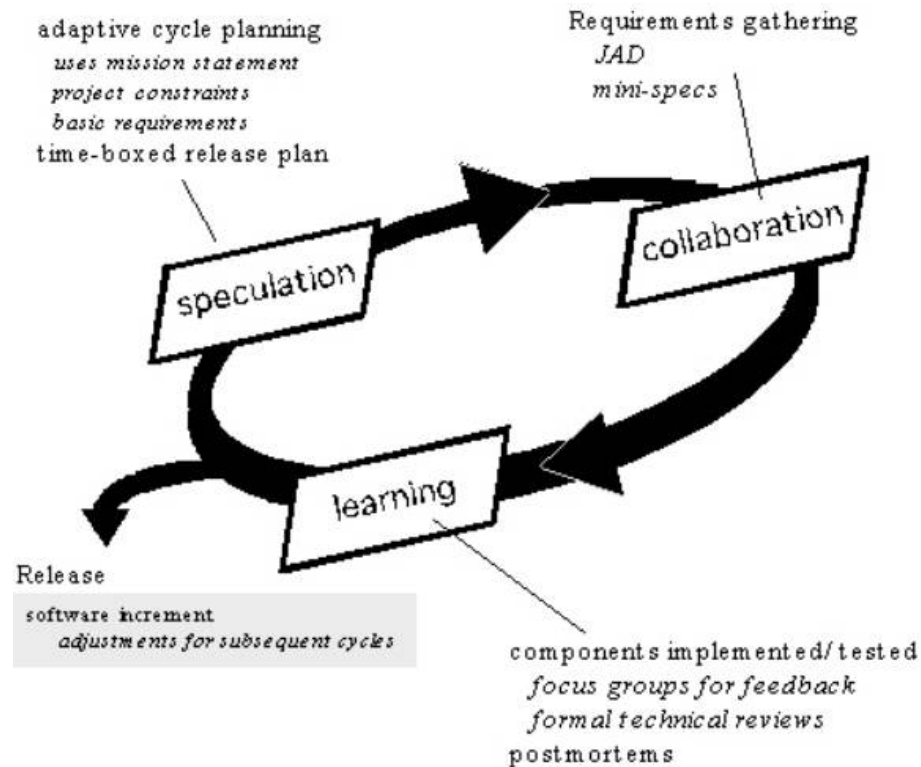
Adaptif Yazılım Geliştirme (Adaptive Software Development -ASD)

ASJim Highsmith tarafından önerilmiştir

D — ayırt edici özellikleri

- Göreve-dayalı planlama.
- Bileşene-dayalı odak
- “zaman aralıkları” kullanır
- Riskleri açık şekilde göz önünde bulundurur
- Gereksinim toplamada işbirliğini vurgular
- Süreç boyunca “öğrenme” yi vurgular

Adaptif Yazılım Geliştirme



Dinamik Sistem Geliştirme ("Dynamic System Development")

DSDM Konsorsiyum tarafından önerilmiştir (www.dsdm.org).

DSDM—ayırt edici özellikleri

- Pek çok yönden XP ve/veya ASD'ye benzer
- Dokuz yönlendirici prensip
 - Kullanıcıların aktif katılımı önemlidir.
 - DSDM takımları karar vermede yetkilidirler
 - Sık teslim edilen ürünlere odaklanılır.
 - Teslim edilebilirliğin kabul kriteri olarak iş amacına uygunluk kullanılır
 - Tekrarlanan (iterative) ve artımlı (incremental) geliştirme doğru iş sonucuna ulaşmak için gereklidir.
 - Geliştirme sırasında gerçekleştirilen tüm değişiklikler geri alınabilirdir..
 - Gereksinimler yüksek seviyede temellendirilirler
 - Test tüm yaşam döngüsü boyunca entegredir

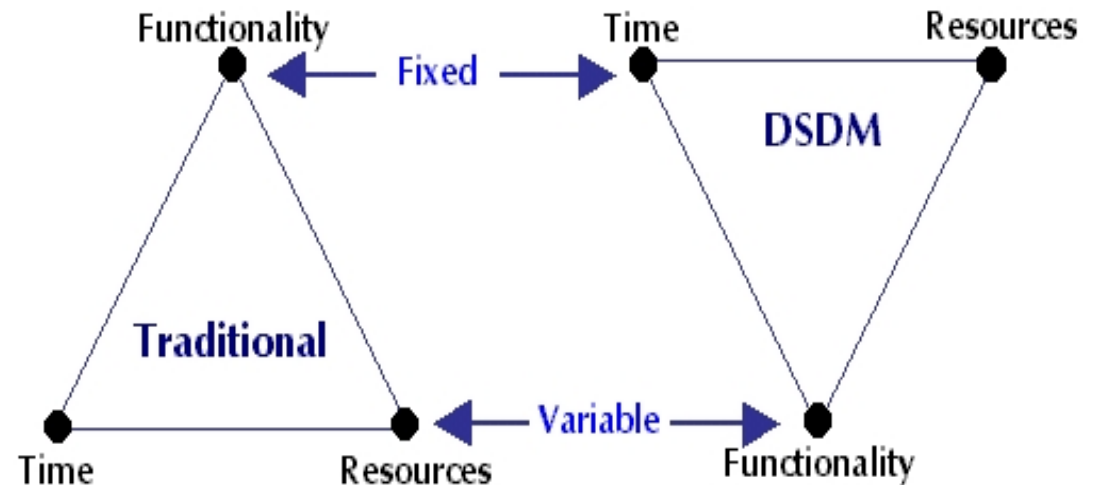
Dinamik Sistem Geliştirme

➤ Basit.

➤ Genişletilebilir.

➤ Düz ileri Framework tabanlı.

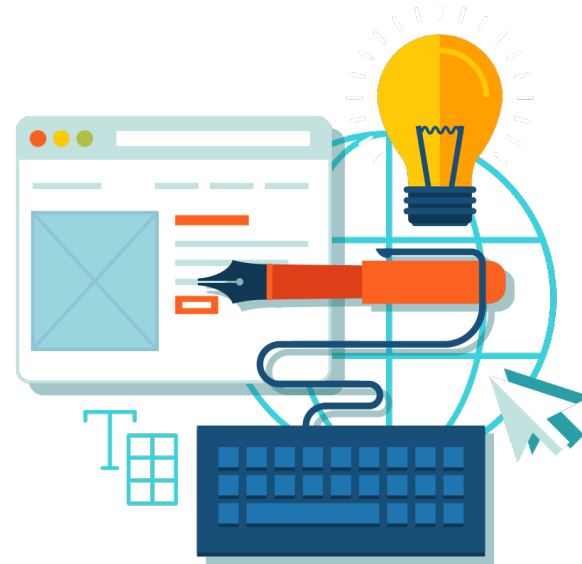
➤ Tüm proje türlerinde çözüm sağlayıcı değildir.



Dinamik Sistem Geliştirme Proje Yapısı

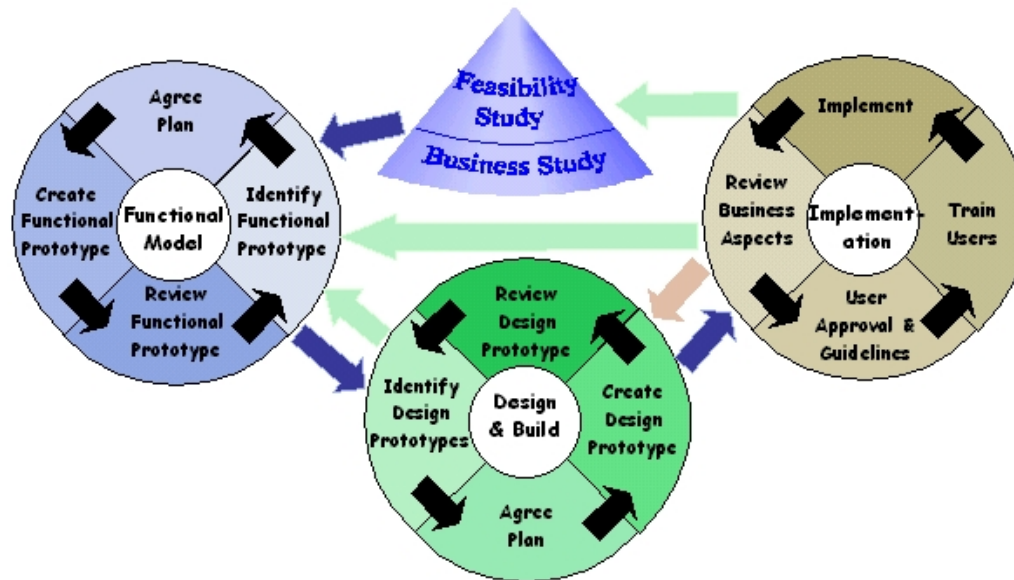
DSDM projesi, organize rolleri, gömülü rolleri ve sorumlulukları ile zengin bir küme ve çeşitli çekirdek teknikleriyle desteklenen 7 aşamalı adımdan oluşur.

- Roller ve Sorumluluklar
- Takım Organizasyonu ve Boyutu
- 7 Aşamalı kurallar. Bunlar:
 1. Ön Proje
 2. Fizibilite Çalışması
 3. İş Çalışması
 4. Fonksiyonel Model Yineleme
 5. Tasarım & Yapı Yineleme
 6. Uygulama
 7. Proje Sonrası



Dinamik Sistem Geliştirme Süreci

The DSDM Development Process



Scrum

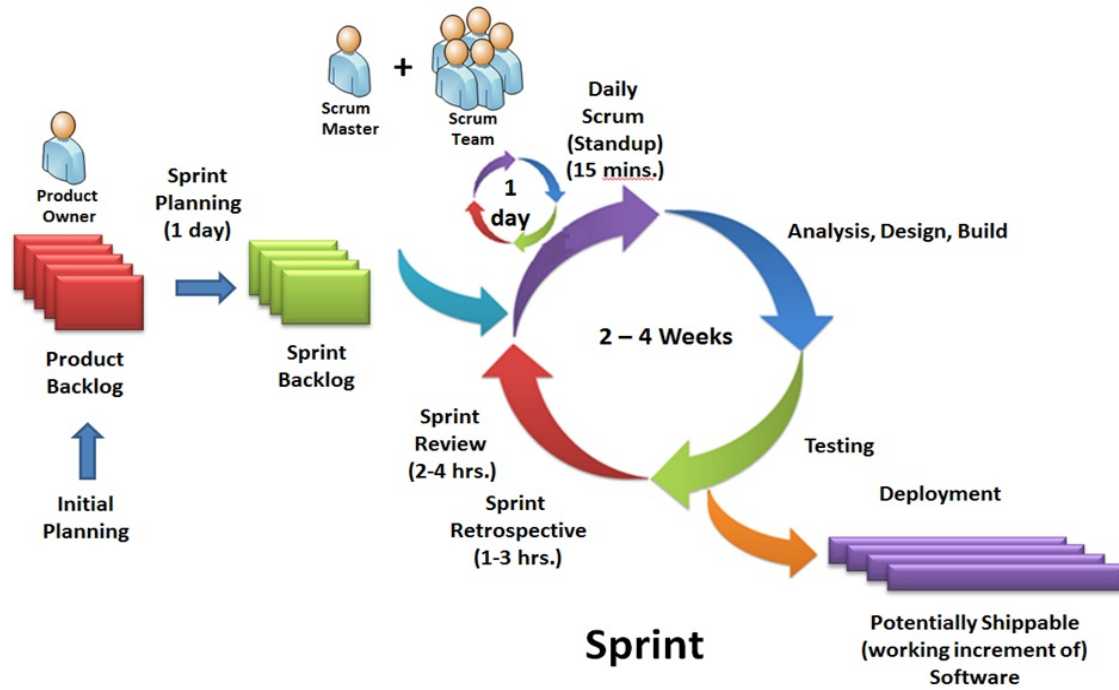
- Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir.
- Yazılımı küçük birimlere (sprint) bölerek, yinelemeli olarak geliştirmeyi öngörür.
- Bir yinelemenin tanımlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık toplantılarla sürekli is takibi yapılır.
- Karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler (<20) için uygundur.
- Yüksek seviyede belirsizlik arz eden projelerde, son yıllarda daha yaygın biçimde uygulandığı ve başarılı sonuçlar ürettiği bildirilmiştir.

Scrum Ayırt Edici Özellikleri

Scrum

- Geliştirme işi “paket”lere bölümlenmiştir.
- Test ve belgelendirme ürün oluşturuldukça süregelendir.
- İş “sprint” ler halinde gerçekleştirilir ve var olan gereksinimlerin “backlog” u olarak çıkarılır.
- Toplantılar çok kısadır ve bazen başkan bile içermez.
- Zaman aralıklı olarak müşteriye “demo” lar sunulur.

Scrum Süreci



Kaynak: <http://www.insolaria.it/blog/2014/04/scrum-e-le-medotologie-agili/>

Scrum Rollerleri

Şeffaflık

Sürecin tümleriyle geliştiriciye görünür olmasıdır. Böylece ortak bir görüş ortaya konur. Örneğin, “bitti” takımdakilerin artık işin sonuna gelindiğinde hep bir ağızdan söyledikleri bir ortak bir cümledir. Çalışılan sürecin bittiği anlamına gelir.

Denetim

Scrum, kullanıcılarına yönelik istenmeyen sapmaları tespit etmek için sürecin ve programın sürekli denetim altında tutulması gerekir. Scrum karmaşık ürün gelişimini basite indirmek için yapılandırılmış bir çerçeve olarak düşünebiliriz

Uyum

Yazılan programların talebe uygun olup olmaması durumudur. Herhangi bir saptama söz konusu ise bu açıklar giderilir ve talebe uyumlu hale getirilir. Saptamaları aza indirmek kısa bir sürede yapılmalıdır.

Scrum Takımı

Ürün Sahibi, Geliştirme Ekibi ve Scrum Master'dan oluşur. Takım kendi kendini örgütler. Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar. Scrum'ın takım modeli esneklik, yaratıcılık ve verimliliği optimize etmek için tasarlanmıştır.

Ürün Sahibi (Product Owner)

Ürünün değerinden ve gerikaydından (Backlog) sorumlu olan kişidir . Ürün Backlog öğeleri açık olmalıdır. Bunuda sağlayacak olan kişi Product Owner'dir. Sorumluluğun büyük bir kısmı Product Owner'dadır. Takımda Product Owner tektir. Product Owner'in başarılı olması için, tüm organizasyonun kendi kararlarına saygı göstermesi gerekir. Geliştirme Ekibi, Product Owner'in kararlarına göre hareket eder .

Scrum Rollerleri

Sprint

Scrum çerçevesinde her bir iterasyon Sprint olarak isimlendirilmektedir. Sprintleri paketlenmiş zaman aralıkları olarak düşünebilirsiniz. Yeni Sprint önceki Sprint sonucundan hemen sonra başlar. Sprintlerimiz başlangıç ve bitiş tarihleri bulunmaktadır, ve ekibin hızını ayarlamak için her bir sprint'in aynı sürede tutulması önerilmektedir. Scrum yöntemi ile yönetilen projeler "sprint" denilen fazlara ayrılır. Her bir "sprint" in süresi genellikle (en fazla) 30 gündür. İdeal olan proje boyunca tüm "sprint" ler aynı süreli olur.

Sprint İnceleme

Burada katılımcılar, gelecek hakkında iş birliği yapabilir. Gayriresmi bir toplantı olup iş birliğini daha iyi gerçekleştirmek için toplanılır. Bu, bir aylık sprint için dört saatlik zamanı kapsar. Sprint İnceleme sonucu bir sonraki Sprint için muhtemel Ürün Backlog öğelerini tanımlar.

Sprint Planlama Toplantısı

Sprint ile yapılacak çalışma Sprint Planlama Toplantısı ile planlanmıştır. Bu plan tüm Scrum Ekibinin ortak çalışmasıyla oluşturulur. Örneğin, iki haftalık Sprint'te dört saatlik Sprint Planlama Toplantıları vardır. Bu toplantılarda artımın nasıl sağlanacağı konuşulur.

Günlük Scrum (Daily Scrum)

15 dakikalık toplantılardan oluşur. Bu günlük Scrum'larda planlama yapıldığı gibi bir sonraki gün için tahminler yapılır. Bu toplantılar her gün düzenlenir ve amaç karmaşıklığı azaltmaktır. Bu toplantılarda her bir üye aşağıdaki sorulara cevap verir :

- Son toplantıdan bu yana neler tamamlandı ?
- Sonraki toplantıda neler yapılacak ?
- Engeller ne şekildedir ?

Kısaca hedefe doğru ilerlemeler değerlendirilir.

Scrum Rollerleri

Sprint Backlog

Takım belirlenir ve “Sprint Planning” dediğimiz en fazla 4 hafta sürececek olan küçük döngülerle “Sprint” ile işe başlanır. Her döngü için ürün gerikaydından önemli gereksinimler seçilerek sprint gerikaydı (Sprint Backlog) oluşturulur ve sprint boyunca bu gereksinimler geliştirilir.

Product Backlog

İlk önce müşteriye yani ürün sahibine (Product Owner), istediği ürün gereksinimlerinin neler olduğu sorulur ve bu ihtiyaçlar çıkartılır (Product BackLog). Product BackLog içerisinde maddeler en önemlisinden en aza doğru sıralanır. Belirli bir periyot içerisinde belirtilen gereksinimleri karşılayan tam olarak bir portatif müşteriye teslim edilir .

Sprint Retrospective

Sprintte bir sonraki aşama için iyileştirme fırsatı yakalanır. Bu bir aylık sprint için toplamda 3 saatlik toplantıdır. Burada takım işini kolaylaştırmak için planlar oluşturulur. Böylece gelişim süreci ve sonraki Sprinti daha etkili ve zevkli hale getirmek için uygulamalar geliştirme ortamı yaratılır. Bu toplantılar sonucunda ürün kalitesini arttırma yolları planlanmış olur.

Scrum Rollerleri

Scrum Ustası(The Scrum Master)

Scrum'dan sorumlu olan kişidir. Scrum Master, Scrum Takım teorisi ve uygulamaları kurallara uyarak sağlar. Scrum Master takım için liderdir. Scrum Master, ekip tarafından yaratılan değeri maksimize etmek için çalışır. Scrum Master, çeşitli yollarla Ürün Sahibine yardım eder . Bunlar:

- Ürün Backlog yönetimi bulma teknikleri
- Geliştirme ekibinin vizyon ve hedeflerini belirlemek
- Açık ve özlü Ürün Backlog öğeleri oluşturmak için geliştirme ekibine temel yolların öğretilmesi
- Anlaşılabilir bir ortamda uzun vadeli ürün planlamaları yapmak
- Çevikliği anlama ve uygulama

Scrum Master, geliştirme ekibinde hizmet eder:

- Örgütlenmeye koçluk eder
- Geliştirme ekibi için değeri yüksek ürünler oluşturmak
- Geliştirme ekibinin ilerlemesine engel olacak şeyleri ortadan kaldırmak
- Talebe göre Scrum olaylarının kolaylaştırılması

Scrum Master organizasyon içinde çalışır:

- Scrum'ın belirlenmesinde koçluk eder
- Örgüt içinde planlama yapar
- Takımın verimliliğini artırır

Scrum Örnek

Firma bir proje alıyor ve proje tarihi sabit değil. Proje sahibinin beklentileri yüksek ancak istekleri yeterince net değil. Ayrıca süreçler içinde yeterli zaman bulunmamakta. Proje ekipleri arasında iletişim problemleri de var. Proje ise Kredi Kartı Taahhüt Programı. Sonunda iş birliği ile 8 ay olarak yer alan proje, 3'er haftalık 6 Sprint ile 18 haftada tamamlanıyor.



Özellik GÜdümlü Gelistirme (Feature-Driven Development – FDD)

- Jeff De Luca ve Peter Coad tarafından 1997’de geliştirilen ve özellik tabanlı gerçekleştirimi esas alan bir modeldir.
- Süreç beş ana basamaktan oluşur; her bir basamak için çıkış şartları tanımlanır ve bunlara uyulur.
 - Genel sistem modelinin geliştirilmesi,
 - Özellik listesinin oluşturulması,
 - Özellik güdümlü bir planlama yapılması,
 - Özellik güdümlü tasarımın oluşturulması,
 - Özellik güdümlü geliştirmenin yapılması.
- Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur.



Çevik Tümlşik Süreç (Agile Unified Process – AUP)

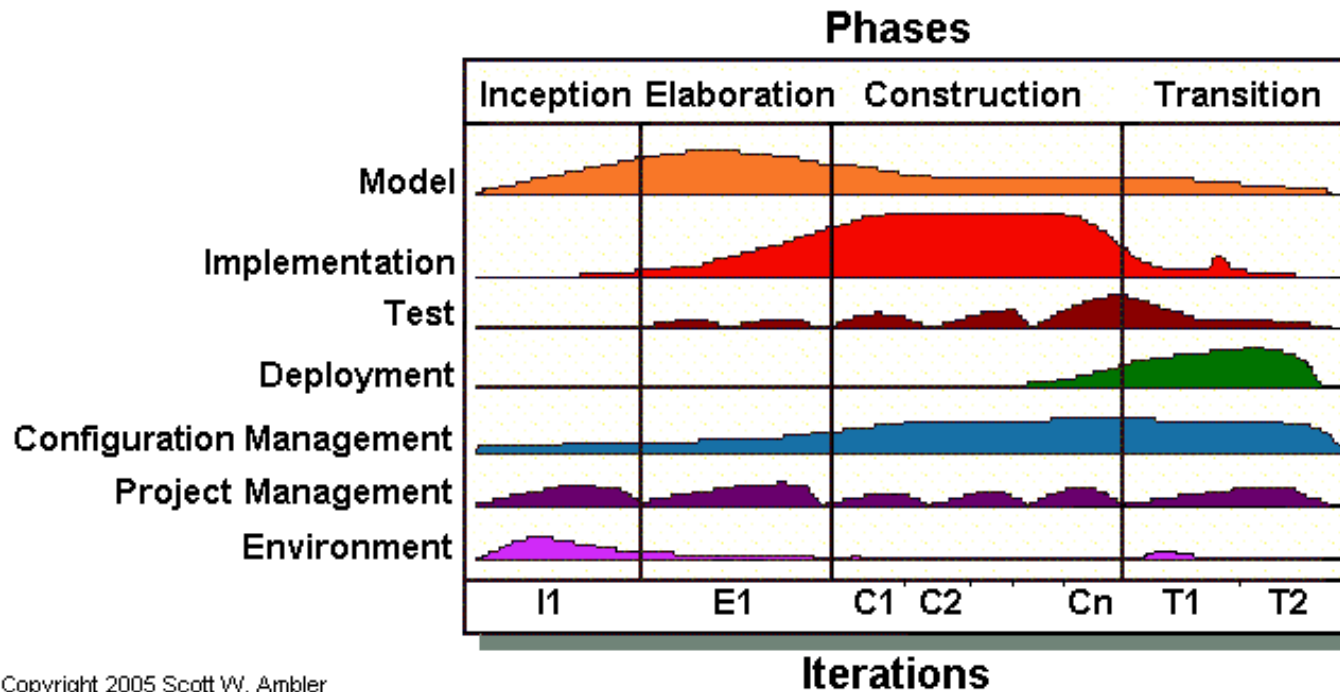
Çevik Tümlşik Süreç (Agile Unified Process – AUP”); Rational Unified Process (RUP)’un, çevik yaklaşıma göre adapte edilerek basitleştirilmiş halidir.

Test-güdümlü geliştirme (“test driven development – TDD”), çevik değişiklik yönetimi, veri tabanını yeniden yapılandırma gibi çevik pratikleri uygulatır.

RUP’dan farklı olarak, 7 adet disiplin içerir.



Çevik Tümlerik Süreç



Copyright 2005 Scott W. Ambler

Yazılım Süreç Modeli Seçimi

- Yazılım süreç modelleri birbirinden tümüyle ayrı değildir ve çoğu zaman aslında beraber kullanılırlar.
- Hangi modelin ve model içerisindeki hangi adımların gerçekleştirileceğini seçmekle görevli olan kişiye “süreç mimarı” denir.
- Her modelin güçlü ve zayıf yanlarını değerlendirdikten sonra süreç mimarı proje için en iyi modeli seçmelidir.

Yazılım Süreç Modeli Seçimi

Süreç modeli seçiminde yararlı olabilecek bazı kriterler;

- Modelin oluşabilecek riskleri tolere edebilme kapasitesi
- Geliştirici kurumun son kullanıcılara erişim imkanı
- Bilinen gereksinimlerin ne kadar iyi tanımlanabildiği
- Erken işlevlerin önemi
- Problemin karmaşıklığı ve çözüm için olası adaylar
- Gereksinim değişikliğinin tahmin edilen sıklığı ve oranı
- Kurumun yönetsel kapasitesi

Yazılım Süreçleri – IEEE/IEA 12207 nedir?

- Bilgi Teknolojileri-Yazılım Yaşam Döngüsü Süreçleri için Standard.
- Bir yazılım sisteminin tüm yaşam döngüsünü kapsayan süreçler kümesini tanımlar.
 - Kavramsal fikrin ortaya çıkışından
 - Yazılımın emekli oluşuna kadar
- Yazılım süreç modelleri için ortak bir çerçeve sağlar
- Bir organizasyon kendi iç kullanımı için ya da birden çok organizasyon kontrata bağlı çalışmak için kendine adapte edebilir.
- Projeye göre uydurulabilir
 - Genel, büyük ve karmaşık projeler için yazılmıştır
 - İhtiyaç, büyüklük, karmaşıklık, maliyet, zaman ve performansa uydurulabilecek şekilde tasarlanmıştır.
- Uygulamalar için bir kılavuzdur

12207 ne değildir?

➤ Yazılım geliştirme için bir reçete değildir.

IEEE/EIA 12207.0-1996
(A Joint Standard Developed by IEEE and EIA)

➤ Yönetim ya da mühendisliğin yerine geçmez.

IEEE/EIA Standard

➤ Ürün ya da ölçme standardı da değildir.

*Industry Implementation of
International Standard
ISO/IEC 12207 : 1995*

*(ISO/IEC 12207) Standard for Information
Technology—*

Software life cycle processes

March 1998



Networking the World™



Electronic Industries Alliance

Kullanımı

➤ Birincil yaşam döngüsü süreçleri:

- Acquire, supply, develop, operate, and maintain software – Yazılım kazanma, tedarik, geliştirme, işletim ve sürdürme.

➤ Destekleyici yaşam döngüsü süreçleri:

- Yukarıdaki fonksiyonları, kalite güvencesi, konfigürasyon yönetimi, ortak gözden geçirme, denetleme, geçerleme, doğrulama, problem çözme ve belgelendirme ile destekler.

➤ Kurumsal yaşam döngüsü süreçleri:

- Organizasyonun süreçleri ve personelini yönetmesini ve geliştirmesini sağlar.

➤ Yazılım ürün yaşam döngüsünde yer alan müşteri, sağlayıcı ve tüm iştirakçiler arasında anlaşmayı artırır.

Yaşam döngüsünü bölümlendirme

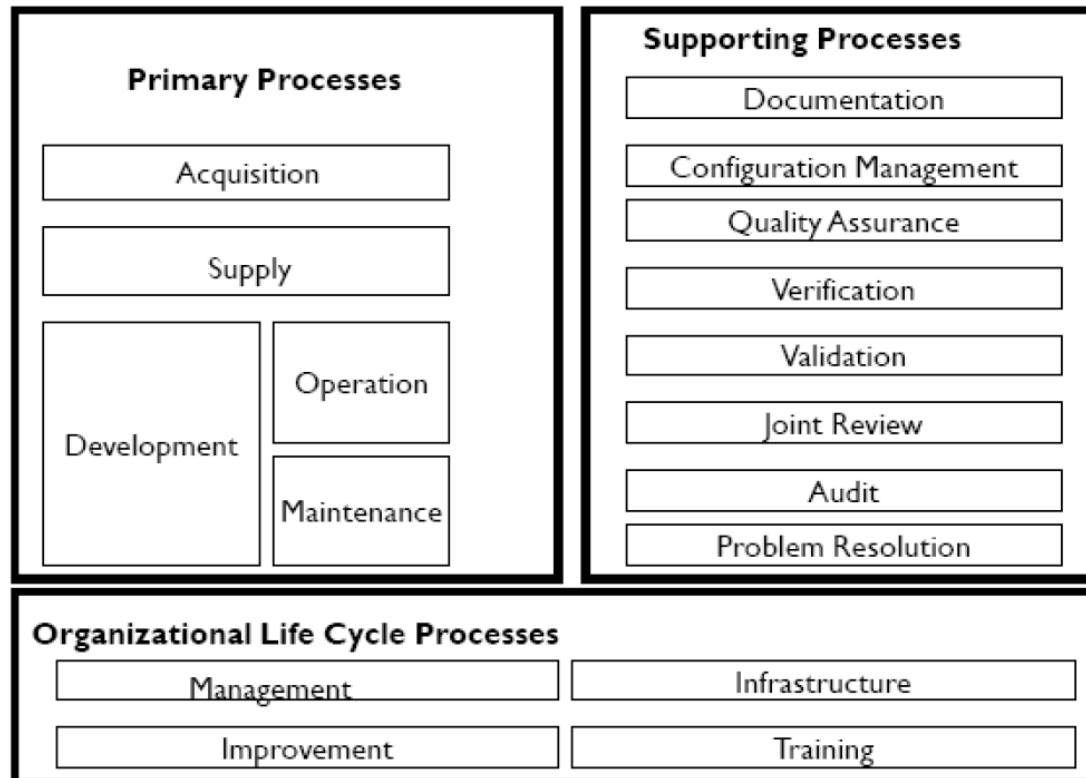
Modularity (modülerlik)

- Strongly cohesive (güçlü bağlı): süreç içindeki işler (task) fonksiyonel olarak birbirine bağlı olmalı
- Loosely coupled (zayıf birleşme) Süreçler arasındaki bağlar en az olmalı
- Association rules (ilişkilendirme kuralları)
 - Bir fonksiyon birden fazla süreç tarafından kullanılıyorsa, o durumda fonksiyon kendi başına bir süreç haline gelir
 - Eğer süreç A sadece ve sadece süreç B tarafından çağrılıyorsa, süreç A B'ye aittir.

Sorumluluk

- Bir süreç yazılım yaşam döngüsündeki bir organizasyon ya da bir tarafın sorumluluğuna verilir
 - Parçaları farklı organizasyonların sorumluluğunda olan fonksiyonlar süreç olamaz.

IEEE/EIA 12207 Yaşam Döngüsü (Yazılım Süreçleri)



Uygulama

- Bir BS(Bilgisayar Sistemi) organizasyonunda proje yöneticisi olarak görevlendirildiniz.
 - İşiniz daha önce geliştirdiklerinize benzer bir sistem geliştirmek olan sözleşmeli bir projeyi yönetmek. Yalnız bu defaki proje daha öncekilere göre daha büyük ve daha karmaşık.
 - Gereksinimler müşteri tarafından baştan sona belgelendirilmiş.
- ➡ **Hangi süreç modelini kullanırsınız? Neden?**

Özet

- Birleşik Süreç 3 aşamalıdır. Bunlar; yinelemeli, arttırmalı ve evrimsel, risk güdümlü şeklindedir.
- Birleşik Süreç Fazları başlangıcından üretime kadar 7 aşamada iterasyon yapılarak gerçekleştirilir.
- Çevik modeller, mevcut geleneksel modellere alternatif olarak, 1990'larda ortaya çıkmaya başlamıştır.
- Çevik (Agile) Yaklaşım da daha önemliden az önemliye doğru bir gerçekleştirim mevcuttur.
- Çevik (Agile) Yazılım Süreç Modelleri 7 ana başlık altında toplanmıştır.
- 2001 yılında, dünyanın önde gelen çevik modellerinin temsilcilerinin çıkardıkları Çevik Yazılım Geliştirme Manifestosuna göre;
 - Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden;
 - Çalışan yazılım, detaylı belgelerden;
 - Müşteri ile işbirliği, sözleşmedeki kesin kurallardan;
 - Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten; daha önemli ve önceliklidir.

Özet

- Çevik Yazılım Geliştirme Manifestosu- Prensipleri projenin baştan sona müşteri ile çalışanlar arasında gerçekleştirilip, minimum sürede tamamlanmadı öngörülmektedir. Eğer müşteri istekleri değişirse yine de gereken zaman diliminde yapılanma hızlı bir şekilde gerçekleşmektedir.
- Uçdeğer Programlama (Extreme Programming – XP) 4 prensip etrafında toplanır. (Basitlik, İletişim, Geri bildirim, Cesaret)
- Uçdeğer Programlama 12 temel pratiğin birlikte uygulanmasını gerektirir.
- Adaptif Yazılım Geliştirme kurgu, işbirliği ve öğrenmeyi temel almaktadır.
- Dinamik Sistem Geliştirme Dokuz yönlendirici prensibi benimsenmiştir.
- DSDM projesi, organize rolleri, gömülü rolleri ve sorumlulukları ile zengin bir küme ve çeşitli çekirdek teknikleriyle desteklenen 7 aşamalı adımdan oluşur.

Özet

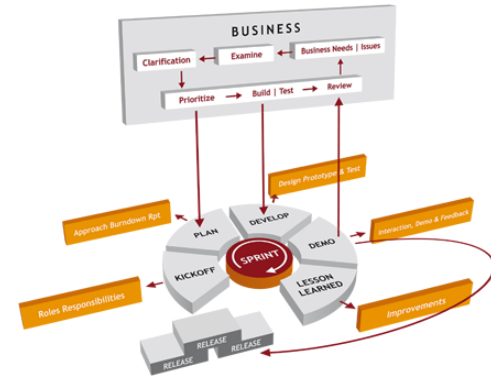
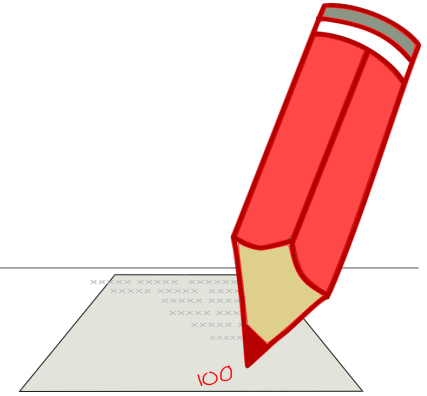
- Scrum, Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir.
- Scrum, yazılımı küçük birimlere (sprint) bölerek, yinelemeli olarak geliştirmeyi öngörür.
- Scrum, Karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler (<20) için uygundur. İş, "sprint" ler halinde gerçekleştirilir ve var olan gereksinimlerin "backlog" u olarak çıkarılır. Zaman aralıklı olarak müşteriye "demo" lar sunulur.
- Scrum süreci önemli bir model olmakla birlikte rolleri bir projede olması gereken en belirleyici kısımlardır.
- FDD Süreci beş ana basamaktan oluşur; her bir basamak için çıkış şartları tanımlanır ve bunlara uyulur. Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur.
- Çevik Tümlleşik Süreç (Agile Unified Process – AUP"); Rational Unified Process (RUP)'un, çevik yaklaşıma göre adapte edilerek basitleştirilmiş halidir. RUP'dan farklı olarak, 7 adet disiplin içerir.
- IEEE/IEA 12207 Bilgi Teknolojileri-Yazılım Yaşam Döngüsü Süreçleri için bir Standarttır. Yazılım süreç modelleri için ortak bir çerçeve sağlar

Çalışma Soruları

1. Birleşik süreç kavramı ve aşamaları nelerdir?
2. Birleşik süreç fazları nelerdir söyleyiniz?
3. Çevik modeller alternatif modellere istinaden kaç yılında ortaya çıkmıştır?
4. Çevik yazılım da nasıl bir gerçekleştirim vardır?
5. Çevik yazılım süreç modelleri kaç aşamalıdır açıklayınız?
6. Çevik modellerin temsilcilerinin çıkardıkları Çevik Yazılım Geliştirme Manifestosunu neyi amaçlamaktadır?
7. Uçdeğer Programlamanın benimsediği 4 temel prensibi söyleyiniz?
8. Adaptive Yazılım Geliştirmenin temel aldığı modelleme kavramları nelerdir?
9. DSDM projesi kaç aşamadan oluşur bunlar nelerdir?
10. Scrum nedir? Scrum modelini açıklayınız.
11. Scrum rolleri ne amaçla faaliyet göstermektedir. Bu rolleri söyleyiniz .
12. FDD Süreci aşamaları nelerdir söyleyiniz?
13. IEEE/IEA 12207 ne anlama gelmektedir?

Ödev

1. CRYSTAL modelini araştırınız.
2. Feature Driven Development modelini araştırınız.
3. Çevik bilgi Metodu (Agile Data Method) araştırınız.
4. Rational Unified Process (RUP)'i araştırınız.



Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Saridoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

<http://dsdmofagilemethodology.wikidot.com/>

<http://caglakaya.piququestion.com/2014/07/01/244/>

Sorularınız





YMH311-Yazılım Tasarımı ve Mimarisi

Çevik Yazılım Geliştirme Yöntemleri

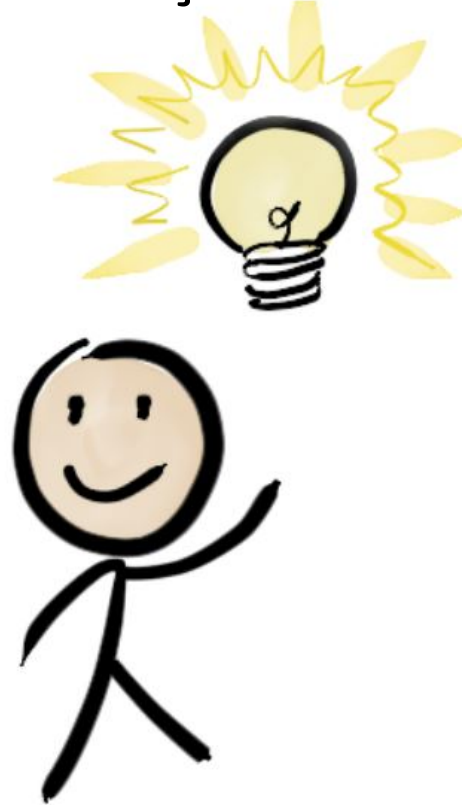
Prof. Dr. Resul DAŞ
Fırat Üniversitesi Yazılım Mühendisliği Bölümü

İçindekiler

1	Günümüzde Yazılım Projelerinin Durumu.....	3
2	Çevik Yazılım Geliştirme Yöntemi.....	9
3	Geleneksel Model vs. Agile.....	17
4	Değerlendirme.....	26
5	Çevik Yazılım Şemsiyesi.....	28
6	Scrum Modeli.....	29

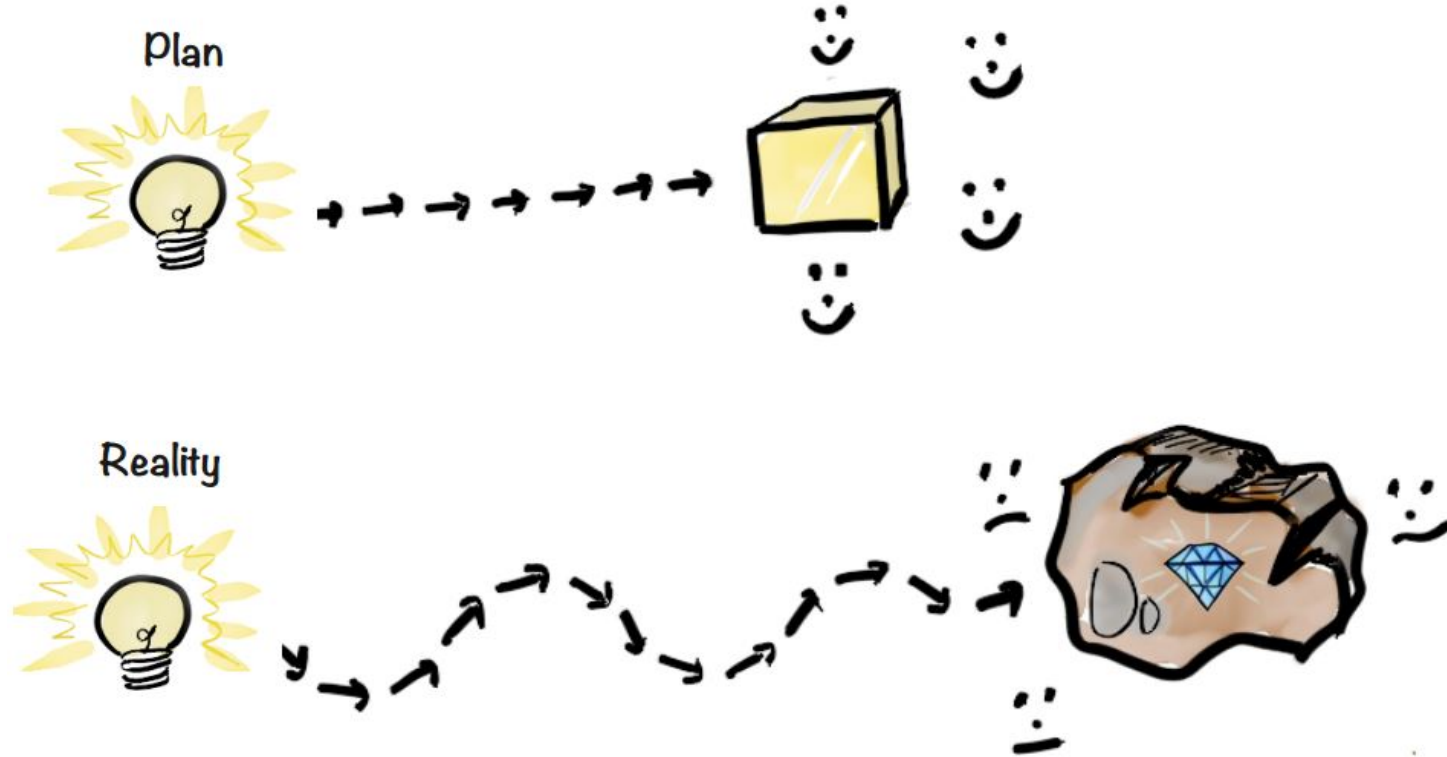
1.Günümüzde Yazılım Projelerinin Durumu

- Birçok proje harika bir fikir ile başlar!



1.Günümüzde Yazılım Projelerinin Durumu

- Bu projelerin büyük bir kısmının başarısız olması muhtemeldir!

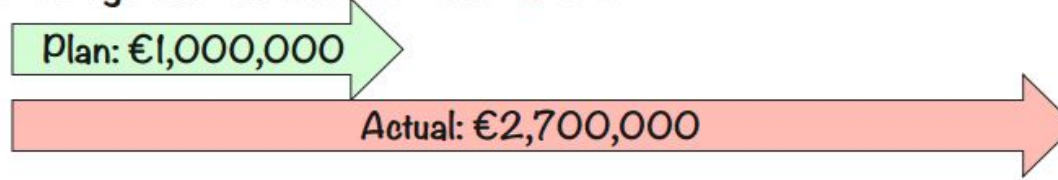


1.Günümüzde Yazılım Projelerinin Durumu

- Birçok Bilgi Teknolojisi projesi başarısız olmuş veya gecikmiştir. The Standish Group, 10 yıl içerisinde 40.000'den fazla proje üzerinde çalışmıştır.

IT project success rate 1994: **15%**

Average cost & time overrun: **≈170%**



IT project success rate 2004: **34%**

Average cost & time overrun: **≈70%**



1.Günümüzde Yazılım Projelerinin Durumu

Ülkemizde durum nasıl?

Durum	Oran
Tam başarılı	%4-5
Kısmen başarılı	%45-50
Çöpe gidenler	%50

Agile Turkey

1.Günümüzde Yazılım Projelerinin Durumu

➤ Başarısızlığın ana sebepleri:

- Müşterinin isteklerini doğru analiz edememek;
- Proje için uygun ekibi kuramamak;
- Yanlış teknoloji ve mimari seçimleri;
- Geleneksel yöntemlerin eksiklikleri;
- Müşteriyle iletişimden kaçınmak vs.



1.Günümüzde Yazılım Projelerinin Durumu

Peki ne yapmalıyız?



2.Çevik Yazılım Yöntemi



➤ **Tekrarlamalı** ve **artımsal** bir ürün geliştirme yöntemidir.

▪ **Bireyler ve etkileşimi**, süreç ve araca tercih eder.

▪ **Çalışan bir yazılımı**, detaylı ürün belgelendirmeye tercih eder.

▪ **Müşteri ile işbirliğini**, sözleşmedeki kesin kurallara tercih eder.

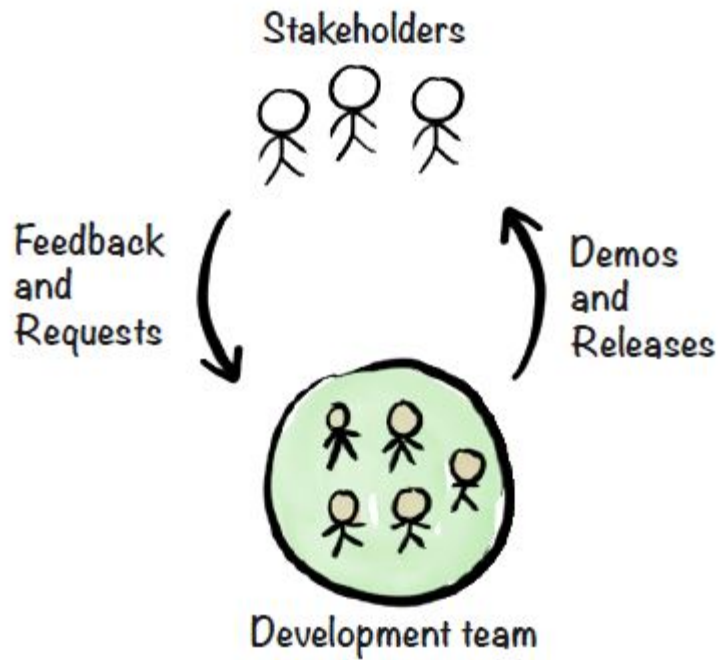
▪ **Değişikliklere uyum sağlayabilmeyi**, belirli bir plana tercih eder.



2.Çevik Yazılım Yöntemi

- Hızlı, devamlı ve kullanışlı yazılım üreterek müşteri memnuniyeti sağlamayı amaçlar.
- Geliştiriciler ile iş adamları arasında günlük ve yakın işbirliği bulunmalıdır.
- Çalışan yazılım gelişimin en önemli ölçüsüdür.
- Taleplerdeki geç değişikliklerin de memnuniyetle karşılanır.
- Yüz yüze görüşme iletişimin en güzel yoludur.
- Kendi kendini organize eden takım yapısı gereklidir.
- Basitlik önemlidir.

2.Çevik Yazılım Yöntemi



➤ Çevik yazılım metodu, kısa vadeli planlar ve küçük parçalar halinde yazılımın geliştirilmesini ön görür. Yazılımın geliştirilmesindeki geri dönüş (**feedback**) ve değişikliklere uyum sağlamak son derece önemlidir. Her yapılan yineleme yazılımı hedeflenen adıma bir adım daha yakınlaştırır. İstenilen sonuca ulaşmak adına birden çok yineleme gereklidir.

2.Çevik Yazılım Yöntemi

➤ Örnek: 2007



2008
- App Store
- 3G



2009
- Copy/Paste
- Search



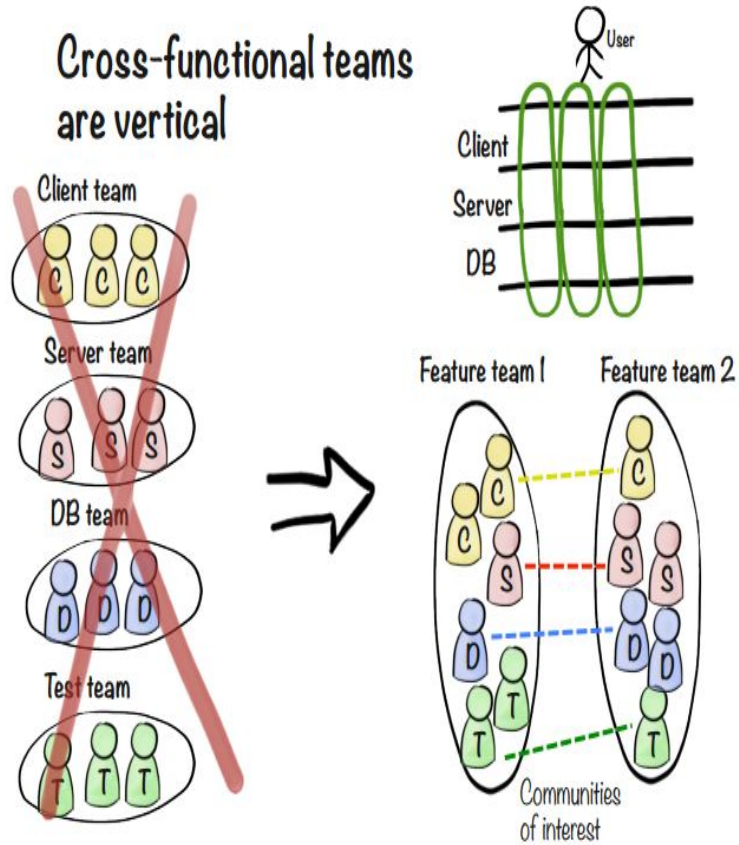
2010
- Multitasking
- Video calls



2.Çevik Yazılım Yöntemi

- Temel prensipler:
- Müşteriyi memnun etmek
- Değişen ihtiyaçları karşılamak
- Sık aralıklarla ürün teslimi yapmak
- Yüz yüze iletişime önem vermek
- Sürdürülebilir gelişmeyi desteklemek
- Teknik mükemmeliyete, iyi dizayna ve sadeliğe odaklanmak
- **Kendi kendine organize olan takımlar kurmak ?**

2.1.Çevik Model Takımları



- Biraraya gelmiş,
- Kendi kendilerine organize olan,
- Çapraz fonksiyonlu,
- İşine odaklanmış,
- Hedefleri net olan,
- Teslim edilebilecek düzeyde ürün ortaya koyabilen
- Küçük(3-7 kişilik) gruplar.

2.1.Çevik Model Takımları

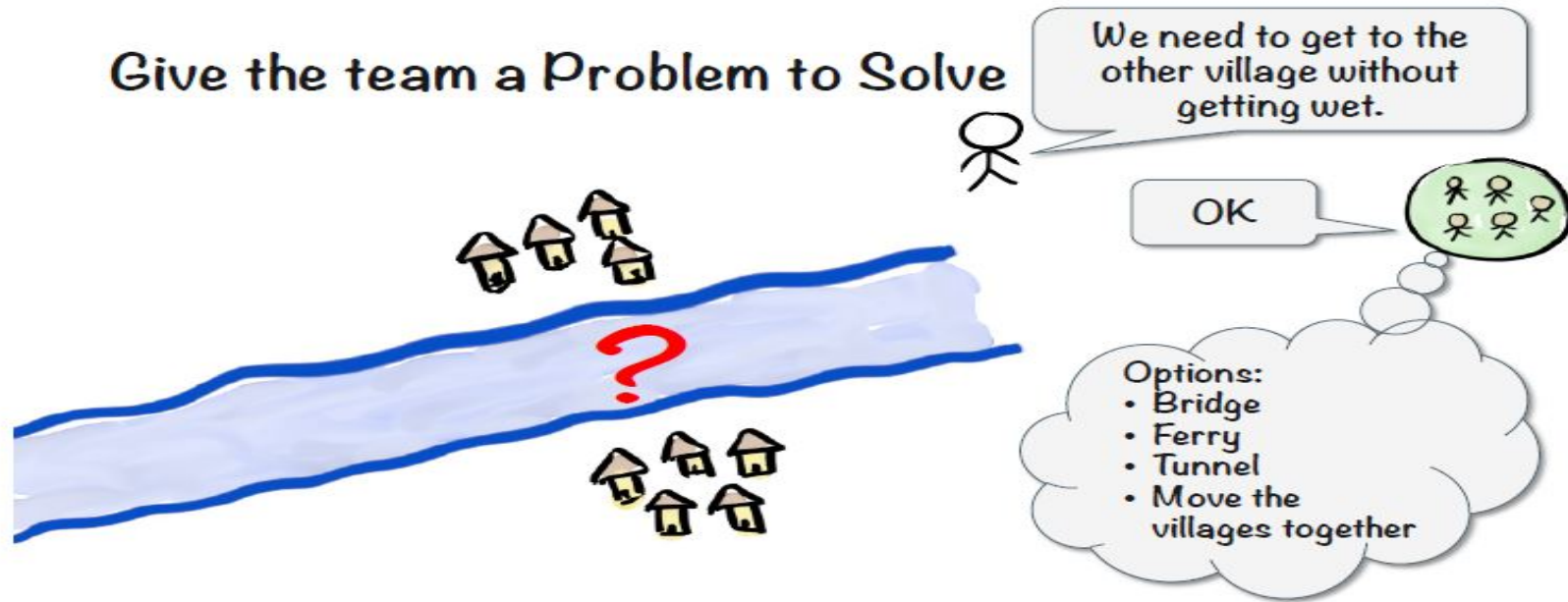
Takımlara çözümü söylemeyin!

Don't give the team a Solution to Build



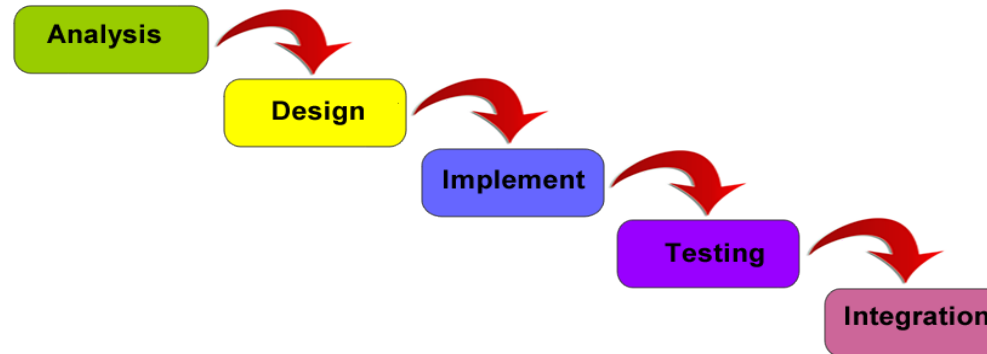
2.1.Çevik Model Takımları

Sorunu söyleyin, onlar çözümü üretsin!



3. Geleneksel Model vs. Agile

➤ **Çağlayan modeli** 2008 yılında dahi geçerliliğini koruyan bir modeldir ve çevik modellemeden farklılık gösterir. Bu model yazılım projesini baştan sona planlar. Gelişim, sunulabilir işler açısından ölçülür: talep açıklamaları, tasarım dokümanları, test planları, kod incelemeleri vb. Bu durum belli aralıklara bölünmeye uygun değildir ve ilerideki değişikliklere uyum gösterilemez.



Waterfall Model

3. Geleneksel Model vs. Agile



- Geleneksel Yöntemler
- Müşteriler ne istediğini iyi bilir.
- Geliştiriciler neyi, ne şekilde üreteceklerini iyi bilir.
- Bu yol boyunca hiç birşey değişmeyecektir.

3. Geleneksel Model vs. Agile

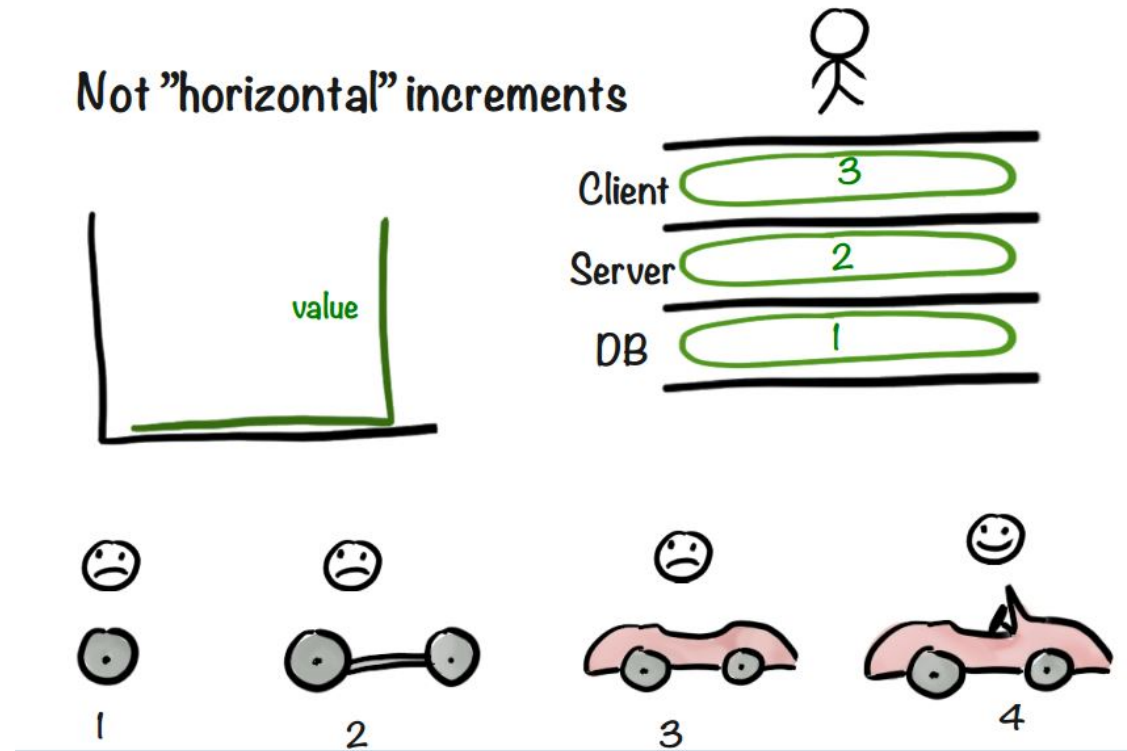


➤ Çevik Yöntemler

- Müşteriler ne istediğini keşfeder.
- Geliştiriciler neyi nasıl üreteceğini keşfeder.
- Bu yol boyunca bir çok değişiklik yapılabilir.

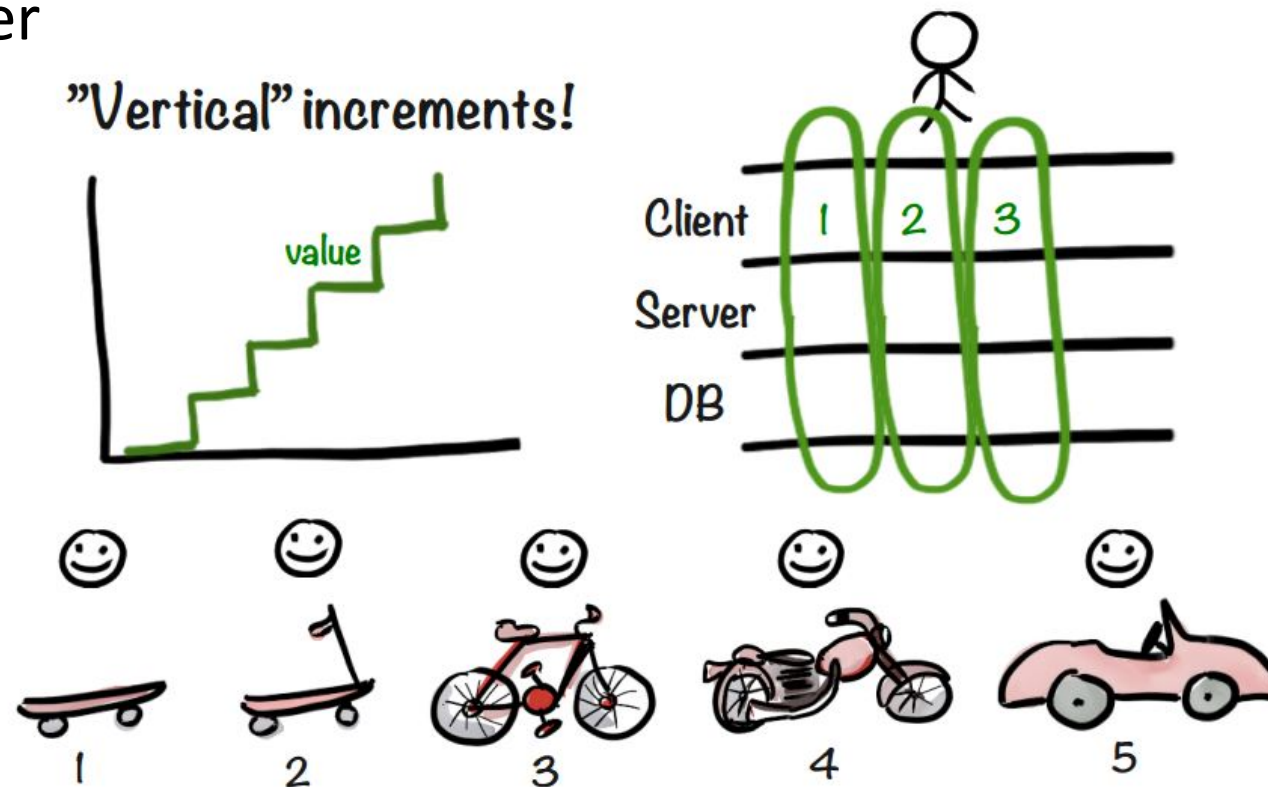
3. Geleneksel Model vs. Agile

Geleneksel Yöntemler

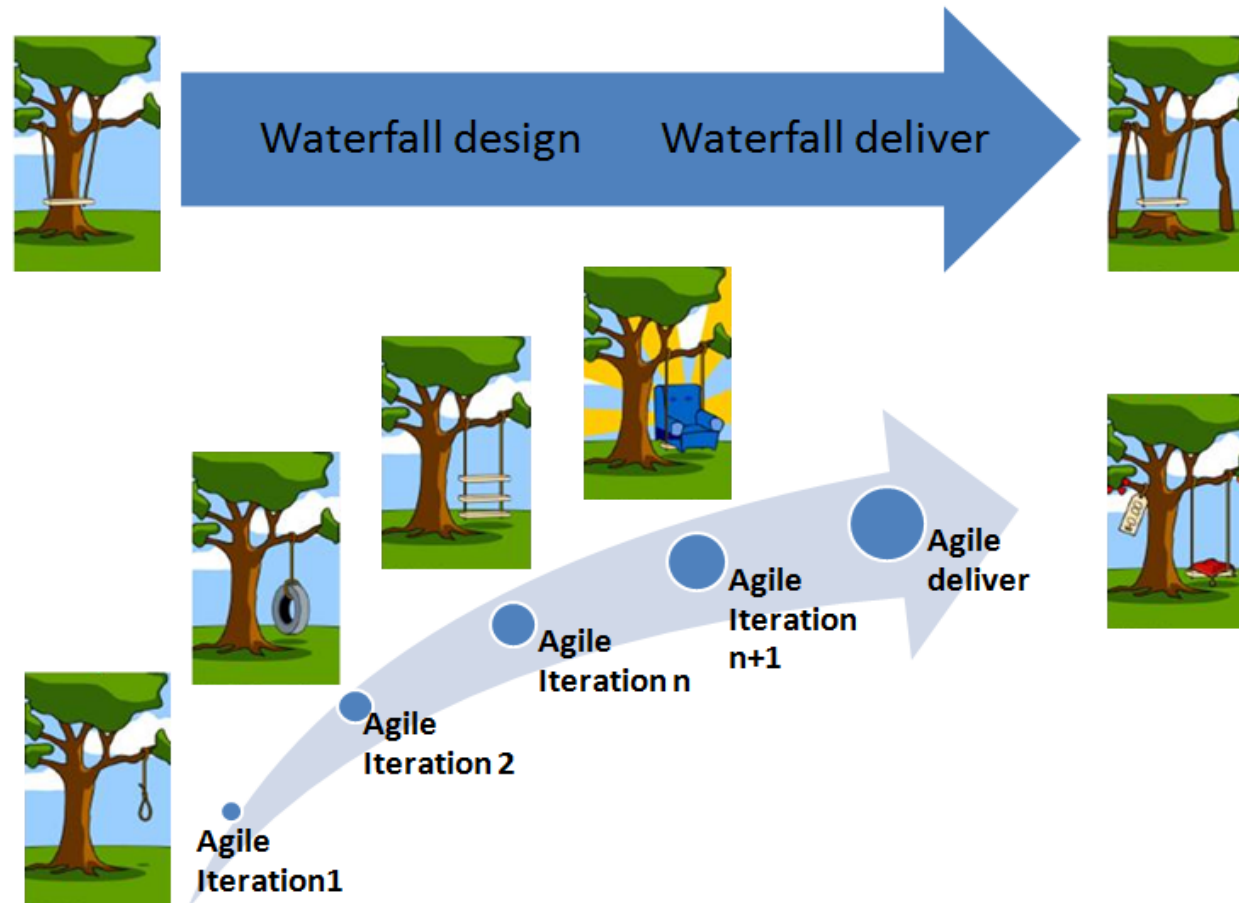


3. Geleneksel Model vs. Agile

Çevik Yöntemler

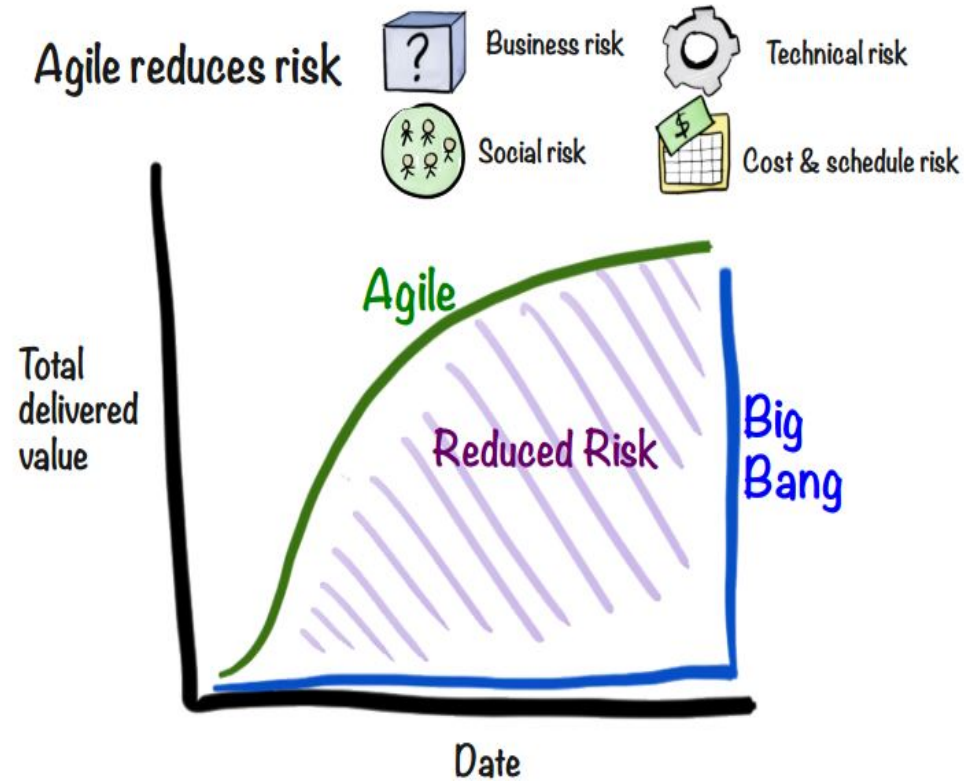


3. Geleneksel Model vs. Agile

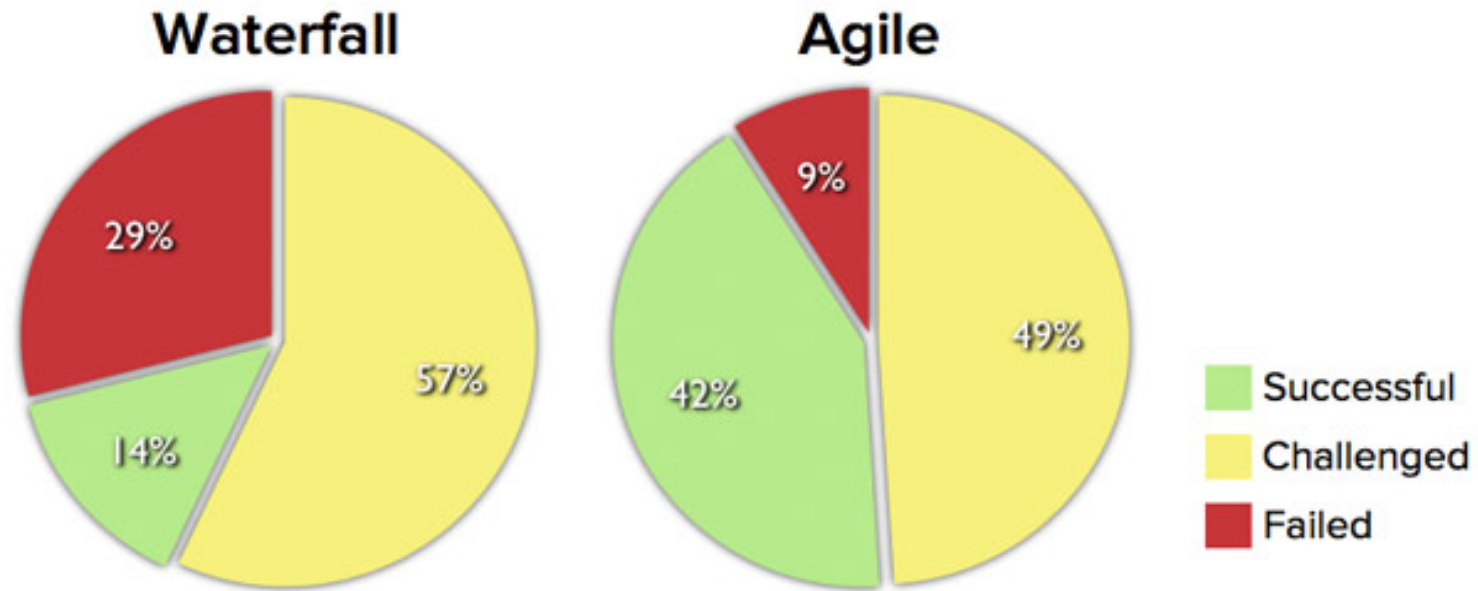


3. Geleneksel Model vs. Agile

Çevik modeller riski azaltır!



3. Geleneksel Model vs. Agile

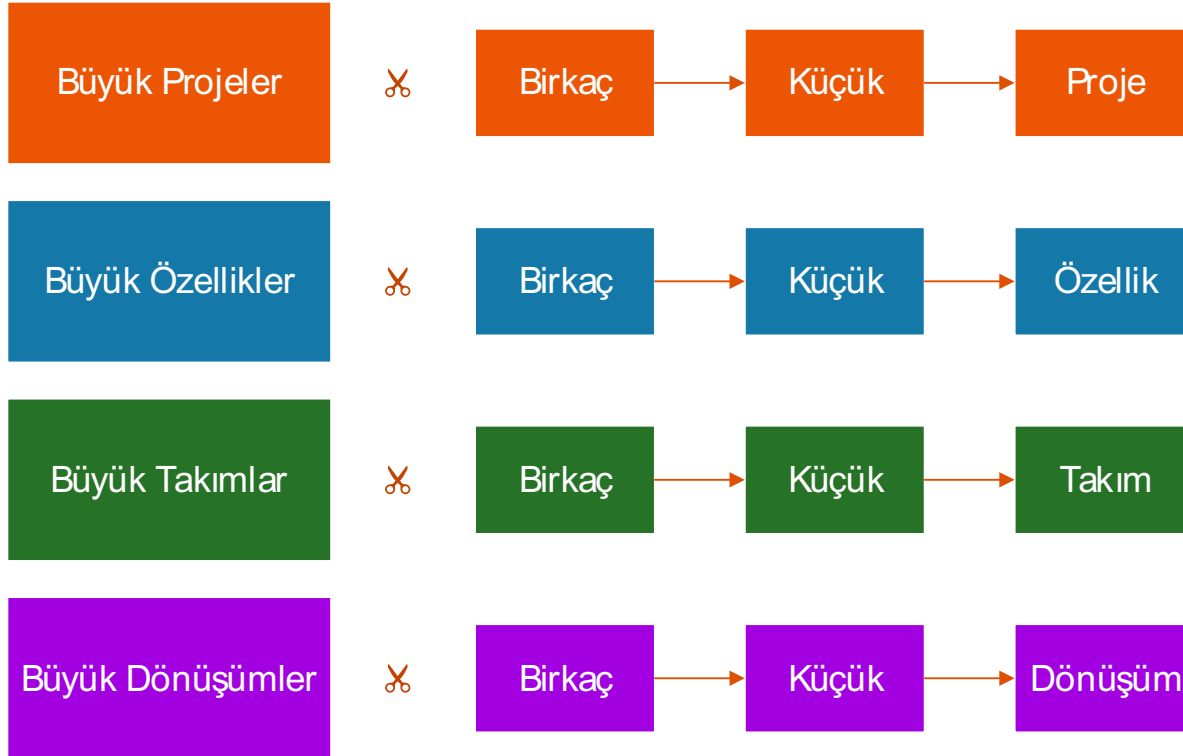


Source: The CHAOS Manifesto, The Standish Group, 2012.

3. Geleneksel Model vs. Agile

Ölçüm	Çevik Modelleme	Çağlayan Modeli
Planlama ölçeği	Kısa dönemlik	Uzun dönemlik
Müşteri ile geliştirici arasındaki mesafe	Kısa	Uzun
Özelleştirme ve uygulama arasındaki zaman	Kısa	Uzun
Sorunları keşfetmek için zaman	Kısa	Uzun
Proje tamamlanma riski	Düşük	Yüksek
Değişikliklere uyum yeteneği	Yüksek	Düşük

4. Değerlendirme



Sonuç

➤ Büyüklük kötüdür, onu parçalara ayırın. Böylece daha başarılı projeler geliştirebilirsiniz.

4. Değerlendirme

➤ 3 somut değişiklik yapın!

➤ 1. Gerçek takımlar oluşturun

- Küçük, çapraz fonksiyonlu, kendi kendine organize olabilen

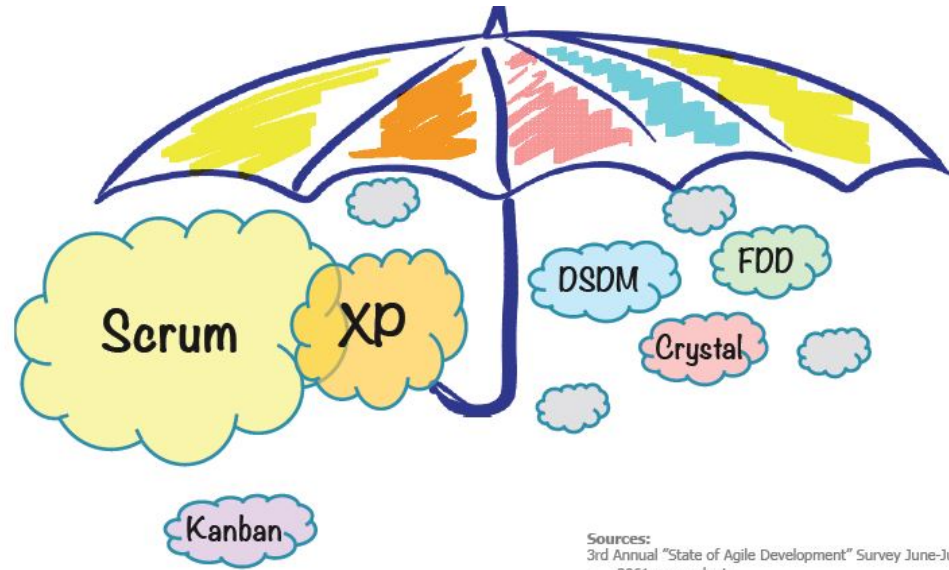
2. Sık sık teslimat yapın

- Normal olarak ortalama her 3 haftanın sonunda
- Ek olarak projenin tüm çeyreklerinin sonunda

3. Gerçek kullanıcıları dahil edin.

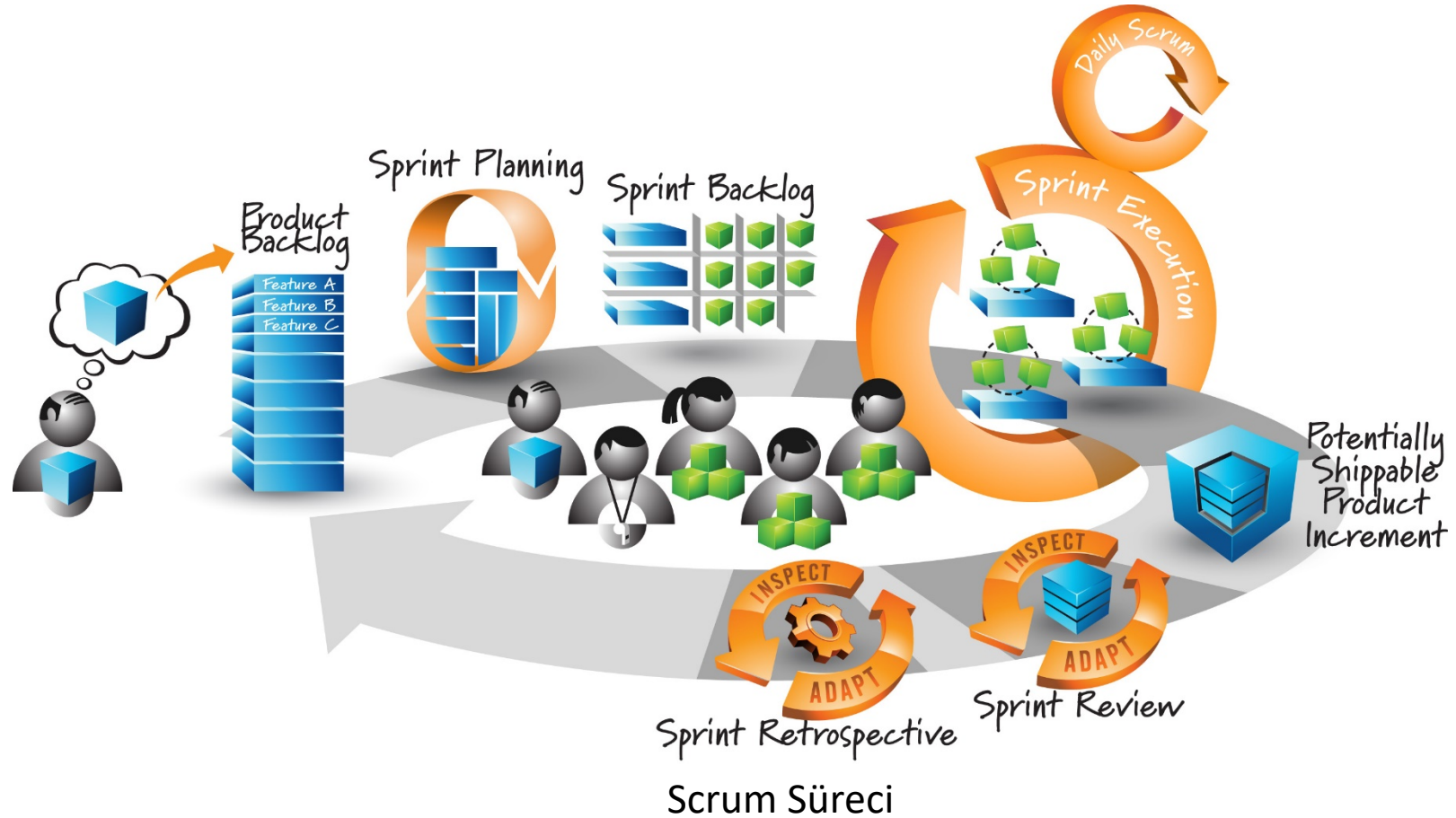
- Takım ve kullanıcılar arasında doğrudan ve hızlı geri dönüşler

5. Çevik Yazılım Şemsiyesi

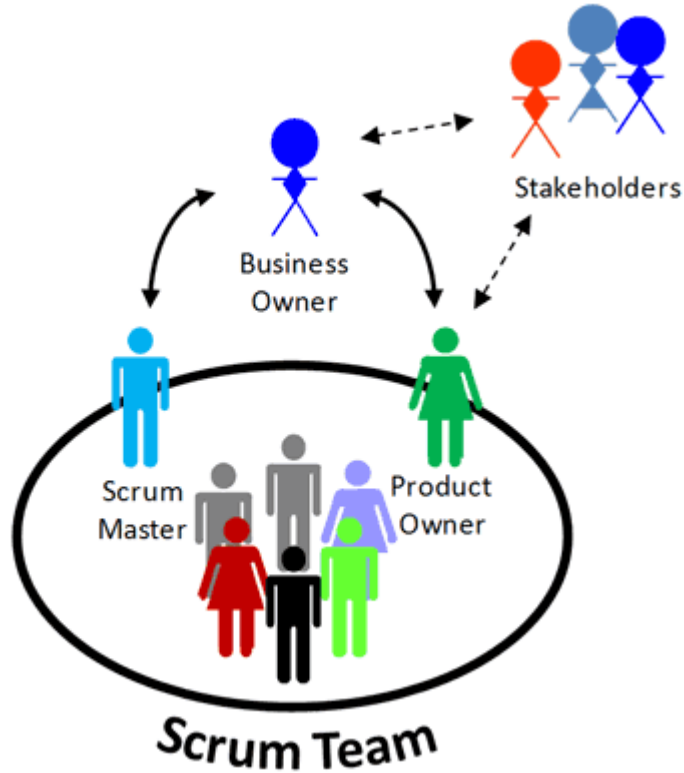


Sources:
3rd Annual "State of Agile Development" Survey June-July 2008
• 3061 respondents
• 80 countries

6. Scrum Modeli



6. Scrum Modeli



➤ **Scrum Takımı:** Ürün Sahibi, Geliştirme Ekibi ve Scrum Master'dan oluşur. Takım kendi kendini örgütler. Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar. Scrum takım modeli esneklik, yaratıcılık ve verimliliği optimize etmek için tasarlanmıştır.

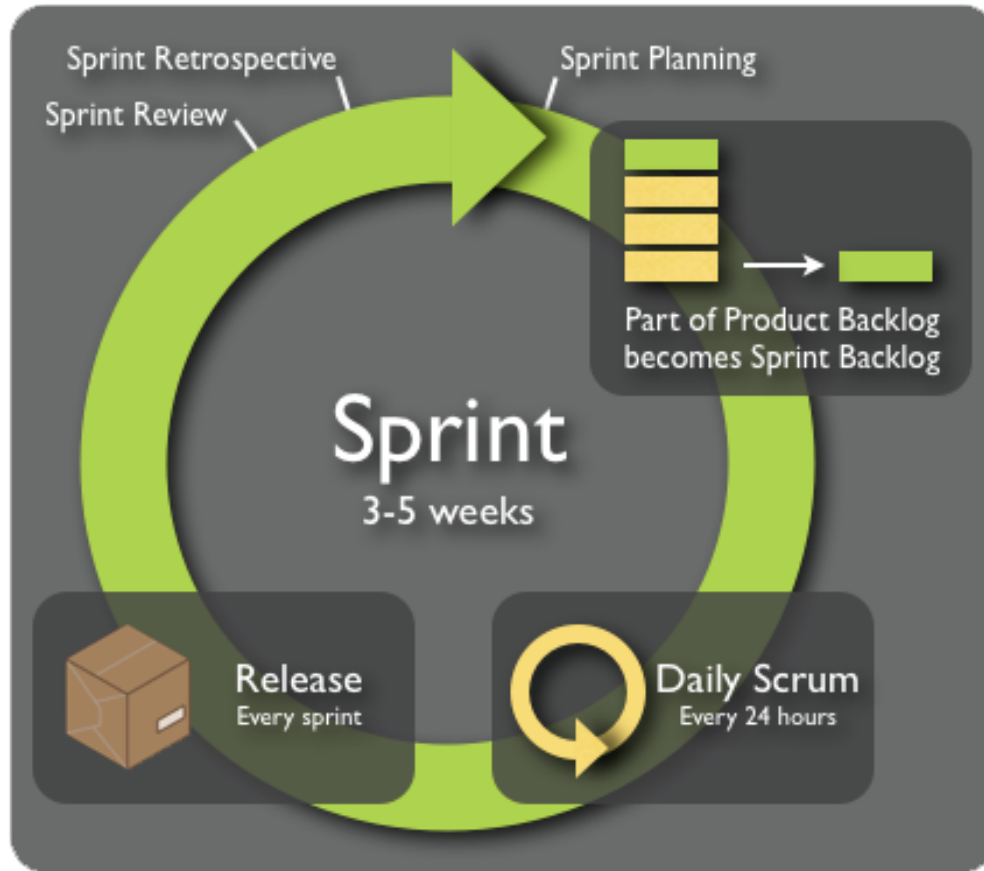
6. Scrum Modeli



Backlog:

- Müşteriden ve son kullanıcıdan gelen gereksinimleri içerir.
- "Ne yapacağız" sorusunun yanıtını içerir.
- Herkese açık ve herkes tarafından müdahale edilebilir.
- Risk, iş değeri, zaman gibi kavramlara göre ürün sahibi tarafından sıralandırılır.
- User Story'lerden oluşur.

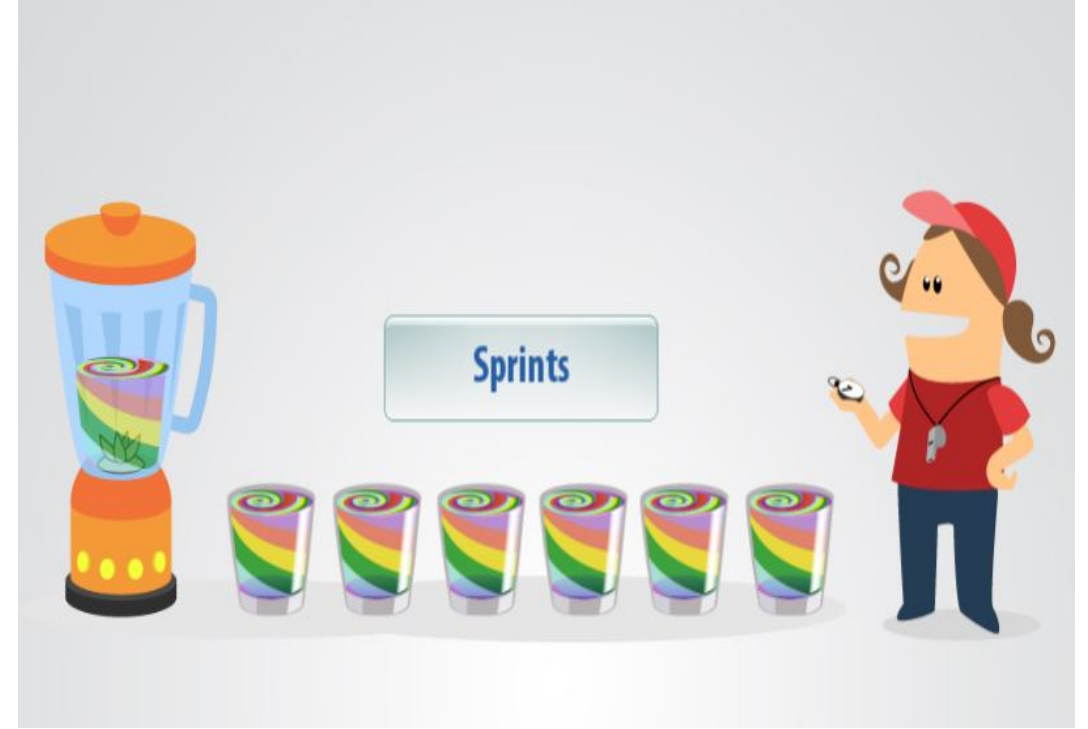
6. Scrum Modeli



➤ Sprint

- Belirli bir süreye sahiptir.
- Sonunda ortada değeri olan bir çıktı olmalıdır.
- Toplantılarla içerik belirlenir.
- Sprint süresi boyunca her gün toplantılar yapılır.

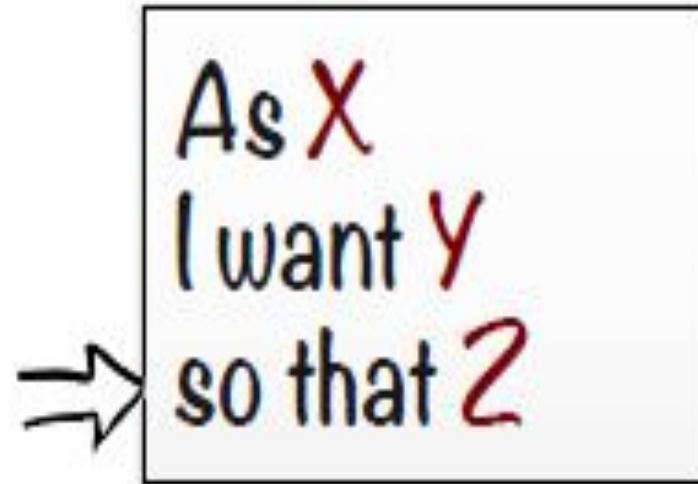
6. Scrum Modeli



Sprint Gösterimi

6. Scrum Modeli

- **User Story:** Müşteri, son kullanıcı veya ürün sahibi için değerli olan ve anlam ifade eden genellikle fonksiyonel özelliklerin belirtildiği ifadelerdir.



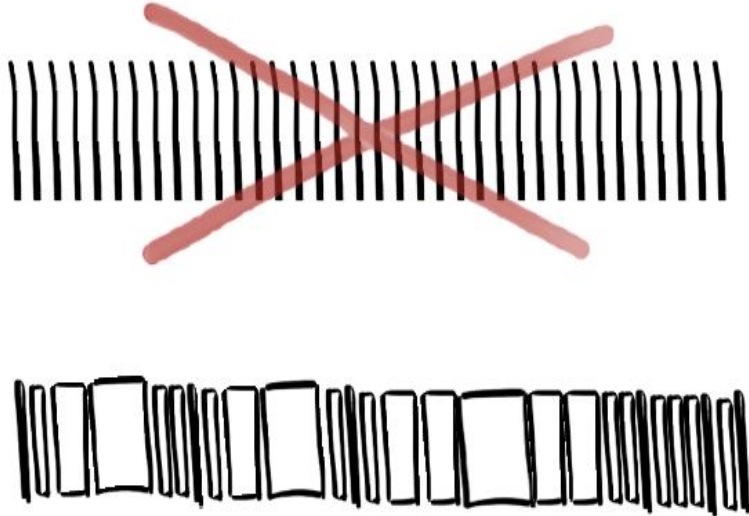
6. Scrum Modeli

- **Örnek User Story:** Online alışveriş yapan biri **olarak**, alışverişe daha sonra devam edebileyim **diye**, alışveriş kartımın kaydedilmesini **istiyorum**.

As online buyer
I want to save my shopping cart
so that I can continue shopping later

6. Scrum Modeli

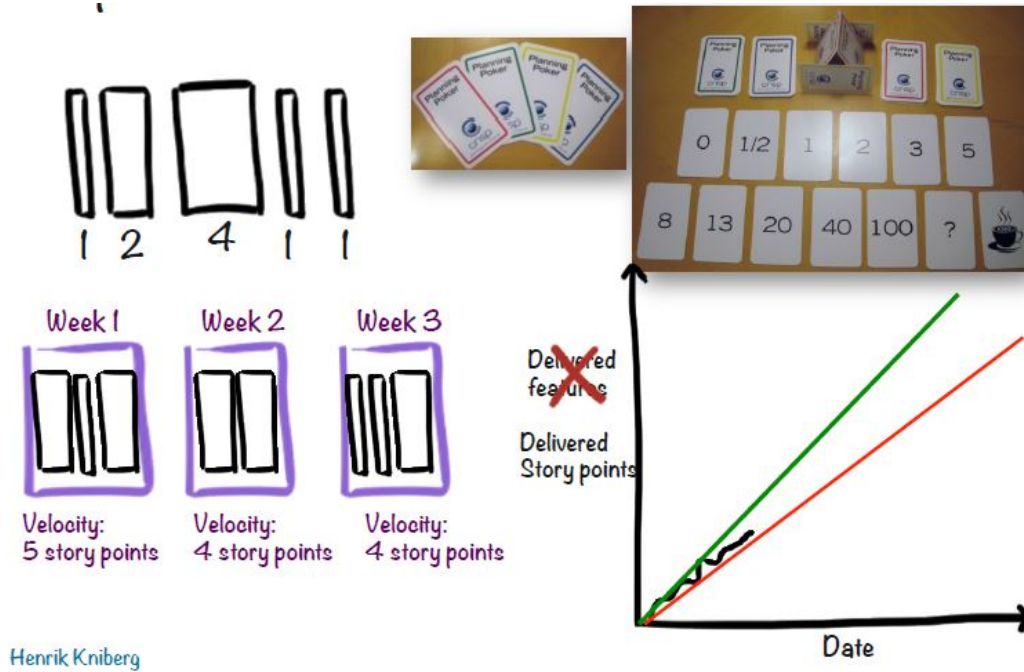
Fact: Features have different sizes



➤ Her bir user story farklı bir boyuttur. Somut olarak bakarsak, bir projedeki her bir gereksinim için gereken iş gücü ve zaman aynı değildir. Bu sebeple ürün backlogları sprintlere bölünürken, user storylerin boyut ve öncelikleri göz önünde bulundurulur. Örneğin bir sprint 3 user story içerirken diğeri daha küçük boyutlarda 5 user story içerebilir. **Peki boyutları nasıl belirleyeceğiz?**

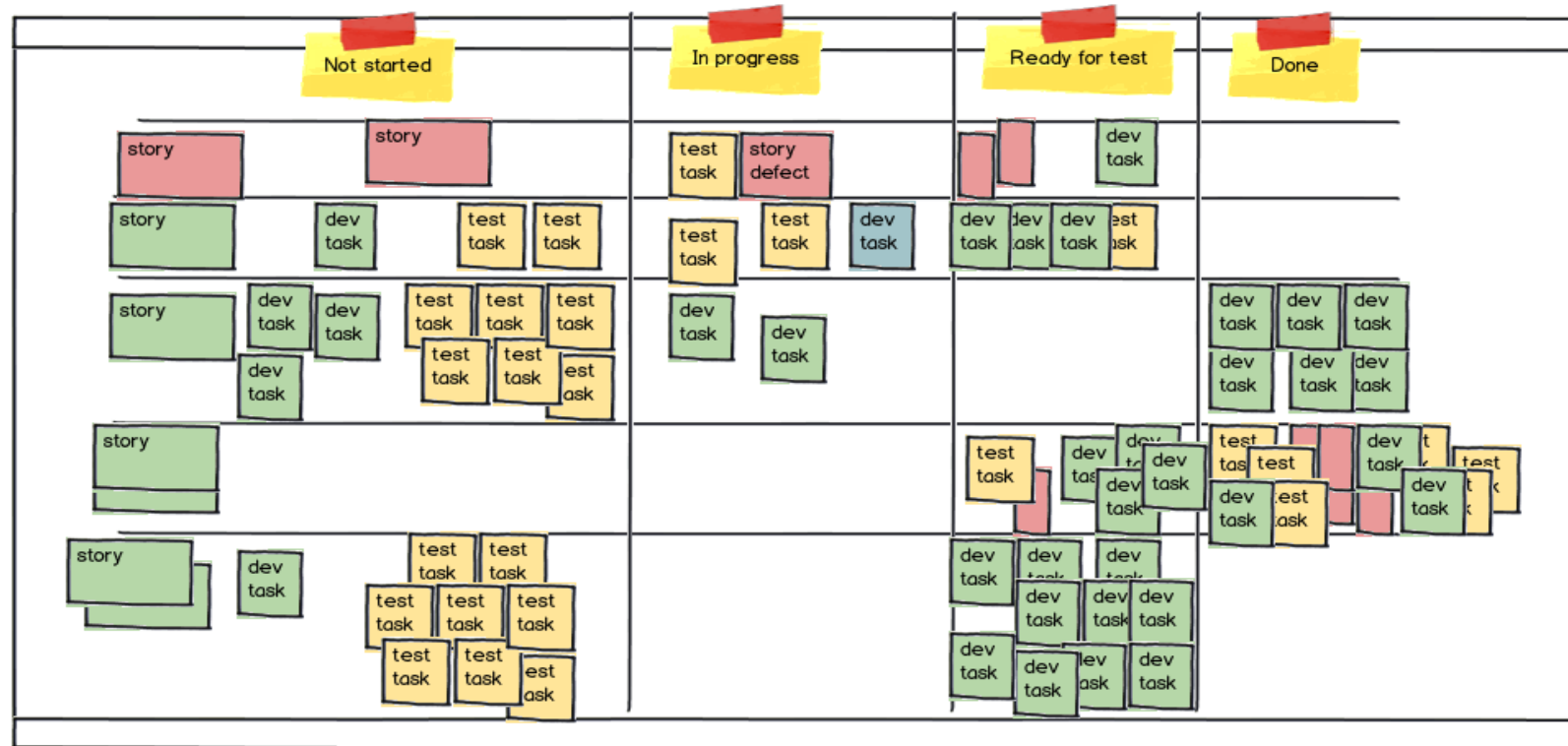
İş görüldüğü gibi değil!

6. Scrum Modeli



➤ **Poker Kartları:** Scrum takım üyeleri bir araya gelir. Scrum master bir user story okur. Takımdaki her bir üye user story için uygun gördüğü poker kartlarından birini seçer. Herkes kartları seçtikten sonra tüm kartlar açılır ve değerlendirilir. Böylece herkesin ortak görüşü sonunda user story'lerin büyüklüğü belirlenir.

6. Scrum Modeli



created with Balsamiq Mockups - www.balsamiq.com

Scrum Board

6. Scrum Modeli

SCRUM

- Sprint (2 hafta-1 ay)
- Sprintler en son halini aldıktan, toplantı yapıldıktan sonra değişmez.
- Özellikler geliştiriciler tarafından derecelendirilir.
- Herhangi bir mühendislik pratiği tanımlamaz.

XP (EXTREME PROGRAMMING)

- Sprint (1 yada 2 hafta)
- Sprintler değişebilir.
- Özellikler ürün sahibi tarafından derecelendirilir.
- Mühendislik pratikleri tanımlar. Eşli programlama, otomatik test, basit dizayn vs.

Agile is...

Early delivery of business value

Less bureaucracy

Sorularınız



Çalışma Soruları

1. Agile modelinin geliştirilmesine neden ihtiyaç duyulmuştur? Açıklayınız.
2. Agile takımları kimlerden oluşur ve iş paylaşımları nasıldır?
3. Agile modelinin diğer modellerden farkları nelerdir? Üzerinde durduğu temel noktalar nelerdir?
4. User story, backlog ve sprint kavramlarını açıklayınız.
5. Agile ile geliştirilmiş büyük yazılım projelerine örnekler veriniz.
6. Scrum haricindeki diğer çevik yazılım geliştirme yöntemlerini açıklayınız.

Kaynaklar

- [1] Martin, Micah, and Robert C. Martin. *Agile principles, patterns, and practices in C#*. Pearson Education, 2006.
- [2] Kniberg, Henrik. "Scrum and XP from the Trenches." *Lulu. com* (2007).
- [3] Kniberg, Henrik. " What is agile" , 2013
- [4] http://en.wikipedia.org/wiki/Agile_software_development
- [5] <http://www.mshowto.org/microsoft-visual-studio-team-foundation-server-nedir.html>
- [6] www.kurumsaljava.com/download/10/



YMH311-Yazılım Tasarım ve Mimarisi Planlama ve Sistem Çözümleme

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Proje Planlama Aşamaları için Gerekli Koşullar4

Proje Maliyet Kestirim Yöntemleri14

Sistem Çözümleme40

Gereksinim Çözümleme Çalışması44

Gereksinim Veri Toplama Yöntemleri55

Veri Modelleme Yöntemleri66

Veri Akış Diyagramı71

Süreç Tanımlama Dili77

Amaçlar



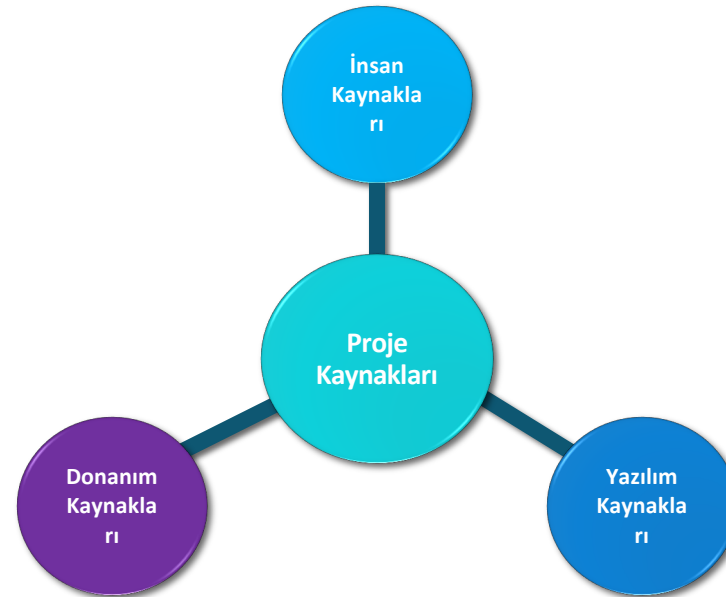
- Yazılımda Planlamanın nasıl yapılacağını öğrenmek,
- Planlama aşamasında gerekli olan yöntemleri öğrenmek,
- Planlama aşamasında kaynakların kullanımını görmek,
- Proje yapımında maliyet kestirimlerini öğrenip projede uygulamak,
- COCOMO Modeli hakkında bilgi sahibi olmak,
- Sistem Çözümleme aşamasını öğrenmek,
- Sistem çözümleme aşamasında kullanılan yapıları kavramak,
- Ayrıntılı bir sistemi çözümleyerek yeni bir sistem oluşturmak,
- Proje için gereksinimlerin ne anlama geldiğini öğrenmek.
- Veri akış diyagramlarının ne tür amaca göre oluşturulduğunu görmek,
- Genel olarak projemizde Tasarım aşamasından önceki Planlama ve Sistem Çözümleme aşamalarını kavramak.

Planlama

- Yazılım geliştirme sürecinin ilk aşaması,
- Başarılı bir proje geliştirebilmek için projenin tüm resminin çıkarılması işlemi,
- Proje planlama aşamasında yapılan işlemler
 - Proje Kaynaklarının Belirlenmesi
 - Proje Maliyetlerinin Kestirilmesi
 - Proje Ekip Yapısının Oluşturulması
 - Ayrıntılı Proje Planının Yapılması
 - Projenin İzlenmesi
- Proje planı tüm proje süresince sürekli olarak kullanılacak, güncellenecek ve gözden geçirilecek bir belgedir.



Proje Kaynakları



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

İnsan Kaynakları

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

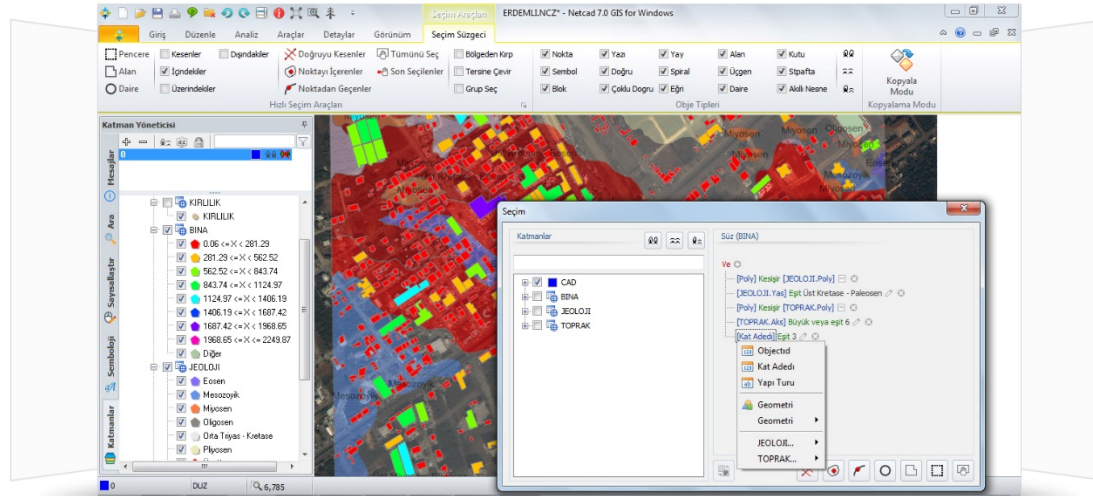
Proje Yöneticisi	Donanım Ekip Lideri
Yazılım Ekip Lideri	Donanım Mühendisi
Web Tasarımcısı	Ağ Uzmanı
Sistem Tasarımcısı	Yazılım Destek Elemanı
Programcı	Donanım Destek Elemanı
Sistem Yöneticisi	Eğitmen
Veri Tabanı Yöneticisi	Denetleyici
Kalite Sağlama Yöneticisi	Çağrı Merkezi Elemanı

Donanım Kaynakları

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

Yazılım Kaynakları

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



Yazılım Kaynakları

■ İş sistemleri planlama araçları

- İş akış yapısının üst modelinin üretilmesinde kullanılır.
- Bilgi akışı, bilgi yapısı iş birimlerindeki tıkanıklıklar bu araçlar kanalıyla ortaya çıkarılır.

■ Proje yönetim araçları

- Yönetici tarafından, projede yapılan işlerin izlenmesi, kaynak ataması, proje iş yapısının üretilmesi, gözlenen değerlerin işlenmesini sağlayan araçlar.

■ Analiz ve tasarım araçları

- Kullanılan modelleme tekniklerini ayrı ayrı ya da bütünleşik olarak uygulayan araçlar. Üretilen modelin kalitesinin ölçülmesi

■ Programlama araçları

- Derleyiciler, nesne-tabanlı programlama araçları, görsel programlama platformları.

Yazılım Kaynakları

■ Test araçları

- Yazılımı doğrulama ve geçerleme işlemlerinde kullanılır. Test verisi üreticiler, otomatik test yordamları, ...

■ Prototipleme ve simülasyon araçları

- Geliştirmenin erken aşamalarında kullanıcıya, sonuç ürünün çalışması ile ilgili fikir veren ve yönlendiren araçlar.

■ Bakım araçları

- Programın bakımını kolaylaştıran, bir kaynak koddan program şemalarının üretilmesini, veri yapısının ortaya çıkarılmasını sağlayan araçlar.

■ Destek araçları

- İşletim sistemleri, ağ yazılımları, e-posta ve ortam yönetim araçları.

Proje Maliyetleri

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

- **Kullanılan Unsurlar**

- Geçmiş projelere ilişkin bilgiler
- Proje ekibinin deneyimleri
- İzlenen geliştirme modeli

birden çok kez uygulanabilir



Proje Maliyetleri

Maliyet yönetimi sayesinde;

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



Gözlemlenebilecek değerler

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



Maliyet Kestirim Yöntemleri

1. Projenin boyut türüne göre

- Proje büyüklüğünü kestiren yöntemler
- Proje zaman ve işgücünü kestiren yöntemler

2. Projelerin büyüklüğüne göre

- Makro yöntemler (büyük boyutlu projeler 30 kişi-yıl)
- Mikro Yöntemler (orta ve küçük boyutlu projeler)

3. Uygulanış biçimlerine göre

- Çok yalın düzeyde
- Orta ayrıntılı düzeyde
- Çok ayrıntılı düzeyde

4. Değişik aşamalarda kullanılabilirlik

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

5. Yöntemlerin yapılarına göre

- Uzman deneyimine gereksinim duyan
- Önceki projelerdeki bilgileri kullanan yöntemler



İşlev Noktaları Yöntemi

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

Problemin bilgi ortamının incelenmesi

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

Bunların ağırlık faktörleriyle çarpımları toplanarak, **Ayarlanmamış İşlev Nokta (AIN) sayısı** hesaplanır.

Problem Bilgi Ortamı Bileşenleri

Ölçüm Parametresi	Sayı	Ağırlık Faktörü				
		Yalın	Ortalama	Karmaşık		
Kullanıcı Girdi sayısı	?	3	4	6	=	
Kullanıcı Çıktı sayısı	?	4	5	7	=	
Kullanıcı Sorgu Sayısı	?	3	4	6	=	
Kütük Sayısı	?	7	10	15	=	
Dışsal Araryüz Sayısı	?	5	7	10	=	
Toplam Sayı					=	

Problemin teknik karmaşıklığının incelenmesi

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



Cevaplar 0 ile 5 arasında puanlandırılır.

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

İşlev Noktası Sayısı Hesaplama

■ $\dot{I}N = A\dot{I}N * (0,65 * 0,01 * TKF)$

Değişik amaçlarla kullanılabilir

- **Üretkenlik** = $\dot{I}N / \text{Kişi-Ay}$
- **Kalite** = $\text{Hatalar} / \dot{I}N$
- **Maliyet** = $\text{\$/} \dot{I}N$

Satır Sayısı Kestirimi

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

İN=300 ise ve Nesne Tabanlı bir dil (SmallTalk) kullanılıyor ise

Satır Sayısı=300*30

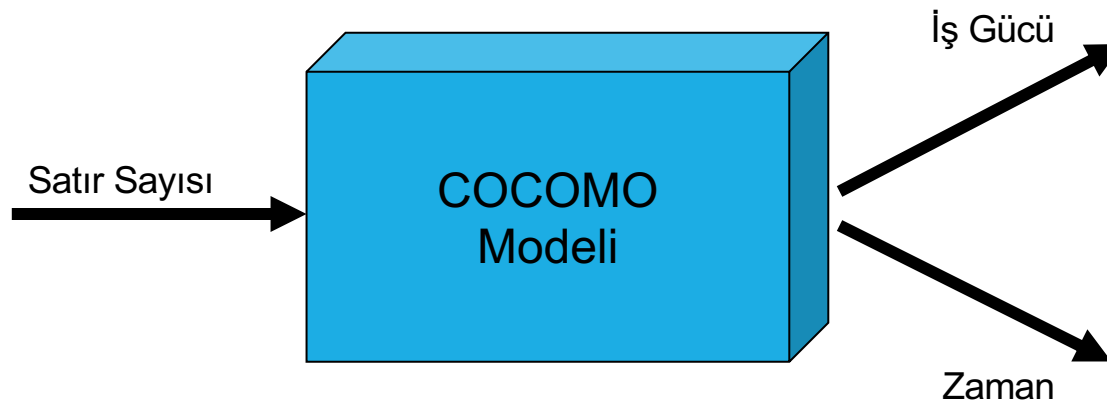
olarak hesaplanır.

Etkin Maliyet Modeli

- COCOMO 1981 Boehm
- Mikro maliyet kestirim modeline örnektir.
- Kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir:
 - Temel Model
 - Ara Model
 - Ayrıntı Model



COCOMO Modeli



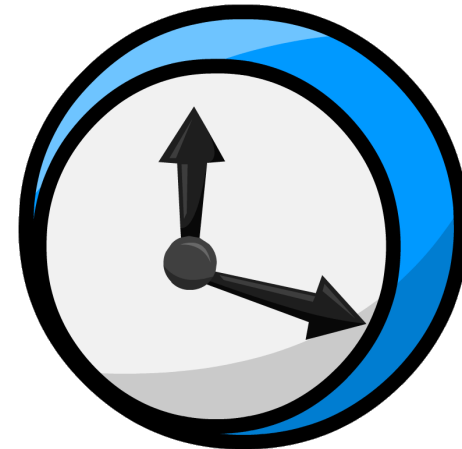
COCOMO Formülleri

İs Gücü (K) $K=a*S^b$

Zaman (T) $T=c*K^d$

a,b,c,d : her bir model için farklı katsayılar

S : bin türünden satır sayısı



Proje Sınıfları

■ **Ayrık Projeler:**

- Boyutları küçük,
- Deneyimli personel tarafından gerçekleştirilmiş
- LAN üzerinde çalışan insan kaynakları yönetim sistemi gibi

■ **Yarı Gömülü:**

Hem bilgi boyutu hem donanım sürme boyutu olan projeler

■ **Gömülü Projeler:**

Donanım sürmeyi hedefleyen projeler (pilotsuz uçağı süren yazılım - donanım kısıtları yüksek)

Temel Model

- Küçük-orta boy projeler için hızlı kestirim yapmak amacıyla kullanılır
- **Dezavantajı:** Yazılım projesinin geliştirileceği ortam ve yazılımı geliştirecek ekibin özelliklerini dikkate almaz
- **Avantajı:** Hesap makinesi ile kolaylıkla uygulanabilir



Temel Model

■ **Ayrık Projeler**

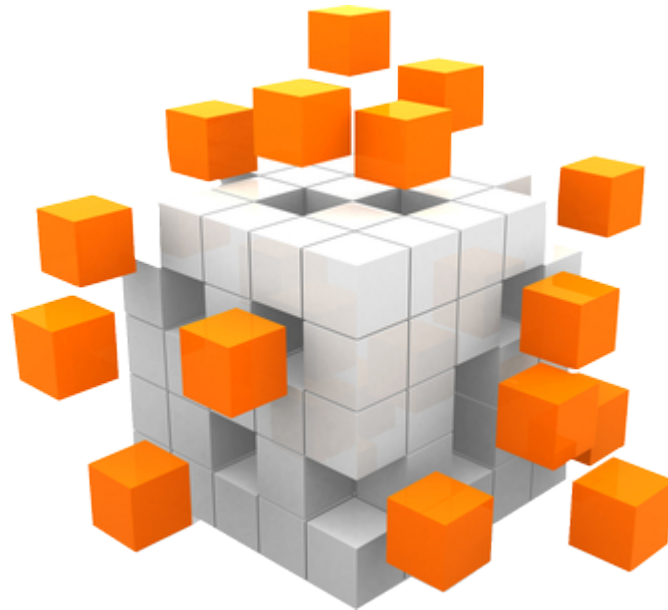
- İş Gücü $K=2.4*S^{1,05}$
- Zaman $T=2.5*K^{0,38}$

■ **Yarı Gömülü Projeler**

- İş Gücü $K=3,0*S^{1,12}$
- Zaman $T=2.5*K^{0,35}$

■ **Gömülü Projeler**

- İş Gücü $K=3,6*S^{1,20}$
- Zaman $T=2.5*K^{0,32}$



Ara Model

- Temel modelin eksikliğini gidermek amacıyla oluşturulmuştur.
- Bir yazılım projesinin zaman ve iş gücü maliyetlerinin kestiriminde;
 - Proje ekibinin özelliklerini,
 - Proje geliştirmede kullanılacak araçları, yöntem ve ortamı dikkate alır.
- Üç Aşamadan oluşur:
 - İş gücü hesaplama
 - Maliyet çarpanı hesaplama
 - İlk iş gücü değerini düzeltme



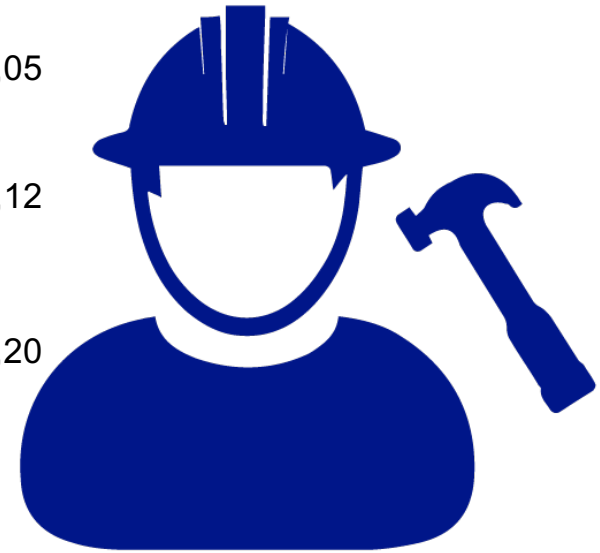
İş Gücü Hesaplama

- Ayrık Projeler
- Yarı Gömülü Projeler
- Gömülü Projeler

$$K=3.2*S^{1,05}$$

$$K=3,0*S^{1,12}$$

$$K=2.8*S^{1,20}$$



Maliyet Çarpanı Hesaplama

- Maliyet Çarpanı 15 maliyet etmeninin çarpımı sonucudur.

$$C = C1 * C2 * C3 * ... * C15$$



Maliyet Etmenleri

Maliyet etmeni		Seenekler					
		ok Düşük	Düşük	Normal	Yüksek	ok Yüksek	Olduka Yüksek
Ürün Özellikleri	RELY	0,75	0,88	1,00	1,15	1,40	-
	DATA	-	0,94	1,00	1,08	1,16	-
	CPLX	0,70	0,85	1,00	1,15	1,30	1,65
Bilgisayar Özellikleri	TIME	-	-	1,00	1,11	1,30	1,66
	STOR	-	-	1,00	1,06	1,21	1,56
	VIRT	-	0,87	1,00	1,15	1,30	-
	TURN	-	0,87	1,00	1,07	1,15	-
Personel Özellikleri	ACAP	1,46	1,19	1,00	0,86	0,71	-
	AEXP	1,29	1,13	1,00	0,91	0,82	-
	PCAP	1,42	1,17	1,00	0,86	0,70	-
	VEXP	1,21	1,10	1,00	0,90	-	-
	LEXP	1,14	1,07	1,00	0,95	-	-
Proje Özellikleri	MODP	1,24	1,10	1,00	0,91	0,82	-
	TOOL	1,24	1,10	1,00	0,91	0,83	-
	SCED	1,23	1,08	1,00	1,04	1,10	-

Ürün Özellikleri

- **Rely:** Yazılımın güvenilirliği
- **Data:** Veri Tabanının Büyüklüğü.
Burada program büyüklüğüne oranı dikkate alınır.
- **Cplx:** Karmaşıklığı.



Bilgisayar Özellikleri

- **Time:** İşletim zamanı kısıtı
- **Stor:** Ana Bellek Kısıtı
- **Virt:** Bilgisayar Platform Değişim Olasılığı.
Bellek ve Disk kapasitesi artırımı,
CPU Upgrade
- **Turn:** Bilgisayar İş Geri Dönüş Zamanı.
Hata düzeltme süresi.



Personel Özellikleri

- **Acap:** Analist Yeteneği:
Deneyim, Birlikte çalışabilirlik.
- **Aexp:** Uygulama Deneyimi.
Proje ekibinin ortalama tecrübesi.
- **Pcap:** Programcı Yeteneği.
- **Vexp:** Bilgisayar Platformu Deneyimi.
Proje ekibinin geliştirilecek platformu tanıma oranı.
- **Lexp:** Programlama dili deneyimi.



Proje Özellikleri

- **Modp:** Modern Programlama Teknikleri.
 - Yapısal programlama,
 - Görsel programlama,
 - Yeniden kullanılabilirlik.
- **Tool:** Yazılım Geliştirme araçları kullanımı.
 - CASE araçları
 - Metin düzenleyiciler
 - Ortam yönetim araçları
- **Sced:** Zaman Kısıtı.



İlk İşgücü değerini Düzeltme

■ $K_d = K * C$

K_d =Düzeltilmiş İşgücü

- * Temel Formüldeki Zamanla formülü kullanılarak zaman maliyeti hesaplanır.



Ayrıntı Modeli

Temel ve ara modele ek olarak iki özellik taşır.

- Aşama ile ilgili işgücü katsayıları: her aşama için (planlama, analiz, tasarım, geliştirme, test etme) farklı katsayılar, karmaşıklık belirler
- Üç düzey ürün sıra düzeni: yazılım maliyet kestiriminde
 - Modül
 - Altsistem
 - Sistem

Sıra düzenini dikkate alır.



Proje Ekip Yapısı Oluřturma

- PANDA proje Ekip yapısı temel olarak her proje biriminin doğrudan proje yönetimine baėlı olarak çalışması ve işlevsel bölümlenme esasına göre oluşturulur. Temel bileşenler
 - Proje Denetim Birimi
 - Proje Yönetim Birimi
 - Kalite Yönetim Birimi
 - Proje Ofisi
 - Teknik Destek Birimi
 - Yazılım Üretim Eşgüdüm Birimi
 - Eğitim Birimi
 - Uygulama Destek Birimi



Yüklenici Proje Ekip Yapısı

- **Proje Denetim Birimi:** En üst düzey yönetimlerin proje ile ilgisinin sürekli sıcak tutulması ve onların projeye dahil edilmesi
- **Proje Yönetim Birimi:** Proje yönetiminden en üst düzeyde sorumlu birim. Proje boyutuna göre bir yada daha çok yöneticiden oluşur.
- **Kalite Yönetim Birimi:** Projenin amacına uygunluğunu üretim süreci boyunca denetler ve onaylar
- **Proje Ofisi:** Her türlü yönetsel işlerden(yazışma, personel izleme) sorumlu birimdir.

Yüklenici Proje Ekip Yapısı

- **Teknik Destek Birimi:** Donanım, İşletim sistemi, Veri tabanı gibi teknik destek
- **Yazılım Üretim Eşgüdüm Birimi:** Yazılım Üretim Ekiplerinden oluşur(4-7 kişilik sayı fazla artmaz). Eğer birden fazla yazılım Üretim Ekibi varsa Ortak uygulama yazılım parçalarının geliştirilmesinden sorumlu Yazılım Destek Ekibi de olur.
- **Eğitim Birimi:** Proje ile ilgili her türlü eğitimden sorumludur.
- **Uygulama Destek Birimi:** Uygulama anında destek. (mesela telefonla)

İş Sahibi Proje Ekip Yapısı

- Proje Eşgüdüm Birimi
- Kalite Yönetim Birimi
- Proje Ofisi
- Teknik Altyapı izleme birimi
- Yazılım Üretim İzleme Birimi
- Eğitim İzleme Birimi
- Kullanıcı Eşgüdüm Birimi



Sistem Çözümleme

- Üretim sürecinin başlangıcıdır.
- Temel olarak mevcut sistemin nasıl çalıştığını araştırır.
- Temel hedef gereksinimlerinin saptanmasıdır.
- Mantıksal bir model
- Mutlaka bir model/yöntem kullanma zorunluluğu vardır.
- Yöntemler; veri modelleme ve süreç modelleme olmak üzere ikiye ayrılır.
- Çözümleme değişik açılardan değerlendirilir.

Gereksinim Nedir?

- Sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.
- Gereksinim sistemin yada işlevlerinin nasıl yerine getirileceği ile ilgili değildir. **Ne olduğu ile ilgilidir.**
 - hangi veri tabanı,
 - hangi tablolar,
 - ne kadar bellek kullanılıyor,bunlar tasarım ve gerçekleştirim aşamasında ele alınır.
- Gereksinim, kullanıcı ve tasarımcı ya da yazılım mühendisi ile ilgili olarak iki amaca yönelik olacak biçimde tanımlanmalıdır;
 - **Kullanıcılar**, geliştirilecek sistemin amaçları istenilen ölçüde tanımlanmış mı sorusuna yanıt ararken,
 - **Tasarımcılar** ise gereksinimlerin tasarıma dönüştürülebilme uygunluğunu ararlar.

Gereksinim Çeşitleri

İşlevsel
Gereksinimler



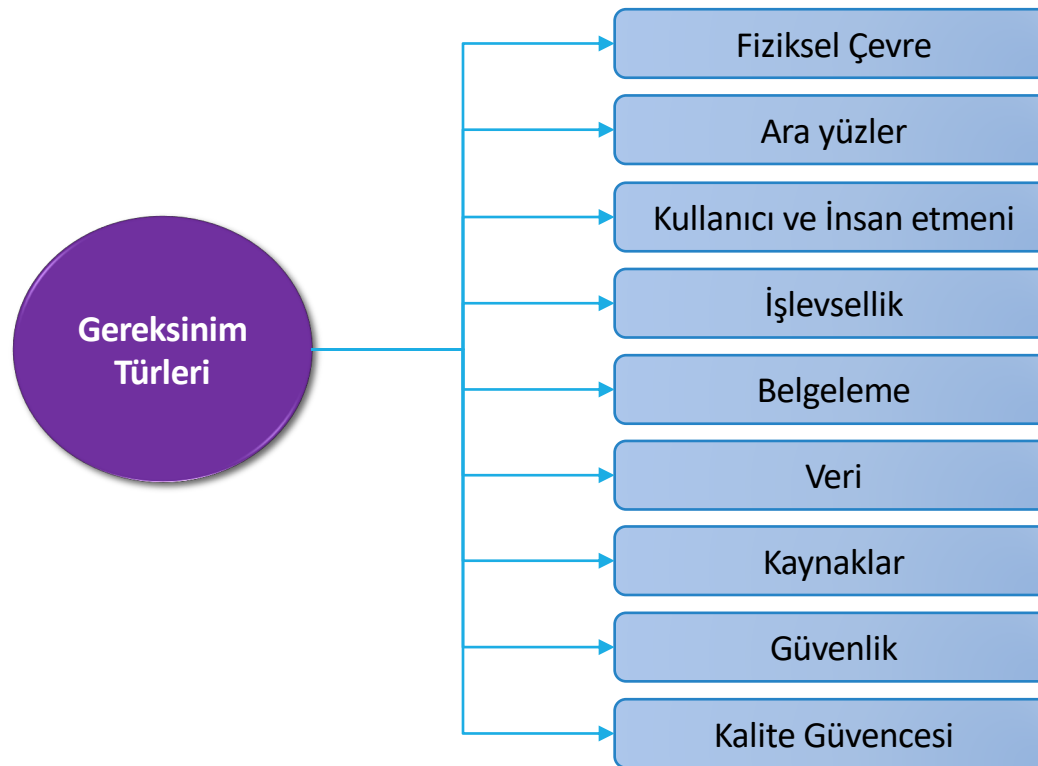
- Sistem ile çevresi arasındaki iletişimi belirleyen gereksinimler işlevsel gereksinim olarak tanımlanır.
- İşlevsel gereksinimler ayrıca sistemin herhangi bir durum karşısındaki davranışını belirler..
- Ör: Maaş çeki hazırlama da, işlevsel gereksinimler;
 - Maaş çekinin ne zaman hazırlanacağı,
 - Çek hazırlamak için ne tür girdiler verilmesi gerektiği,
 - Hangi koşullarda bir kişiye maaş verilmeyeceği gibi gereksinimler işlevsel gereksinimlerdir.

İşlevsel
Olmayan
Gereksinimler



- Kullanıcının sorunundan bağımsız olarak çözülmesi gereken sorunlar işlevsel olmayan gereksinim olarak tanımlanır.
- Bir çok kaynakta bu terim yerine "Sistem Kısıtları" terimi de kullanılmaktadır.
- Ör: İşlevsel olmayan gereksinimler
 - Kullanılacak bilgisayar türü,
 - yazılım geliştirme ortamı,
 - kullanılacak veri tabanı yönetim sistemi vb.bu tür gereksinimlere örnek verilebilir.

Gereksinim Türleri



Fiziksel Çevre

- İşlevlerin geliştirileceği, işletileceği aygıtlar nerededir.
- Sistem tek bir yerde mi olacak? birden çok ve fiziksel olarak birbirinden ayrılmış yerler söz konusu mu?
- Sıcaklık nem oranı veya manyetik etkileşim gibi çevresel kısıtlamalar var mı?



Ara yüzler

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



Kullanıcı ve İnsan etmeni

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

İşlevsellik

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



Belgeleme

- Ne kadar belgeleme gereklidir?
- Belgeleme hangi kullanıcı kitlesini hedeflemektedir?



Veri

- Hem giriş hem çıkış için verinin biçimi ne olmalıdır?
- Bu veri ne sıklıkla alınacak veya gönderilecektir?
- Bu verinin doğruluk (kesinlik) ölçüsü ne olmalıdır?
- Hesaplamalar hangi duyarlık derecesine kadar yapılandırılacaktır?
- Sistemde ne kadar veri akışı olacaktır?
- Veri belirli bir zaman süresince kaynağında saklanacak mı?



Kaynaklar

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

Güvenlik

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



Kalite Güvencesi

- Güvenirlilik için gereksinimler nelerdir?
- Sistemin özellikleri insanlara nasıl aktarılmalıdır?
- Sistem çökmeleri arasında öngörülen zaman aralığı nedir?
- Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri nelerdir?
- Hataları kendisi bulup gidermeli mi?
- Çökmeler?
- Tasarımda yapılacak değişiklikler
- Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri?
- Sistemi bir bilgisayardan diğerine aktarmalı mı?

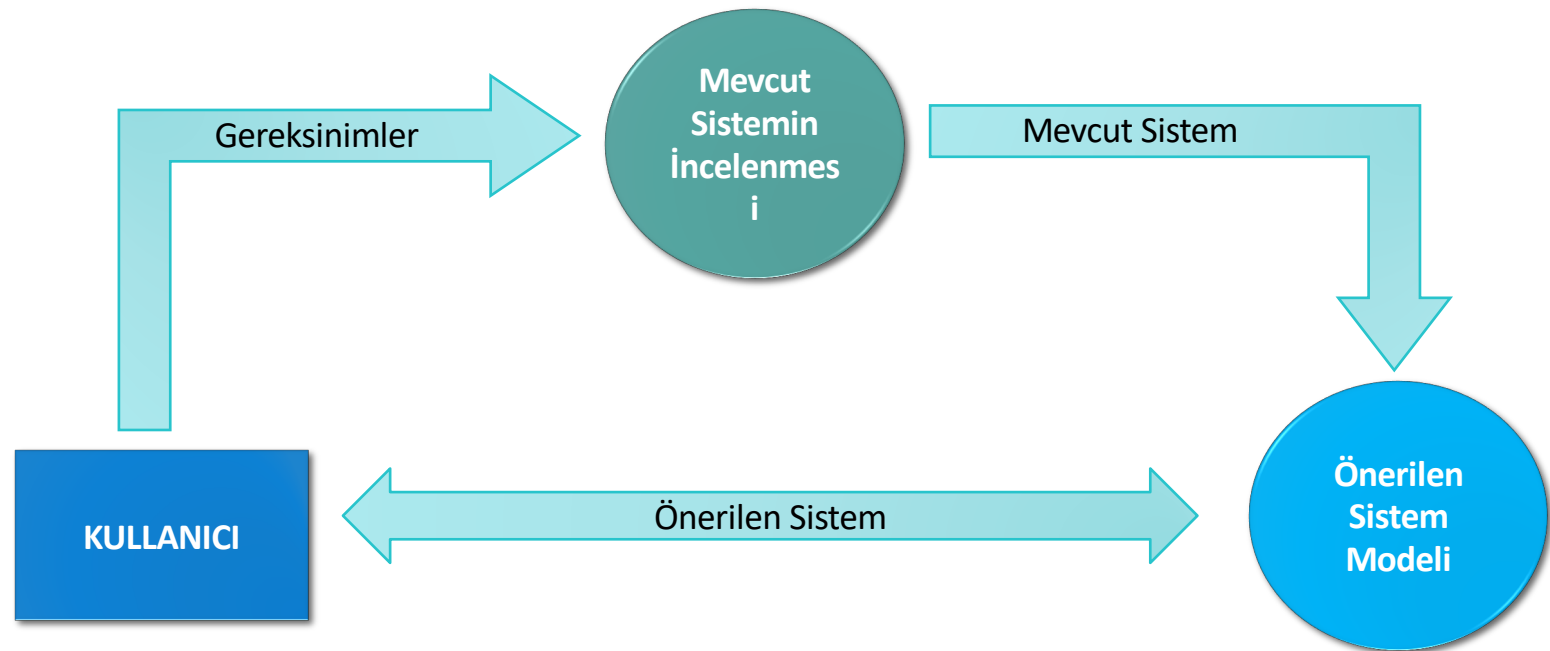
Gereksinim Özellikleri

Gereksinimler üç amaca hizmet eder. Bunlar;

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



Gereksinim Çözümleme Çalışması



Gereksinim Çözümleme Çalışması

Mevcut sistemin incelenmesi

- Amaç: Yazılım geliştirilecek sistemin tanınmasıdır.
- Girdi, İşlev ve çıktı analizi yapılır.
- Kanun, yönerge ve yönetmenlikler incelenir.
- Elde yürütülen işlerde kullanılan form, defter ve yazışma örnekleri incelenir.

Gereksinim Çözümleme Çalışması

Önerilen Sistemin Modellenmesi

- Önerilen sistemin işlevsel yapısını, veri yapısını ve kullanıcı ara yüzünü oluşturur.
- Bu model daha çok bilgi sistemini geliştirecek teknik personele yöneliktir.
- **Mantıksal model** olarak ta tanımlanır.



Doğrulama Süreci

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



Örnek

- Görev planlaması için kesinlik (doğruluk) yeterli olacaktır.
- Pozisyon hatası, yörünge boyunca 50 metreden, yörünge dışında 30 metreden az olacaktır.
- Sistem sorgulamaları gerçek zamanlı olarak yanıtlanmalıdır.
- Sistem kişi sorgulamaları en çok iki saniye içinde verilmelidir.

Gereksinim Verisi Toplanması

1. Sorma Yöntemi
 - Karşılıklı görüşme
 - Anket
2. Psikolojik Türetme Teknikleri
3. İstatiksel Teknikler



Sorma Yöntemi

- Sorma yöntemi, gereksinim verilerinin toplanması sırasında kullanılan en önemli yöntemlerden biridir.
- Karşılıklı görüşme ya da anket yolu ile uygulanır.



Sorma Yöntemi

Karşılıklı Görüşme

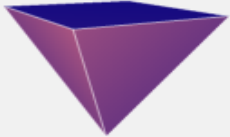
- Karşılıklı görüşme sırasında, gereksinimleri ilişkin amaçlar, düşünceler, remi olmayan yöntemler, duygular ve düşünceler araştırılır.
- Karşılıklı görüşme, veri toplama için en etkin yollardan biridir.
- Sorma yönteminde açık uçlu sorular (yanıtları kişiden kişiye değişebilen, yoruma açık sorular) ya da kapalı uçlu sorular (yanıtları evet hayır ya da bir sayı biçiminde kesin olan sorular) sorulabilir.
- Çözümleme kolaylığı, verinin güvenliği ve duyarlılığı açısından kapalı uçlu sorular tercih edilmelidir.
- Sorular, dağınık olarak değil, yapısal bir biçimde sorulmalıdır. Bu neden bu üç tarzdan biri seçilmelidir.
 - Piramit Tarzı
 - Koni Tarzı
 - Elmas Tarzı

Karşılıklı Görüşme Yapısal Soru Türleri



Piramit Tarzı

Piramit Tarzı: Özel sorularla başlayıp, giderek genel sorularla sürdürme.



Koni Tarzı

Koni Tarzı: Genel sorularla başlayıp özel sorularla sürdürme.



Elmas Tarzı

Elmas Tarzı: Özel sorularla başlayıp, genel sorularla sürdürme, tekrar özel sorularla sona erdirmeye.

Sorular sırasında dikkat edilecek önemli konu; **yönlendirici** sorular ve **iki nesnel** sorulardan kaçınmak olmalıdır.

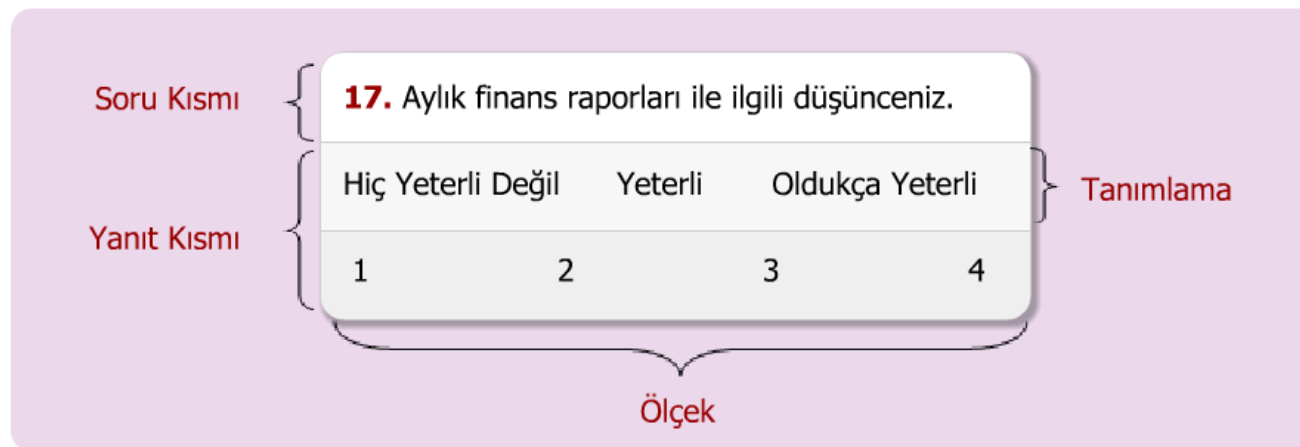
Sorma Yöntemi

Anket Yöntemi

Kullanıcı sayısının fazla olduğu durumlarda eğilimleri ve davranış biçimlerini saptamak için kullanılır.

Genelde yazılı test biçiminde hazırlanır.

Bir anket sorusu temel olarak soru kısmı ve yanıt kısmı olmak üzere iki kısımdan oluşur. Yanıt kısmı da tanımlama ve ölçek kısmı olmak üzere iki bölümden oluşur.



Psikolojik Türetme Teknikleri

- Özellikle belirsizliğin fazla olduğu ve zayıf yapıli ortamlarda, bilgi edinebilmek amacıyla insan psikolojisine dayalı teknikler kullanılır.
- Bu teknikler temelde görüşme ve ankete dayalı tekniklerdir.
- Diğerlerinden farkı, bilgi üretmek için psikolojide bilinen "üçleme" tekniğini kullanmasıdır.
- Bazı diğer psikolojik türetme yöntemleri, karar verme ortamlarında bilgi gereksinimlerini saptamak amacıyla algılama haritaları ve neden-etki çizgeleri kullanmaktadır.

İstatistiksel Teknikler

- Veri yoğun ve veri hacmi yüksek ortamlarda verinin özelliklerini belirlemek amacıyla kullanılır.
 - Bu yöntemlerden en çok bilinen ikisi **Örnekleme Yöntemi** ve **PIRA Modelidir**.
- **Örnekleme yöntemi**, bir topluluk içerisinde, sistematik yolla temsil edici örnek alma biçiminde tanımlanır.
- Amaç, veri toplama hızını arttırmak ve verilerdeki çelişkileri önlemektir.



Veri Modelleme Yöntemleri

Önerilen sistemin mantıksal modelinde veri yapısını açıklamak amacıyla 'Veri Modelleme' yöntemleri kullanılmaktadır.

Bu yöntemler, veri yapısını çeşitli düzeylerde tanımlama (en soyut düzeyden en ayrıntı düzeye kadar) amacını güder.

Sistem Çözümleme aşamasında en yaygın olarak kullanılan veri modelleme yöntemleri:

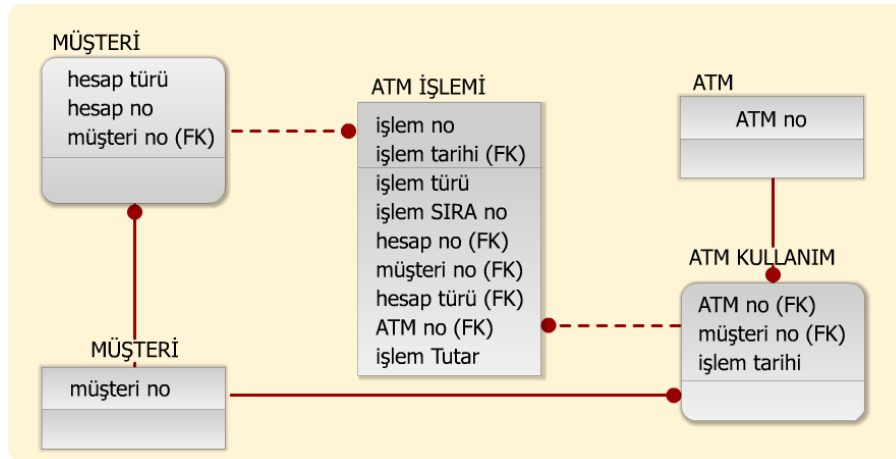
- Nesne İlişki Şemaları
- Veri Sözlüğü

olarak bilinmektedir.

Nesne-İlişki Şemaları, veri yapısını en soyut düzeyde tanımlamak amacıyla kullanılır.

Veri Sözlüğü ise veri yapısına ilişkin ayrıntı bilgileri içerir. Günümüzde Bilgisayar Destekli Yazılım Geliştirme Araçlarında bu yöntemler bütünleşik olarak kullanılmaktadır.

Nesne-İlişki Şemaları

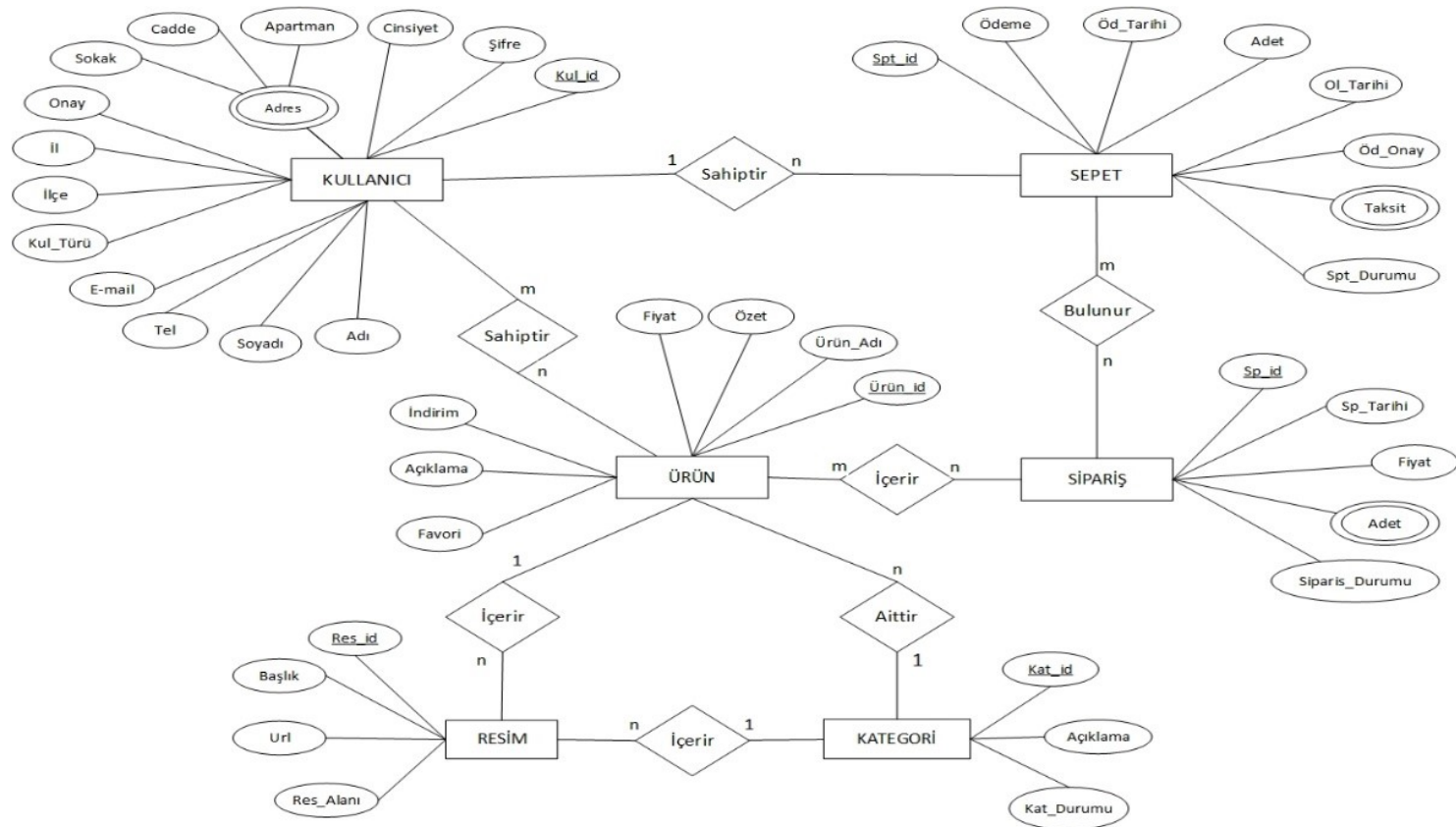


- Nesne ilişki şemaları, veri tasarımı açısından çok önemlidir.
- Geliştirilecek sistemin kullanacağı ana veri nesneleri ve aralarındaki ilişkileri belirtir.
- Günümüzde birçok CASE aracı nesne ilişki şemalarını otomatik olarak veri tabanı tablo yapılarına dönüştürmektedir.

Bir veri nesnesi, üç temel özelliği ile bilinir:

1. Veri nesnesi varlığının adı: Veri nesnesi varlığını tanımlayan özelliktir.
2. Veri nesnesi varlığının özellikleri
3. Veri nesnesi varlığının diğer veri nesnesi varlıklarına referansı: Veri nesnesi varlığının diğer veri nesneleri ile olan ilişkisinin belirtilmesi amacıyla kullanılır. Bu amaçla her bir veri nesnesini tek olarak belirleyen bir belirteç (anahtar) kullanılır. Söz konusu belirteç veri nesnesi varlığının ad özellikleri arasında yer alır.

Veri Modelleme ER diyagramı



Veri Sözlüğü

➤ Nesne ilişki şemalarında belirtilen nesne özelliklerinin ayrıntılı tanımları Veri Sözlüğünde yer alır. Söz konusu ayrıntılı tanımlar genel olarak:

- Veri Adı
- Veri Eş-adı (Aynı veri için kullanılan diğer ad)
- Nerede/nasıl kullanıldığı
- İçerik tanımı

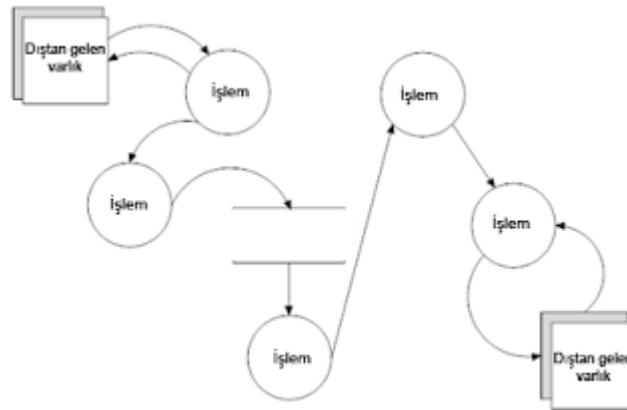
türünde bilgileri içerir. Veri nesnelerinin sırasının tanımlanması, birden çok veri nesnesi arasından seçim ve veri nesnelerinin yinelemeli gruplamaları için özel gösterim biçimleri kullanılarak karmaşık veri nesneleri tanımlanır.

Nitelik	Tanım	Etki alanı	Gerekli	Tür
<i>kat_id</i>	Bu alan adı Primary key ve otomatik olarak artış gösterir.	Primary key ve değiştirilemez. Integer olarak tanımlıdır.	✓	Int(11)
<i>kat_adi</i>	Kategori adı	Kategori seçeneklerinden biri, adı.	✓	Varchar(150)
<i>aciklama</i>	Sipariş tarihi	Boş geçilebilir.		text
<i>kat_durumu</i>	Kategori aktif/pasif	Kategori durumu	✓	Enum('1','0')

Kategoriler Tablosu Veri Sözlüğü

Süreç/İşlem Modelleme Yöntemleri

- Geliştirilecek sistemin süreç ya da işlemlerini ve bu süreçler arasındaki ilişkileri tanımlamak amacıyla kullanılan yöntemlerdir.
 - Veri Akış Diyagramları (DFD)
 - Süreç Tanımlama Dili (PDL)

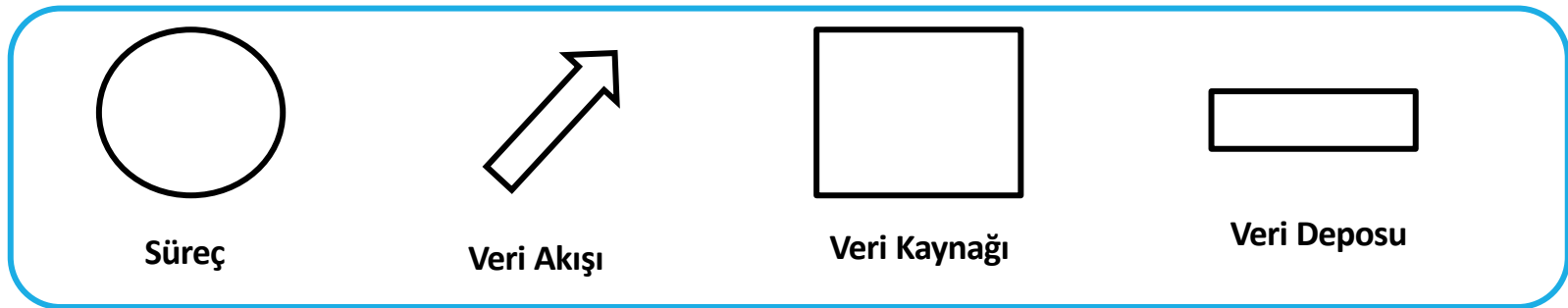


Veri Akış Diyagramı

- VAD kullanılarak, geliştirilecek sistemin mantıksal modeli, 'Yukarıdan Aşağıya' bir yaklaşımla oluşturulur.
- Sistem önce en genel biçimiyle ele alınır, yalnızca dışsal ilişkileri incelenir.
- Daha sonra, sistemin iç yapısındaki süreçler ve bu süreçler arasındaki ilişkiler belirlenen bir ayrıntı düzeyine kadar modellenir.
- Yapısal sistem geliştirme metodolojilerinde kullanılan tek veri modelleme yöntemi, **Veri Akış Diyagramları (VAD)** yöntemidir.
- VAD günümüzde oldukça yaygın olarak kullanılmaktadır. Hemen hemen bütün CASE araç ve ortamları VAD yöntemini içermektedir.
- VAD'nin oldukça yaygın olarak kullanılmasındaki temel etmen, hem bilgisayara yabancı olan kullanıcılar, hem de yazılım geliştiriciler için oldukça kolay algılanması ve kullanılabilmesidir.

VAD Yönteminde Kullanılan Semboller

VAD, 4 temel sembol kullanılarak oluşturulur.



- **Süreç** ya da işlemleri belirtmek amacıyla çember şekli kullanılmaktadır. Verilerin alındığı, ya da gönderildiği **dış birimleri** göstermek amacıyla **dikdörtgenler** kullanılmaktadır.
- Mantıksal veri yığınlarını göstermek amacıyla **ucu açık dikdörtgenler** kullanılmaktadır.
- Gerek süreçler arası gerekse süreçler ile veri kaynakları ve veri depoları arasındaki veri akış ilişkileri göstermek amacıyla **oklar** kullanılmaktadır.

Veri Akış Diyagramı

Kapsam Diyagramı

1

Geliştirilecek bilgi sisteminin dışsal ilişkilerinin göstermek amacıyla kullanılır. Yalnızca bir çember ve gerekli sayıda veri kaynağı ile aralarındaki veri akışlarının belirtilmelerinden oluşur.

Genel Bakış Diyagramı

1

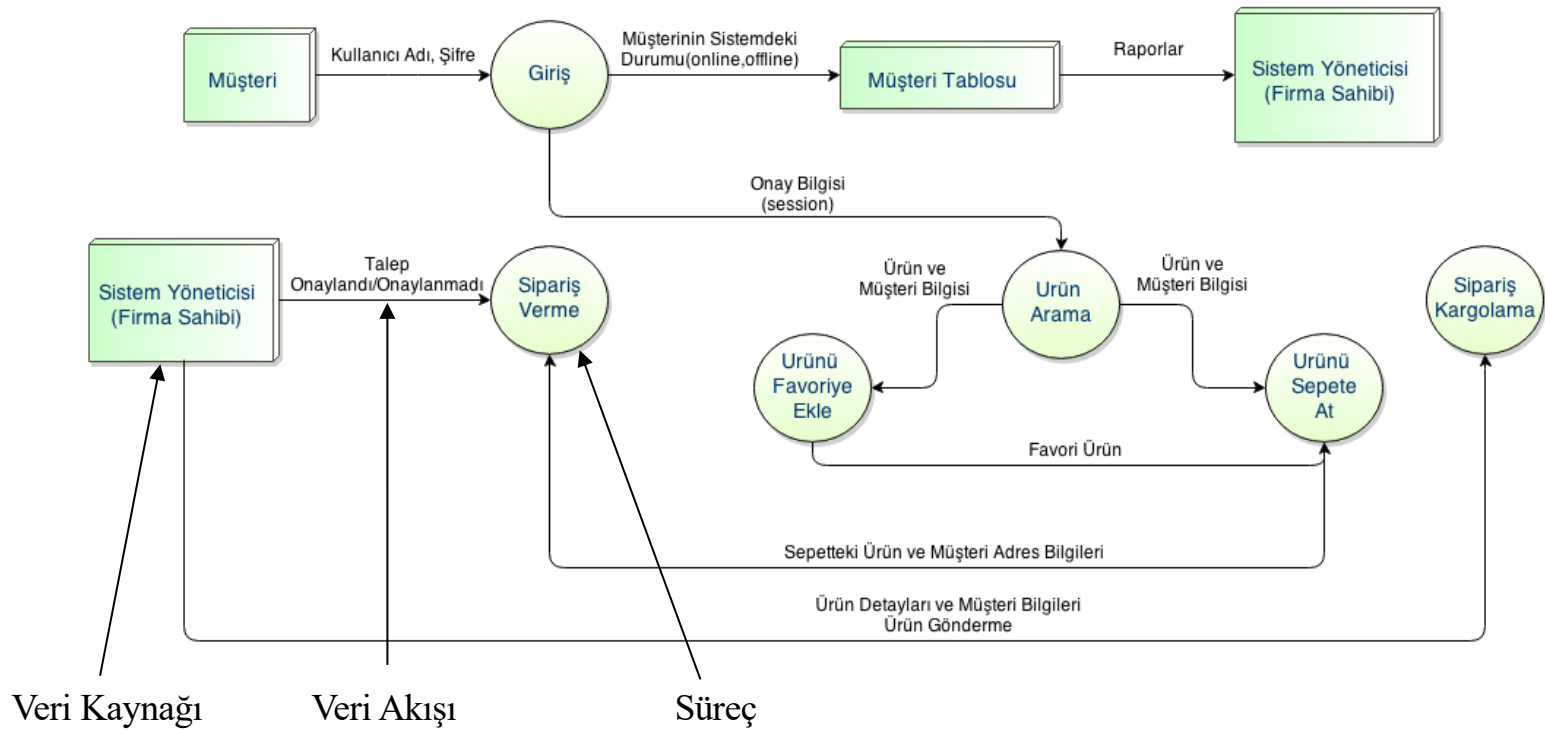
Bir Bilgi sisteminin ana işlevlerini, bu işlevlere ilişkin veri kaynaklarını, veri depolarını ve işlemler arasındaki ilişkileri içeren VAD, Genel Bakış Diyagramı olarak tanımlanmaktadır. Genel Bakış Diyagramı, kapsam diyagramının ayrıştırılmış yada "patlatılmış" biçimidir. Bir bilgi sistemi için, bilgi sisteminin büyüklüğüne göre en çok birkaç Genel Bakış Diyagramı çizilir.

Ayrıntı Diyagramı

1

Geliştirilecek bilgi sisteminin dışsal ilişkilerinin göstermek amacıyla kullanılır. Yalnızca bir çember ve gerekli sayıda veri kaynağı ile aralarındaki veri akışlarının belirtilmelerinden oluşur.

Veri Akış Diyagramı



Veri Akış Diyagramı Neyi Gösterir

- Bilgi sisteminin durağan yapısını,
- Bilgi sisteminin süreçlerini ve bu süreçler arasındaki veri akış ilişkisini,
- Bilgi sistemi ile ilişkili olan kurum birimlerini ya da dış birimleri kaynak olarak,
- Bilgi sistemi için gerekli olan ana veri depolarının neler olduğunu ve hangi süreçler tarafından kullanıldığını,
- Bilgi sisteminin süreçlerini yukarıdan-aşağıya ayrıştırma ile gösterir.



Veri Akış Diyagramı Neyi Göstermez

- Bilgi sistemi süreçlerinin zamana ilişkin durumunu ve bu duruma ilişkin bilgileri göstermez.
- Bilgi sistemi süreçlerinin kendi aralarındaki karar ilişkisini göstermez.
- Gerek bilgi sistemi süreçleri, gerekse akışları ve veri kaynakları ve depoları için ayrıntı içermez.



Süreç Tanımlama Dili

- Bilgi sistemi süreçlerinin iç yapılarını belirtmek amacıyla; kullanılan araç, yöntem ya da gösterim biçimleridir.
- Üç farklı yaklaşım izlenir:
 - Düz Metin
 - Şablon
 - Yapısal İngilizce

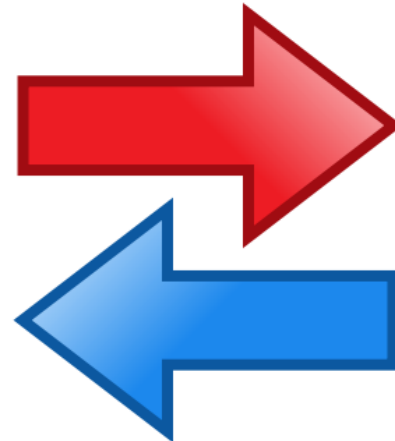


Düz Metin

- Üçgeni inceler, üçgenin kenar boyutlarını A,B,C) girdi olarak alır.
- Süreç önce bütün bu değerlerin pozitif olup olmadığını denetler.
- Eğer değerlerden biri negatif ise hata verir.
- Süreç tüm kenar uzunluklarının bir üçgeni belirleyecek şekilde geçerli olup olmadığını denetler.
- Eğer geçerli ise eşkenar, ikizkenar veya çeşitkenar olduğunu belirler.

Şablon

- Süreç : Üçgeni İncele
- Girdi : Üçgenin kenar boyutları
- Çıktı : Üçgen türü, hata iletisi
- İşlem : A,B,C değerlerinin pozitif olup/olmadığını denetle. Negatif ise hata iletisi ver. A,B,C değerlerinin geçerli olup olmadıklarını denetle. Eğer geçerli değerler ise üçgenin türünü belirle (eşkenar, ikizkenar veya çeşitkenar). Değilse hata iletisi ver



Yapısal İngilizce

■ Procedure : Üçgeni İncele

Üçgenin kenar boyutlarını oku

If herhangi bir boyut negatif then HATA

If en büyük kenar diğer iki kenar toplamından küçük then

begin

eşit kenar sayısını belirle

If 3 kenar eşit then eşkenar

If 2 kenar eşit then ikiz kenar

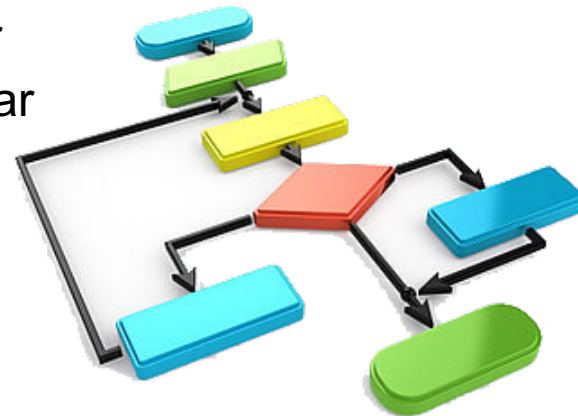
If 1 kenar eşit çeşitkenar

Üçgen türünü yaz.

end

else degerler üçgen belirtmiyor

Endproc.



Kullanıcı Arayüz Prototipleme (KAP)

- Ekran tasarımı için kullanıcıdan onay alınması esastır.
- Geleneksel yaklaşımlarda bilgi sistemi girdi ve çıktılarının tanımları el ile kağıt üzerinde yapılır ve kullanıcılardan bu biçimiyle onay alınmaya çalışılır.
- Gereksinimlerin kesinleştirilmesini kolaylaştırır.



KAP Özellikleri

- Ayrılan zaman sistem analizi için ayrılan zamanın %5'ini aşmamalıdır.
- Her özellik bir kez gösterilmelidir.
- Hiç bir içsel işlem içermemelidir.



KAP Raporları

- Raporların bir kod numarası olmalıdır.
- Her rapor için örnek çıktı yapısı ayarlanır. Word dokümanında örnek yapı hazırlanır. İlgili çıktı gönderilirken bu çıktı gönderilir.

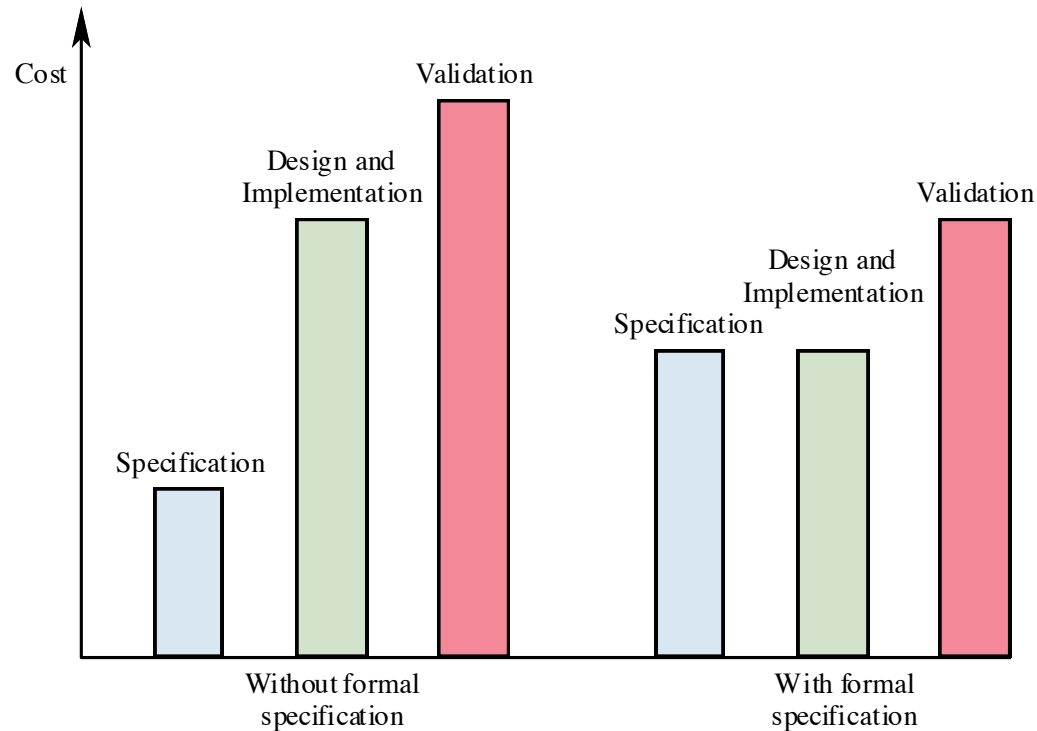


Sistem Analiz Raporu

- Sistem analiz çalışması sonucunda alınan rapordur (şartname). Söz Konusu rapor çalışmanın tüm ayrıntılarını içerir.
- 5 ana bölümde incelenebilir.
 - Giriş
 - Mevcut sistemin incelenmesi
 - İstenen sistem mantıksal modeli
 - Arayüz gerekleri
 - Belgeleme gerekleri



Geliştirim Masrafları Karşılaştırması



Özet

- **Planlama**, yazılım geliştirme sürecinin ilk aşamasıdır.
- **Proje Kaynakları**: İnsan, donanım ve yazılım olarak 3 kategoride incelenir.
- **Maliyet kestirimi**; bir bilgi sistemi ya da yazılım için gerekebilecek iş gücü ve zaman maliyetlerinin üretimden önce belirlenebilmesi için yapılan işlemlerdir.
- Problemin bilgi ortamının incelenmesinde, **kullanıcı girdileri, kullanıcı çıktıları, kullanıcı sorguları, dosyalar** ve **dışsal ara yüzler** mevcuttur.
- **Proje Sınıfları**: Ayırık, Yarı Gömülü ve Gömülü Projeler olmak üzere 3 şekildedir.
- Maliyet Çarpanı hesaplanması 15 maliyet etmeninin çarpımı sonucunda bulunur.
- COCOMO Modeli kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir bunlar: Temel, Ara ve Ayrıntı Modelidir.
- Gereksinim, sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.

Özet

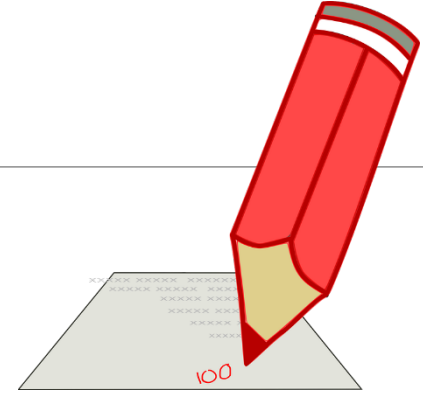
- Gereksinim, kullanıcı ve tasarımcı ya da yazılım mühendisi ile ilgili olarak iki amaca yönelik olacak biçimde tanımlanmalıdır;
- Gereksinim çeşitleri ikiye ayrılır; **İşlevsel** ve **İşlevsel olmayan** gereksinimler.
- Genel olarak 9 tane gereksinim türü bulunmaktadır.
- Gereksinim çözümleme çalışmasında, **mevcut sistem** ve **önerilen sistem** önemli bir gereksinim incelemesidir.
- Gereksinim Verisi Toplanması yöntemi 3 şekilde yapılır. Bunlar; **Sorma**, **psikolojik türetme teknikleri** ve **istatistiksel yöntem**dir.
- Veri Modelleme Yöntemleri: **Nesne-ilişki şemaları** ve **veri sözlüğü** olarak iki tiptedir.
- Süreç/İşlem Modelleme Yöntemleri: Veri Akış Diyagramları ve Süreç Tanımlama Dili
- Veri Akış Diyagramı, yapısal sistem geliştirme metodolojilerinde kullanılan tek veri modelleme yöntemidir.
- Veri Akış Diyagramı: **Kapsam**, **Genel Bakış** ve **Ayrıntı diyagramı** olarak 3 çeşittir.

Sorular

1. Kaç çeşit proje kaynağı mevcuttur?
2. Maliyet kestirimi nedir ve ne için yapılmaktadır?
3. Proje sınıfları nelerdir?
4. COCOMO nedir?
5. Yazılımda Sistem .çözümlemesi aşamasını açıklayınız.
6. Gereksinim nedir ve ne amaçla yapılmaktadır?
7. Kaç çeşit gereksinim türü vardır ? Bunları açıklayınız.
8. Gereksinim çözümleme aşamasında yapılan incelemeleri söyleyiniz.
9. Gereksinim verisi toplarken hangi yöntemlere başvurulur?
10. Veri modelleme yöntemlerinden Nesne-İlişki şemasını açıklayınız.
11. Veri modelleme yöntemlerinden Veri sözlüğü ne anlama gelmektedir. Örnek veriniz.
12. Süreç modelleme yöntemleri nelerdir?
13. VAD ne anlama gelmektedir?
14. Kaç çeşit VAD diyagramı mevcuttur? Açıklayınız.

Ödev

Günümüzde kullanılan maliyet kestirim yöntemlerini araştırınız.



Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Sorularınız





YMH311-Yazılım Tasarım Ve Mimarisi

Yazılım Tasarımı

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Yazılım Tasarımının Önemi.....	4
Tasarım Kavramları.....	8
Yapısal Tasarım.....	16
Tasarlanması Gereken Ortak Alt Sistemler.....	29
Kullanıcı Arayüz Tasarımı.....	36
Tasarım Kalite Ölçütleri.....	43
Yapışıklık.....	50

Amaçlar

- Tasarımın ne olduğunu ve çeşitli tasarım türlerinin ürünün farklı yönleriyle nasıl ilgilendiğini açıklamak
- Tasarımı bir problem çözme etkinliği olarak sunmak, soyutlama ve modellemenin tasarımdaki rolünü ortaya koymak
- Yazılım yaşam döngüsünde tasarımın yerini belirlemek
- Yazılım mühendisliğinde tasarım metotlarını incelemek



Yazılım Tasarımının Önemi

- Tasarlanmış (designed) bir dünyada yaşıyoruz.
- Tasarım ekonomik olarak öneme sahiptir ve yaşam kalitemizi doğrudan etkiler.
- Yazılım son derece yaygın hale gelmektedir.
- Yazılım tasarımının kalitesinin önemli sonuçları olmaktadır ve yazılım tasarımcıları bunların farkında olmalı, bunları ciddiye almalıdır.

Giriş

- Tasarım, Sistem Analizi çalışması sonucunda üretilen **Mantıksal Modelin Fiziksel Modele dönüştürülme çalışmasıdır.**
- Fiziksel Model geliştirilecek yazılımın;
 - hangi parçalardan oluşacağını,
 - bu parçalar arasındaki ilişkilerin neler olacağını,
 - parçaların iç yapısının ayrıntılarını,
 - gerekecek veri yapısının fiziksel biçimini (dosya, veri tabanı, hash tablosu, vektör, vs.)tasarımını içerir.

Yazılım Ürünleri

Bir yazılım ürünü, müşterinin gereksinim ve isteklerini karşılayan bir veya daha fazla programdan, verilerden, ve destekleyici materyal ve hizmetlerden oluşan bir varlıktır. Bu ürün, tek başına bir ürün olabileceği gibi başka bir ürünün temel bileşeni de olabilir.

Yazılım Tasarımı Nedir?

- Yazılım tasarımcıları da temelde diğer disiplinlerdeki tasarımcıların yaptığı işi yapar.
- Tasarlanan şey bir yazılım ürünüdür.

Yazılım tasarımı, müşterinin gereksinim ve isteklerini karşılayan yazılım ürününün doğasını ve bileşimini belirleme etkinliğidir.

Tasarım Kavramları

Soyutlama (abstraction):

- **Soyutlama (abstraction):** Detayları gizleyerek yukarıdan bakabilme imkanı sağlar.



Soyutlama kavramı veri, işlev ve yapısal açılar için geçerlidir. Örneğin bir kapı nesne olarak ele alındığında onun kulpu, rengi menteşeleri, malzemesi gibi detayları düşünmeden kapıyı bir ev mimarisi içinde değerlendirebiliriz. Aksi taktirde diğer detaylara yoğunlaşan bir tasarımcı 'oda' düzeyinde görsel canlandırmalara hakim olamaz.

Tasarım Kavramları

İyileştirme (enhancement):

- **İyileştirme (enhancement):** Soyutlama düzeyinde irdeleme bittikten sonra, daha alt seviyelere inilerek tanımlamalarda ayrıntı, bazen de düzeltme yapılarak **tasarımın daha kesinlik kazanması sağlanır.**

Kulp	Menteşe
Renk	Malzeme

Soyutlama kavramı veri, işlev ve yapısal açılar için geçerlidir. Örneğin bir kapı nesne olarak ele alındığında onun kulpu, rengi menteşeleri, malzemesi gibi detayları düşünmeden kapıyı bir ev mimarisi içinde değerlendirebiliriz. Aksi taktirde diğer detaylara yoğunlaşan bir tasarımcı 'oda' düzeyinde görsel canlandırmalara hakim olamaz.

Tasarım Kavramları

Modülerlik (modularity):

- **Modülerlik (modularity):** Sistemi istenen kalite faktörleri ışığında parçalara ayırıştırma sonucu elde edilir. Bir işlev için sistemin tümü değil, ayrılmış bir kısmı üzerinde çalışma yapabilme olanağı sağlar.

Yuvarlak Köşeli Uzun Kısa ...	Yay Dil Vidalar ...
Ahşap Metal Cam ...	Beyaz Metalik Kahverengi ...

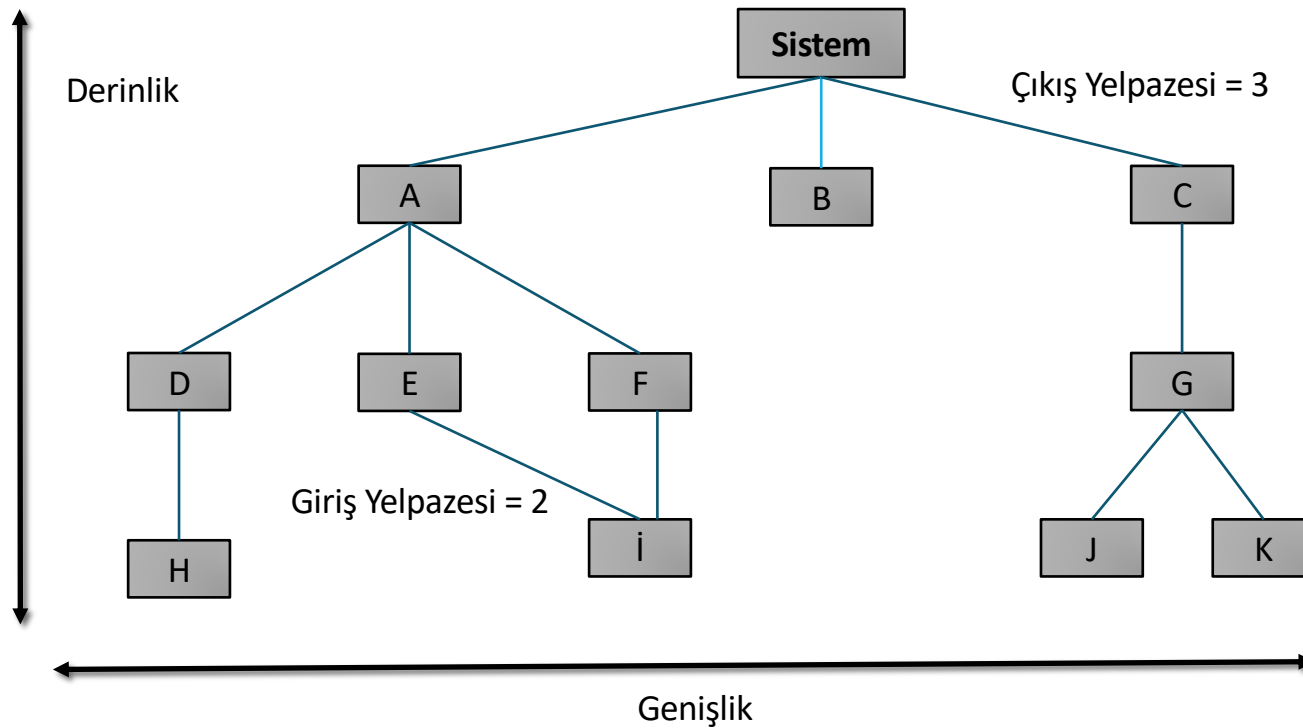
Kapı ve pencerenin de kendi ayrıntılarını, birer kelime ile soyutladığımız isimleri içerisinde saklamaları, onları birer neşene olarak bir oda içerisinde 'modüler' bir yapı düzeninde olabilmelerini sağlar. Bir pencerenin yerini değiştirmek, tasarım esnasında onun camı, menteşesi ve malzemesi gibi detayından bağımsız, aynı zamanda da odadaki diğer 'modüllerden' hemen hemen bağımsız olarak ele alınabilir.

Modülerlik

- Bütün karmaşıklığın tek bir modülde toplanması yerine, anlaşılabilir ve dolayısıyla projenin zihinsel kontrol altında tutulması için sistem bir çok modüle ayrılır.
- Modüller, isimleri olan tanımlanmış işlevleri bulunan ve hedef sistemi gerçekleştirmek üzere tümleştirilen birimlerdir.



Sistem ve Modülleri



İşlevsel Bağımsızlık

- Modüllere parametre ile veri gönderilir ve sonuç değer alınır. Bu modülü çağıran program parçası sadece bu sonucu kullanabilir. Çağrılan modülün işlevsel olarak yaptıkları ile ilgili değildir.



Veri Tasarımı

- Yapı Tasarımı, arayüz tasarımı ve süreç tasarımından önce yapılması gereken ilk tasarım veri tasarımıdır.
- Bilgi saklama ve soyutlama bu işlem için önemli kavramlardır.

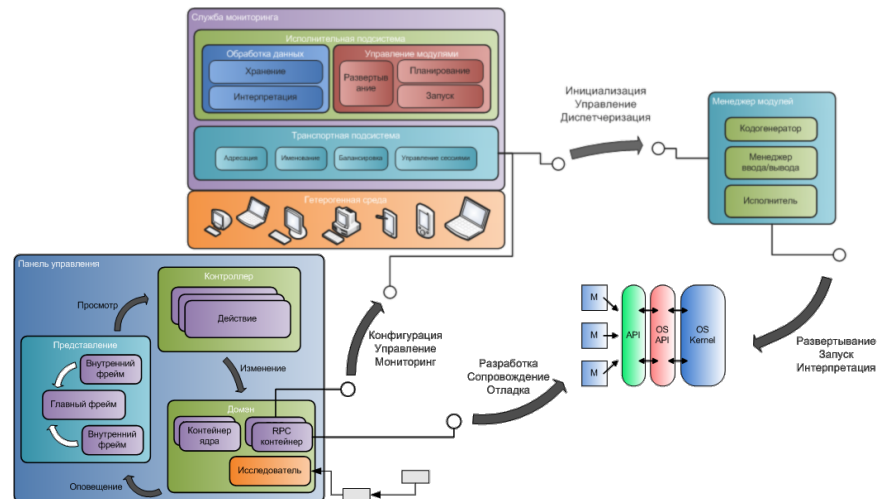


Veri Tasarımında Dikkat Edilecek Konular

- Değişik veri yapıları değerlendirilmelidir.
- Bütün veri yapıları ve bunlar üzerinde yapılacak işlemler tanımlanmalıdır.
- Alt düzeyde tasarım kararları tasarım süreci içerisinde geciktirilmelidir.
- Bazı çok kullanılan veri yapıları için bir kütüphane oluşturulmalıdır.
- Kullanılacak programlama dili soyut veri tiplerini desteklemelidir.

Yapısal Tasarım

- Yapısal Tasarımın ana hedefi modüler bir yapı geliştirip modüller arasındaki kontrol ilişkilerini temsil etmektir.
- Ayrıca yapısal tasarım bazen de veri akışlarını gösteren biçime dönüştürülebilir.
- Veri Akışları Üç parçada incelenebilir
 - Girdi Akışı
 - Çıktı Akışı
 - İşlem Akışı



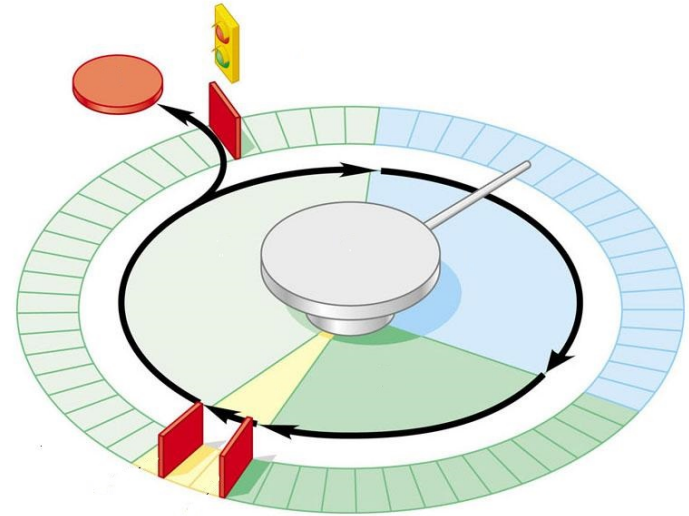
Ayrıntı Tasarım- Süreç Tasarımı

- Süreç tasarımı; veri, yapı ve ara yüz tasarımından sonra yapılır.
- İdeal şartlarda bütün algoritmik detayın belirtilmesi amaçlanır.
- Ayrıca süreç belirtiminin tek anlamı olması gerekir, değişik şahıslar tarafından farklı yorumlanmamalıdır.
- Doğal diller kullanılabilir (açıklamalarda, çünkü doğal dil tek anlamlı değildir)
- Süreç Tanımlama Dili (PDL)

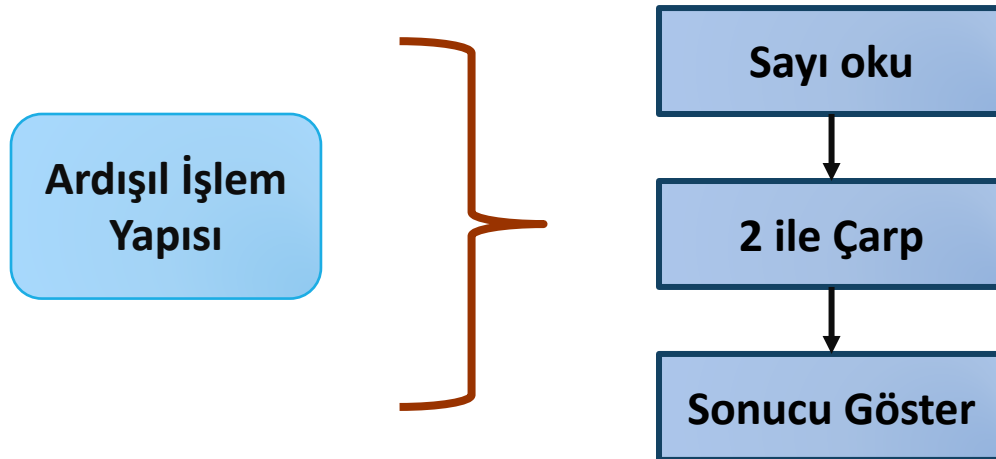


Yapısal Program Yapıları

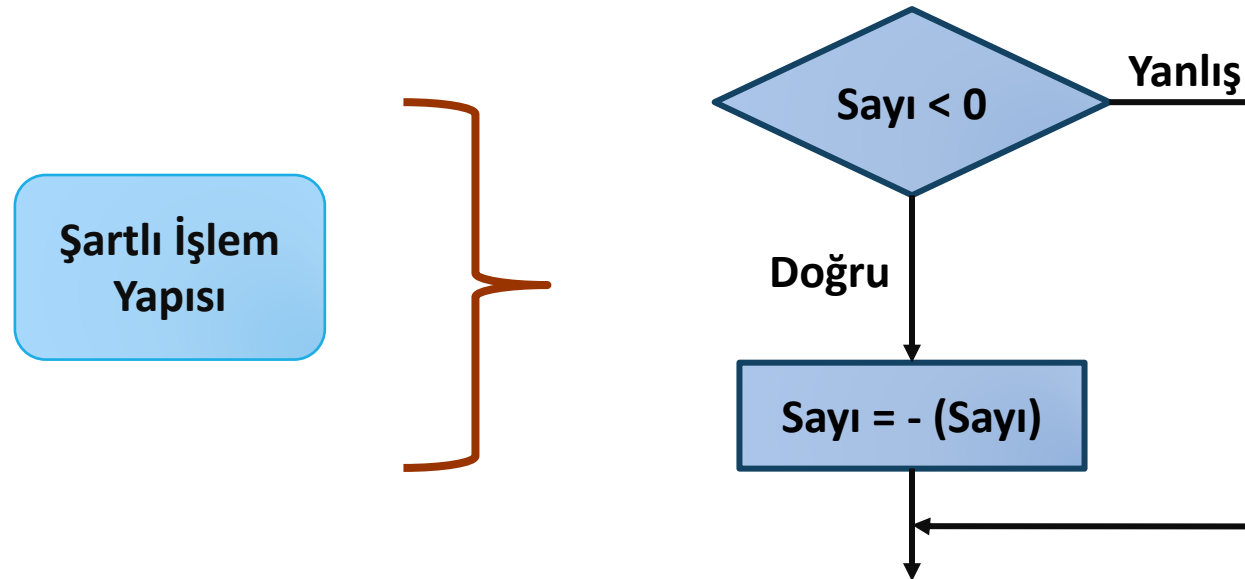
- Yapısal programlamanın temel amacı;
 - program karmaşıklığını en aza indirmek,
 - program anlaşılabilirliğini artırmaktır.
- Bu amaçla şu yapıları kullanılır;
 - Ardışıl İşlem yapısı
 - Koşullu işlem yapısı
 - Döngü yapısı
- GOTO kullanımı uygun değildir.



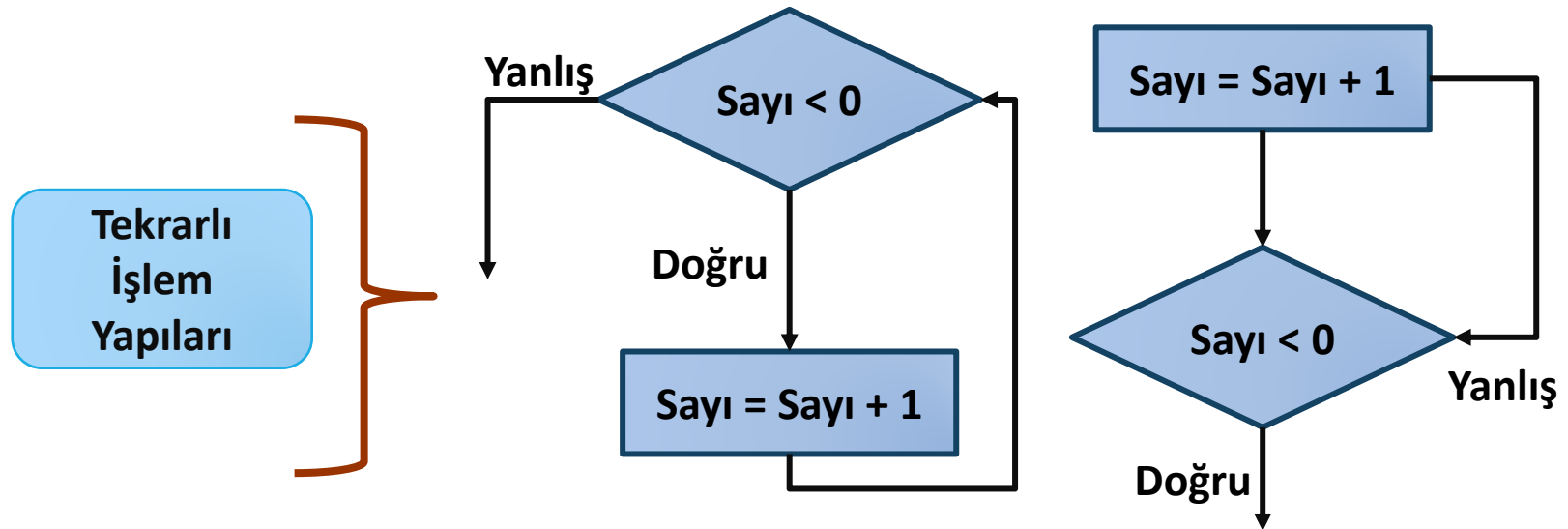
Program Akış Diyagramı Yapıları



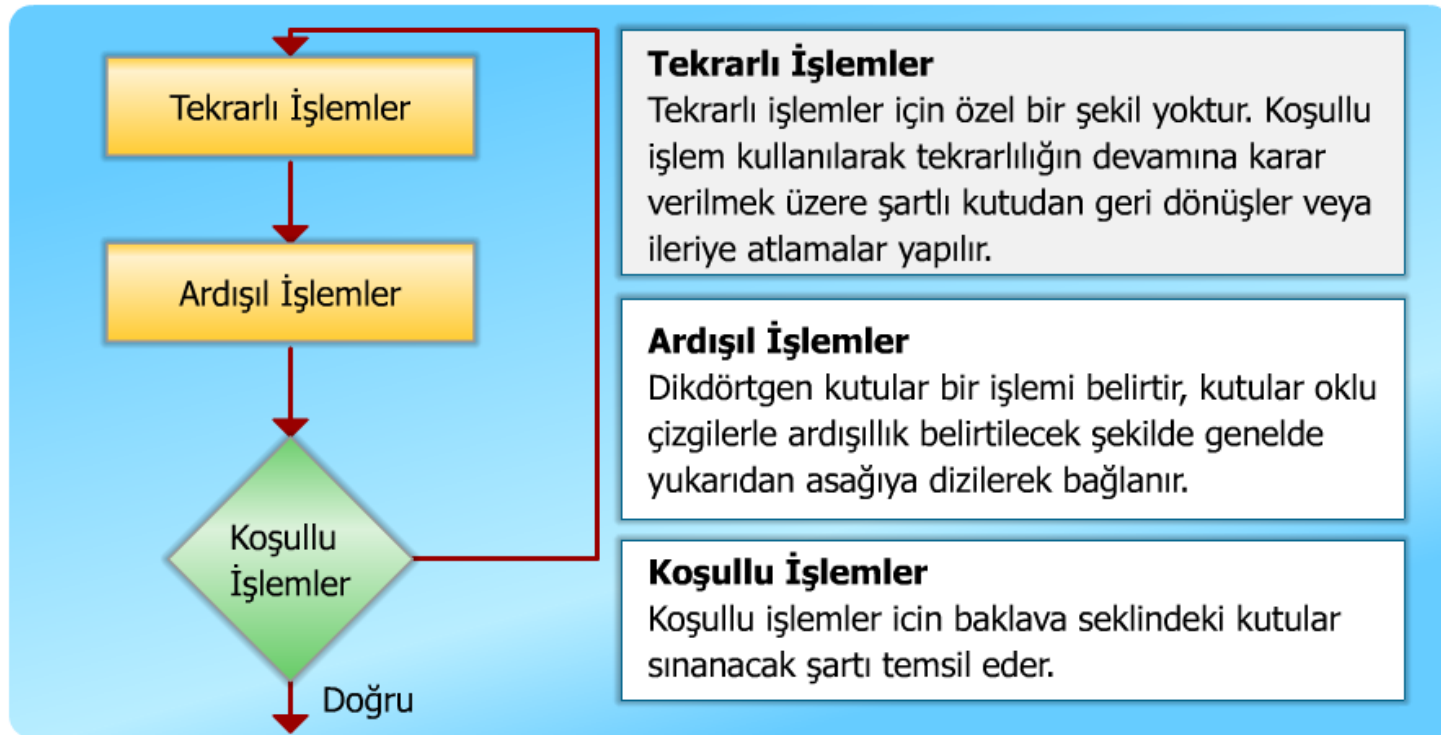
Program Akış Diyagramı Yapıları



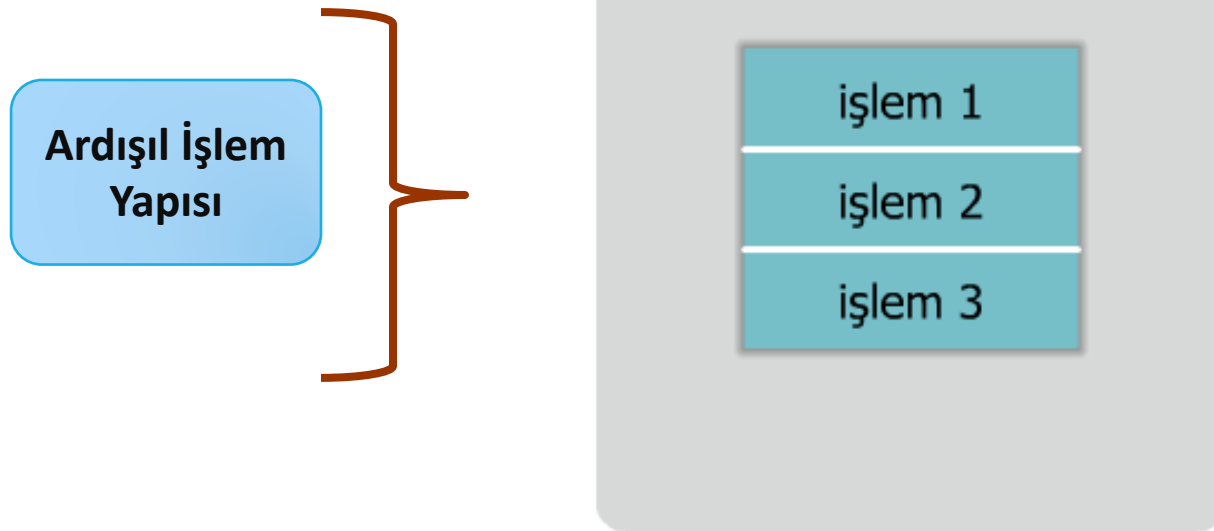
Program Akış Diyagramı Yapıları



Program Akış Diyagramları

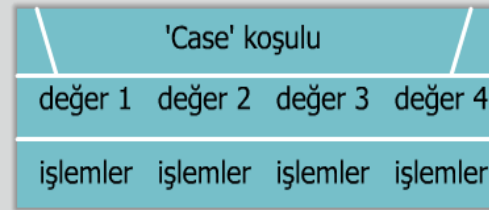
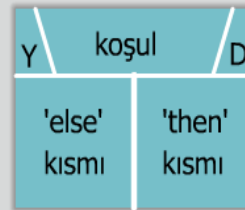


Kutu Diyagramları

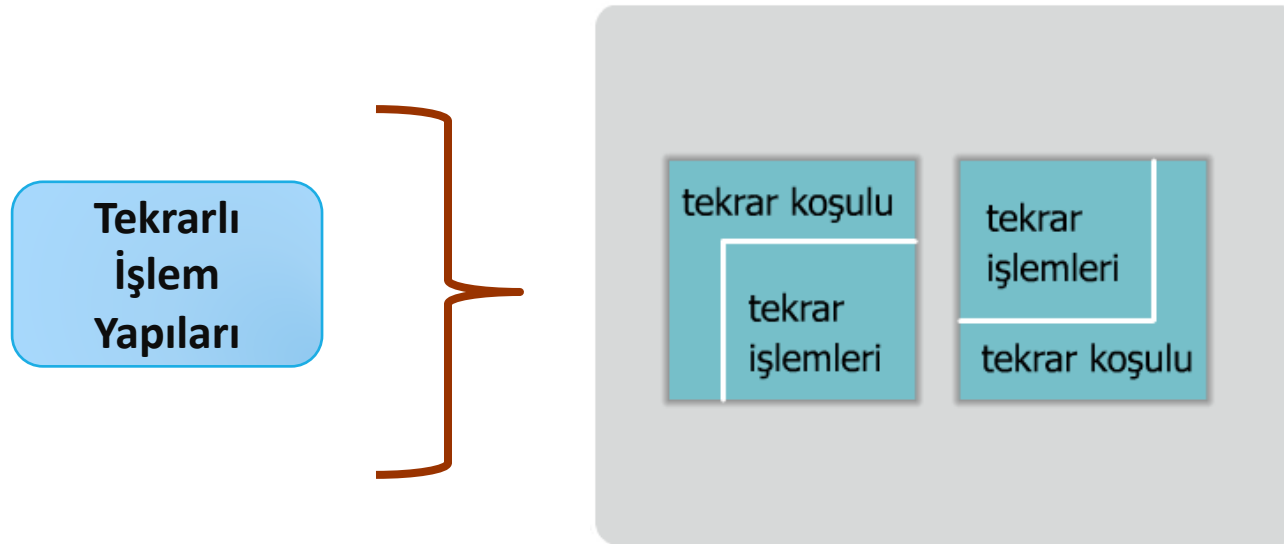


Kutu Diyagramları

Şartlı İşlem Yapısı

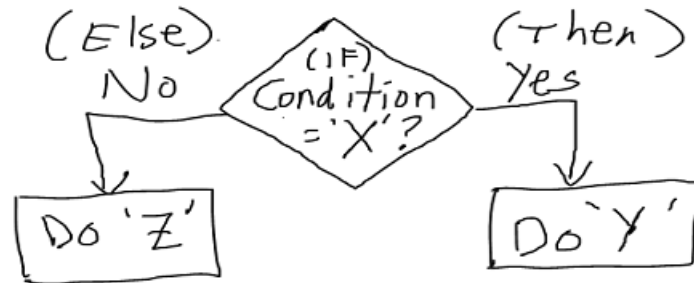


Kutu Diyagramları



Karar Tabloları

- Bazen karmaşık koşul değerlendirmeleri yapmak gerekir. Bunların düzenli bir gösterilimi karar tablolarında yapılabilir.
- Öncelikle, bütün işlemler saptanmalı, sonra ön koşullar belirlenmelidir.
- Belirli işlemler ile belirli koşulları birleştirerek tablo oluşturulur.
- Alt tarafta ise işlemler benzer satırlar olarak gösterilir.



Karar Tabloları

Kurallar	1	2	3	4
Hesap Geçerli	✓	✓	✓	✓
Şifre Geçerli	✓		✓	
Yeterli Nakit Var	✓		✓	✓
Hesapta Yeterli Bakiye Var	✓			✓
Bakiye Bildirme İşlemi	✓			
Para Çekme İşlemi		✓		
Para Yatırma İşlemi			✓	
Para Gönderme İşlemi				

Program Tasarım Dili

- Program Tasarım Dilleri süreç belirtiminde doğal dillerin programlama dili ile sentezlenmesi şeklinde ortaya çıkmıştır.
- Hangi programlama dilinin kullanılacağından bağımsız özellikler bulunmalıdır.

DO

Hesap Numarasını Oku

IF (hesap numarası geçerli değil) başlangıca dön
işlem türünü iste

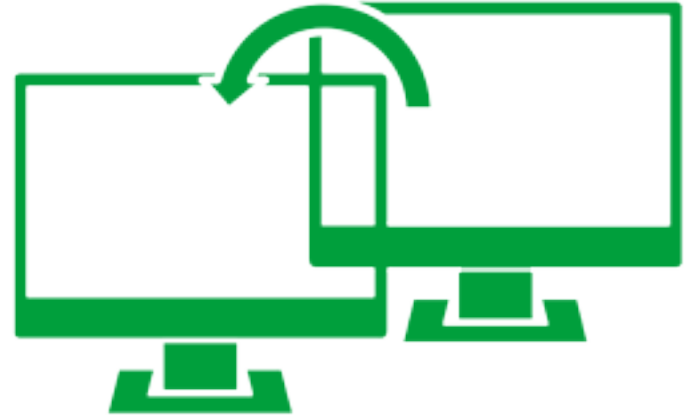
IF (para yatırma işlemi) { para_yatir(); Başlangıca dön}

IF (yeterli bakiye yok) başlangıca dön

WHILE

Tasarlanması Gereken Ortak Alt Sistemler

- Yetkilendirme altsistemi
- Güvenlik altsistemi
- Yedekleme altsistemi
- Veri transferi altsistemi
- Arşiv altsistemi
- Dönüştürme altsistemi



Yetkilendirme Alt Sistemi

- Özellikle kurumsal uygulamalarda farklı kullanıcıların kullanabilecekleri ve kullanamayacakları özellikleri ifade eder.
 - İşlev bazında yetkilendirme
 - Ekran bazında yetkilendirme
 - Ekran alanları bazında yetkilendirme
- Oracle veri tabanına erişim konusunda yetkilendirme yapmaktadır.



Güvenlik Alt Sistemi

- Güvenlik alt sistemi, bilgi sisteminde yapılan işlerin ve yapan kullanıcıların izlerinin saklanması ve gereken durumlarda sunulması ile ilgilidir.
- Bir çok yazılım geliştirme ortamı ve işletim sistemi, bu amaca yönelik olarak, "**sistem günlüğü**" olanakları sağlamaktadır.
- Sistem günlüğü ile sunulan olanaklar yeterli olmadığı durumlarda ek yazılımlar geliştirilmesi gerekmektedir.
- LOG files (Sistem günlüğü)

Yedekleme Alt Sistemi

- Her bilgi sisteminin olağandışı durumlara hazırlıklı olmak amacıyla kullandıkları veri tabanı (sistem) yedekleme ve yedekten geri alma işlemlerinin olması gerekmektedir.
- Günümüzde tüm veri tabanı yönetim sistemi geliştirme platformları, oldukça zengin yedekleme ve yedekten geri alma olanakları sağlamaktadır.
- Bu konuda, tasarım bağlamında yapılması gereken, yedekleme işleminin düzenlenmesini tasarlamak olmalıdır.
- Yedeklemenin hangi sıklıkla yapılacağı, ne zaman, elle ya da otomatik olarak yapılıp yapılmayacağı gibi planlamalar, tasarım aşamasında yapılmalıdır.

Veri İletişim Alt Sistemi

- Coğrafi olarak dağıtılmış hizmet birimlerinde çalışan makineler arasında veri akışının sağlanması işlemleri
- **Çevirim içi veri iletimi (real-time):** Verinin bir birimden diğerine anında iletilmesi olarak tanımlanır.
 - Bu tür veri iletişimi, gerçek zamanlı sistemler için oldukça önemlidir.
 - Bilgi sistemi uygulamalarında, - zamansal kritiklik, gerçek zamanlı uygulamalara oranla daha az olduğu için - bu tür iletişim çok yaygın değildir.
- **Çevirim dışı veri iletimi (disketler, teypler):** Bilgilerin iletişim hatları kanalıyla değil, çevrim dışı ortamlar (teyp, disket, cd vb.) aracılığı ile iletilmesi çevrim dışı iletişim olarak tanımlanmaktadır.
 - Kısacası, "kargo" olarak tanımlanan iletişimin, el ile yapılan türüdür.
 - Ağ iletişiminin sağlanamadığı durumlarda kullanılan bir yöntemdir.
 - Kullanımı giderek azalmaktadır.

Arşiv Alt Sistemi

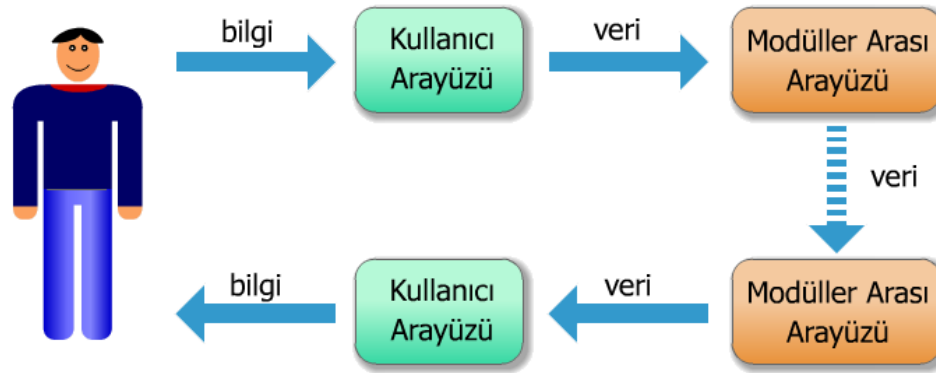
- Belirli bir süre sonrasında sık olarak kullanılmayacak olan bilgilerin ayrılması ve gerektiğinde bu bilgilere erişimi sağlayan alt sistemlerdir.
- Örneğin, insan kaynakları yönetimi bilgi sisteminde, emekli olan bir kişiye ilişkin bilgilerin, çevrim-içi olarak tutulan veri tabanından alınarak, çevrim dışı bir ortama alınması ve aradan örneğin beş yıl geçtikten sonra, pasaport işlemleri için gerek duyulabilecek kişi bilgilerine erişilmesini sağlayan işlemler arşiv alt sistemleri tarafından gerçekleştirilmektedir.
- İşlem türü olarak ortak bir çok özellik içeren arşiv alt sistemleri, uygulama bazında az da olsa farklılaşabilmektedir.
- Aktif veri tabanı.

Dönüştürme Alt Sistemi

- Geliştirilen bilgi sisteminin uygulamaya alınmadan önce veri dönüştürme (mevcut sistemdeki verilerin yeni bilgi sistemine aktarılması) işlemlerine ihtiyaç vardır.
- Mevcut uygulamalardaki bilgisayar ortamında saklanan bilgilerin ortam çeşitliliği, dönüştürme işlemlerini zorlaştırır.



Kullanıcı Arayüz Tasarımı



- Kullanıcı ile ilişkisi olmayan arayüzler
 - Modüller arası arayüz
 - Sistem ile dış nesneler arası arayüz
- Kullanıcı arayüzleri
 - Kullanım kolaylığı ve öğrenim zamanı esastır.
 - Program = arayüz yaklaşımı vardır.

Genel Prensipler

- Komut seçimi, veri giriş formlarının şekli gibi bir çok konuda tutarlı bir yapı izlenmelidir.
- Önemli silmelerde teyit alınmalıdır.
- Yapılan çoğu işlem kolayca geri alınabilmelidir
- İşlemler arasında ezbere tutacak bilgi miktarı azaltılmalıdır.
- Kullanıcı hareketleri, düşünme ve algılamasında verimlilik sağlanmalıdır.
- Hataların affedilmesi, yanlış giriş olduğunda program korunmalı ve düzeltme şansı verilmelidir.
- İşlemleri sınıflandırıp ekran geometrisi buna uygun olarak kullanılmalıdır.
- Komut isimleri kısa ve basit olmalıdır.
- Menülerin ve diğer etkileşimli araçların standart yapıda tasarlanmalıdır.

Bilgi Gösterimi

- Yalnızca içinde bulunulan konu çerçevesi ile ilgili bilgi gösterilmeli
- Veri çokluğu ile kullanıcı bunaltılmamalı, grafik ve resimler kullanılmalı
- Tutarlı başlık, renkleme ve kısaltma kullanılmalı
- Hata mesajları açıklayıcı ve anlaşılır olmalı
- Değişik tür bilgiler kendi içinde sınıflandırılmalı
- Rakamsal ifadelerde analog görüntü verilmeli (%89 değil)



Veri Girişİ

- Kullanıcı hareketleri en aza indirilmelidir. Yazma yerine ekrandaki listelerden seçme, bir komuta en az sayıda fare tıklamasıyla erişme gibi.
- Gösterim ve girdi sahaları birbirinden ayırt edecek biçimler (renk, büyüklük, yerleşim vb.) tutarlı olarak kullanılmalıdır.
- Kullanıcı uyarlamasına izin verilmelidir: kullanıcı bazı özellikleri tanımlayabilir, bazı uyarı mesajlarını istemeyebilir.
- Kullanılan konu ile ilgili gereksiz komutlar geçici olarak etkisizleştirilmelidir.
- Bütün girdiler için yardım kolaylıkları olmalıdır.



Kullanıcı Arayüz Prototipi

- Tasarım çalışması sonucunda, daha önceden gereksinim çalışması sırasında hazırlanmış olan kullanıcı arayüz prototipi, ekran ve rapor tasarımları biçimine dönüşür. Ekranlar son halini alır, raporlar kesinleşir. Kullanıcıya gösterilerek onay alınır.
- Tüm programın tek elden çıktığının ifade edilebilmesi açısından tüm ekranların aynı şablon üzerine oturtulması önerilmektedir.
 - **Menü Çubuğu**
 - **Araç Çubuğu**
 - **Gövde (Değişebilir)**
 - **Durum Çubuğu**

Başlangıç Tasarım Gözden Geçirme

- Yapılan tasarım çalışmasının bir önceki geliştirme aşaması olan analiz aşamasında belirlenen gereksinimleri karşılayıp karşılamadığının belirlenmesidir.
 - Sistem gereksinimlerine yardımcı olan kullanıcılar
 - Sistem analizini yapan çözümleyiciler
 - Sistemin kullanıcıları
 - Tasarımcılar
 - Yönlendirici
 - Sekreter
 - Sistemi geliştirecek programcılardan oluşan bir grup tarafından yapılır.

Ayrıntılı Tasarım Gözden Geçirme

- Başlangıç tasarımı gözden geçirme çalışmasının başarılı bir biçimde tamamlanmasından sonra, tasarımın teknik uygunluğunu belirlemek için **Ayrıntılı Tasarım Gözden Geçirme** çalışması yapılır. Bu çalışmada;
 - Çözümleyiciler
 - Sistem Tasarımcıları
 - Sistem Geliştiriciler
 - Sekreterden oluşan bir ekip kullanılır.



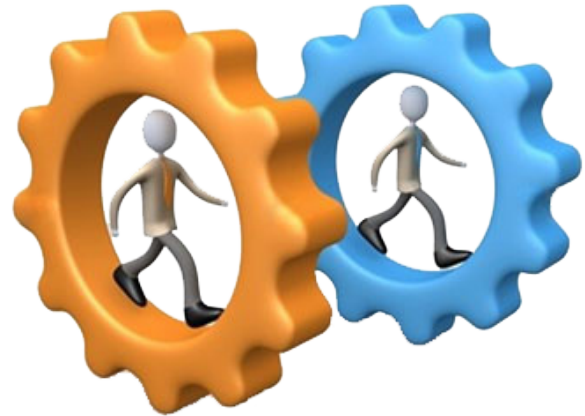
Tasarım Kalite Ölçütleri

- **Bağlaşım (Coupling)**

Tasarımı oluşturan modüller arası ilişki ile ilgilidir.

- **Yapışıklık (Cohesion)**

Modüllerin iç yapısı ile ilgilidir.



Bağlaşım

- Modüller arası bağılılığın ölçülmesi için kullanılan bir ölçüttür.
- Yüksek kaliteli bir tasarımda bağlaşım ölçümü az olmalıdır.
- Bağlaşımın düşük olması
 - Hatanın dalgasal yayılma özelliğinin azaltılması
 - Modüllerin bakım kolaylığı
 - Modüller arası ilişkilerde karmaşıklığın azaltılmasınedenleri ile istenmektedir

Yalın Veri Bağlaşımı

- Herhangi iki modül arası iletişim yalın veriler (tamsayı, karakter, boolean, vs) aracılığı ile gerçekleştiriliyorsa bu iki modül yalın veri bağlaşımlıdır şeklinde tanımlanır.



Karmaşık Veri Bağlaşımı

- Herhangi iki modül arasındaki iletişimde kullanılan parametrelerin karmaşık veri yapısı (kayıt, dizi, nesne, vs) olması durumunda modüller karmaşık veri paylaşımı olarak tanımlanır.



Denetim Baęlaşımı

- İki Modül arasında iletişim parametresi olarak **denetim verisi** kullanılıyorsa bu iki modül denetim baęlaşımli olarak tanımlanır.



Ortak Veri Bağlaşımı

- Eğer iki modül ortak bir alanda tanımlanmış verilere ulaşabiliyorsa bu iki modül **ortak veri bağlaşımlı** olarak tanımlanır.
- Verilerin ortak veri bağlaşımlı olmaları şu nedenlerden dolayı fazla istenmez;
 - Ortak veri alanını izlemek zordur.
 - Ortak veri kullanan modüllerde yapılan değişiklikler diğer modülleri etkiler.
 - Ortak veri üzerinde yapılacak değişikliklerde bu veriyi kullanacak bütün modüller göz önüne alınmalıdır.

İçerik Bağlaşımı

- Modüllerin iç içe tasarlanması sonucu, bir modülün başka bir modül içerisinde tanımlanmış veri alanına erişebilmesi olanaklaşır ve bu durum **içerik bağlaşımına** yol açar.

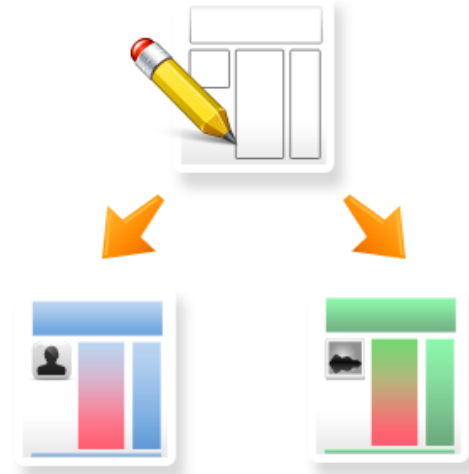


Yapışıklık

- Bir modülün kendi içindeki işlemler arasındaki ilişkilere ilişkin bir ölçüttür. **Modül gücü** olarak ta tanımlanır.
- Tasarımda **yapışıklık** özelliğinin yüksek olması tercih edilir.
- Yapışıklık ile Bağlaşım ters orantılıdır.

İşlevsel Yapışıklık

- İşlevsel Yapışık bir modül, tek bir iş problemine ilişkin sorunu çözen modül olarak tanımlanır.
- Maas_Hesapla, Alan_Hesapla gibi



Sırasal Yapışıklık

- Bir modülün içindeki işlemler incelendiğinde, bir işlemin çıktısı, diğer bir işlemin girdisi olarak kullanılıyorsa bu modül **sırasal yapışık** bir modül olarak adlandırılır.

Ham_Veri_Kaydını_Düzeltil

Düzeltilmiş_Ham_Veri_Kaydını_Dogrula

Dogrulanmis_Kaydi_Gonder

İletişimsel Yapışıklık

- Bir modülün içindeki farklı işlemler aynı girdi ya da çıktıyı kullanıyorlarsa bu modül **iletişimsel yapışık** bir modül olarak adlandırılır.

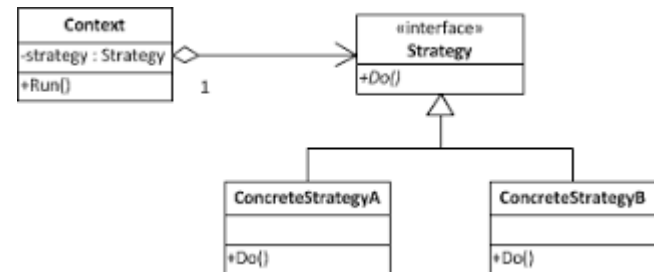
Sicil_No_yu_Al
Adres_Bilgisini_Bul
Telefon_Bilgisini_Bul
Maas_Bilgisini_Bul



Yordamsal Yapışıklık

- Yordamsal Yapışık modüldeki işlemler arasında denetim ilişkisi bulunmaktadır.
- İşlemlerin birbirleri ile veri ilişkisi yoktur, ancak işlem sırası önemlidir.

Ekran_Goruntusunu_Yaz
Giris_Kaydini_Oku



Zamansal Yapışıklık

- Bir modül içindeki işlemlerin belirli bir zamanda uygulanması gerekiyor ve bu işlemlerin kendi aralarında herhangi bir ilişkisi yok, yani işlemlerin sırası önemli değil ise, **zamansal yapıışıklık** vardır.

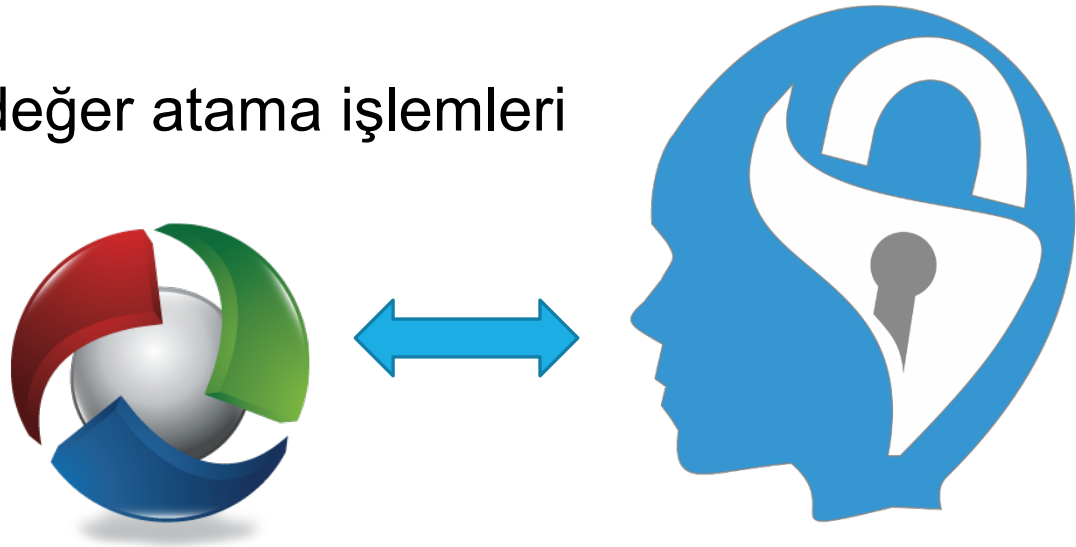
Alarm_Zilini_Ac
Kapiyi_Ac
Kamerayi_Calistir



Mantıksal Yapışıklık

- Mantıksal olarak aynı türdeki işlemlerin bir araya toplandığı modüller **mantıksal yapışık** olarak adlandırılır.

Dizilere değer atama işlemleri



Gelişigüzel Yapışıklık

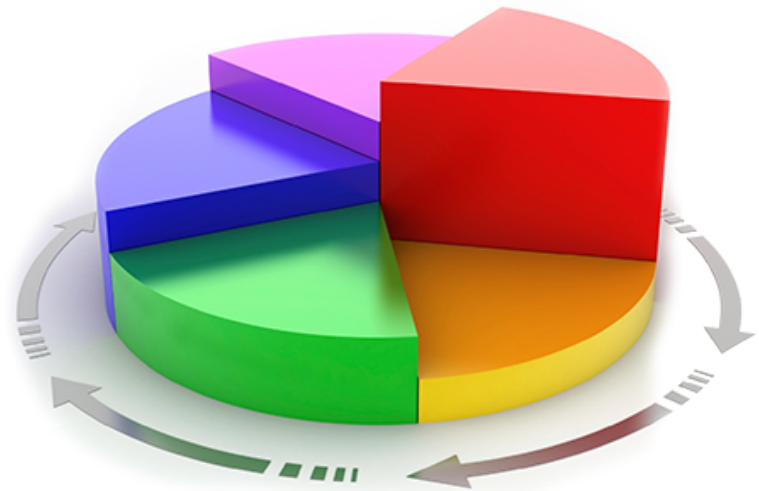
- İşlemler arasında herhangi bir ilişki bulunmaz.

Ara_Kayit_Oku

B_dizisine_baslangic_de

Stok_kutugu_oku

Hata_iletisi_yaz



Özet

- Tasarım, Sistem Analizi çalışması sonucunda üretilen Mantıksal Modelin Fiziksel Modele dönüştürülme çalışmasıdır.
- Tasarım kavramları: Soyutlama, İyileştirme ve Modülerlik olmak üzere 3 çeşittir.
- Yapı Tasarımı, arayüz tasarımı ve süreç tasarımından önce yapılması gereken ilk tasarım veri tasarımıdır.
- Veri Akışları Üç parçada incelenebilir: Girdi Akışı, Çıktı Akışı ve İşlem Akışı
- Süreç tasarımı; veri, yapı ve ara yüz tasarımından sonra yapılır.
- Program Akış Diyagramları: Tekrarlı, ardışıl ve koşullu şeklindedir.

Tasarlanması Gereken Ortak Alt Sistemler;

- Yetkilendirme altsistemi
- Güvenlik altsistemi
- Yedekleme altsistemi
- Veri transferi altsistemi
- Arşiv altsistemi
- Dönüştürme altsistemi şeklindedir.

Sorular

1. Yazılım tasarım sürecinin temel işlemlerini sayınız. E-R diyagramı çizerek ilişkilerini gösteriniz.
2. Geliştireceğiniz bir uygulama için ekran şablonunuzu belirleyiniz.
3. Bağlaşım ve yapışıklık kavramlarını açıklayınız. Program bakımı ile ilişkilerini belirtiniz.
4. Bir sistem tümüyle bağışımsız biçimde tasarlanabilir mi? Yani sistemin tüm modülleri arasında hiç bağlaşım olmadan tasarım yapılabilir mi?
5. Tümüyle işlevsel yapışık modüllerden oluşan bir sistem tasarlanabilir mi? Neden yapılabilir? Neden yapılamaz?
6. Bağlaşım ile Yazılım Taşınabilirliği arasındaki ilişkiyi belirtiniz.
7. Tasarım gözden geçirmenin önemi nedir? Yapılmaması ne tür sonuçlara yol açar?
8. Arşiv alt sistemi ile yedekleme alt sistemi arasındaki benzerlikleri ve farklılıkları belirtiniz.
9. Tasarım ile sınaama arasındaki ilişkiyi belirtiniz.
10. 13. Geliştirdiğiniz bir uygulama tasarımı için
 - (a) Bir birim sınaama belirtimi
 - (b) İki sistem sınaama belirtimi (senaryo) hazırlayınız.

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş ”, TBİL-211, Dr. Ali Arifoğlu.

” Yazılım Mühendisliği ” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

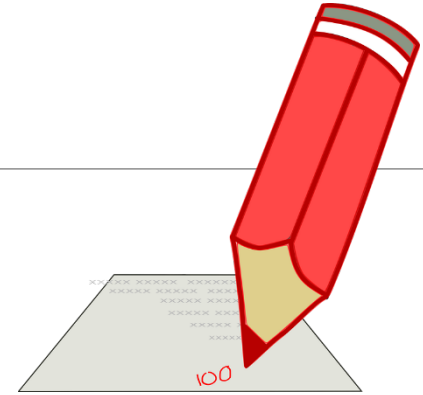
Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Ödev

Yazılım Tasarım Kalıpları hakkında araştırma yapınız.



Sorularınız





YMH311-Yazılım Tasarım Ve Mimarisi

Mimari Tasarım Çözümlemesi

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Mimari Oluşturma Teknikleri.. ..	7
AquaLush Kalite Niteliklerine Göre Ayırıştırma	15
SAD kalite karakteristikleri.....	32
Profiller.....	23
Gözden Geçirmeler (Reviews).....	34

GENEL BAKIŞ

KISIM 1 – Yazılım Mimarilerinin Oluşturulması, Değerlendirilmesi, İyileştirilmesi ve Seçilmesi

KISIM 2 – Yazılım Mimarilerinin Tamamlanması



KISIM 1

Yazılım Mimarilerinin Oluřturulması, Deęerlendirilmesi, İyileřtirilmesi ve Seęilmesi



Amaçlar

- Mimari oluşturma tekniklerini araştırmak ve göstermek
- Sanal aygıt (virtual device) fikrini tanıtmak
- Mimari iyileştirme tekniklerini araştırmak
- Mimarilerin değerlendirilmesi için senaryo ve profillerin kullanımını göstermek
- Mimarilerin değerlendirilmesi için prototiplerin kullanımını göstermek



İçerik

- Mimari oluşturma teknikleri
 - İşlevsel ayrıştırma (Functional decomposition)
 - Kalite nitelikleri bazlı ayrıştırma (Quality-attribute-based decomposition)
- Sanal aygıtlar (virtual device) ve aygıt arabirim modülleri
- Mimari iyileştirme teknikleri
- Mimari alternatiflerinin değerlendirilmesi
 - Senaryolar, profiller, fayda ağaçları
 - Prototipler
- Mimari alternatiflerin seçilmesi



Mimari Oluşturma Teknikleri

- **İşlevsel Bileşenleri Belirleyin**—İşlevsel ve veri gereksinimlerini bütünleşik biçimde gerçekleştirilmesinden sorumlu bileşenler oluşturun.
- **Bileşenleri Kalite Niteliklerine Göre Belirleyin**—İşlevsel olmayan gereksinimleri karşılayacak bileşenler oluşturun, ardından kalan işlevsel ve veri gereksinimi boşluklarını dolduracak bileşenler ekleyin.

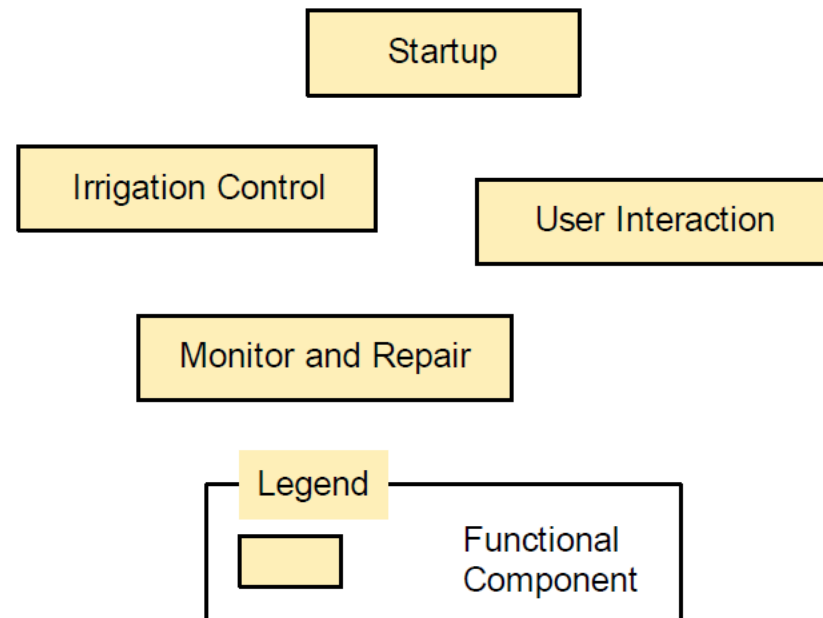


Mimari Oluřturma Teknikleri

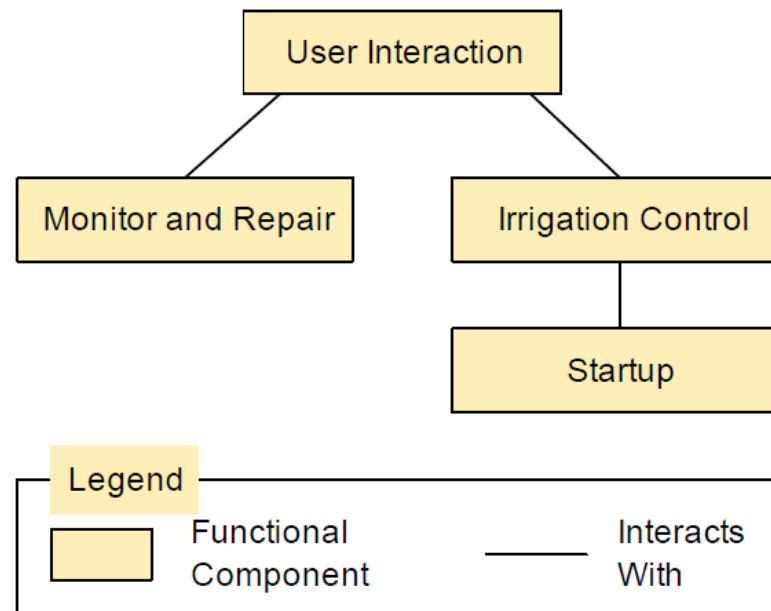
- **Varolan Bir Mimaride Deęişiklik Yapın**—Benzer bir programın mimarisini deęiřtirerek kullanın.
- **Bir Mimari Stilini Ayrıntılandırın**—Bir mimari stili (architectural style), program veya sistemi oluřturan bileřen tiplerinin ve bunların etkileřimlerinin bir paradigmasıdır. Bir mimari stili ayrıntılandırarak bir mimari oluřturun.
- **Kavramsal Bir Modeli Dönüřtürün**—Bir problemin kavramsal modelinde deęişiklik yaparak çözüme yönelik bir model oluřturun.



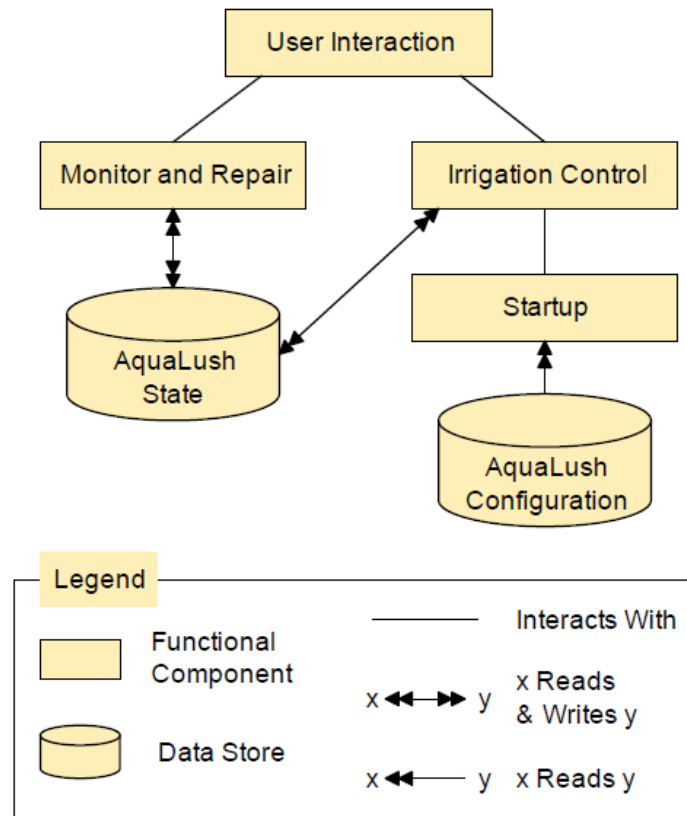
AquaLush İşlevsel Ayırıştırma 1



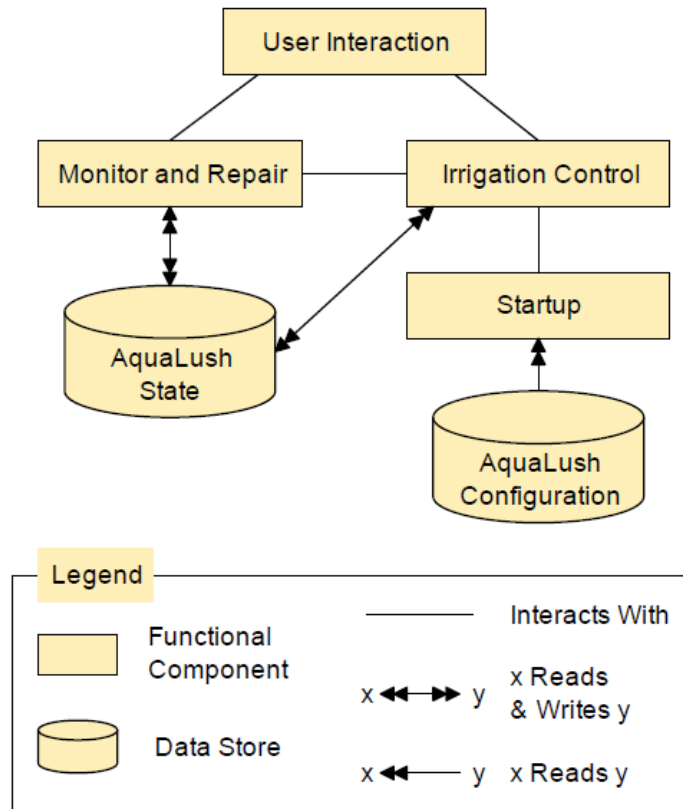
AquaLush İşlevsel Ayırıştırma 2



AquaLush İşlevsel Ayırıştırma 3



AquaLush İşlevsel Ayırıştırma 4



Uyarlanabilirlik Sağlamak (Adaptability)

- Donanım uyarlanabilirliği sağlamanın standart bir yolu sanal aygıtlar içeren bir aygıt arabirim modülü (device interface module) kullanmaktır.

Bir **sanal aygıt** (virtual device), gerçek bir donanım aygıtı veya sisteminin yazılımsal bir **simülasyonu** ya da aygıt/sisteme bir **arabirim**dir.

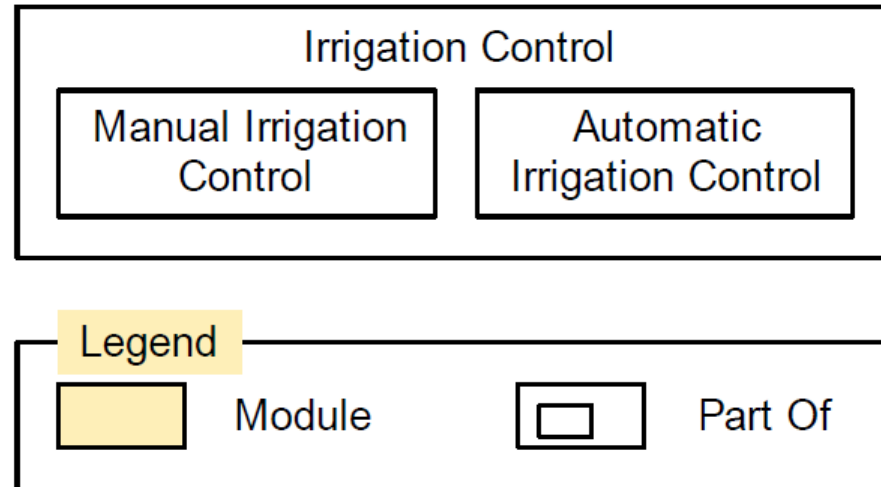
Sanal Aygıt Karakteristikleri

“İdeal” bir cihazı simüle eder

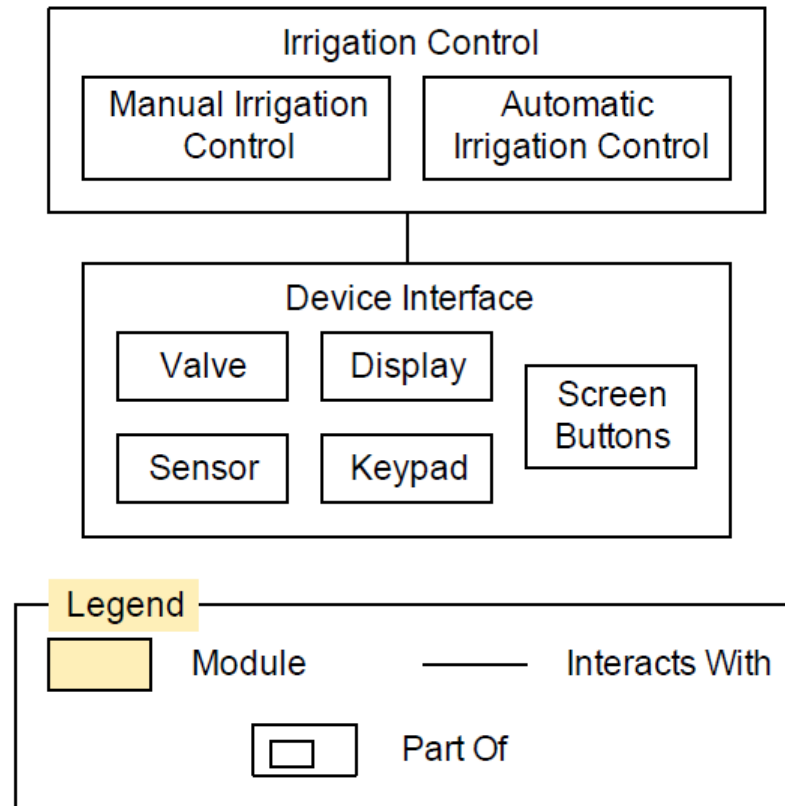
- Tamamen ve sadece bir işi yapar (uyum/cohesion)
- Basit, tutarlı, eksiksiz bir arabirime sahiptir (basitlik/simplicity)
- Programın geri kalanıyla gevşek şekilde bağlıdır (bağlılık/coupling)
- İmplementasyonunu saklar (bilgi saklama/information hiding)
- Asla değişmez (stabilite/stability)

Farklı gerçek aygıtlar veya sistemler için genellikle bir bileşen ailesi olarak gerçekleştirilir.

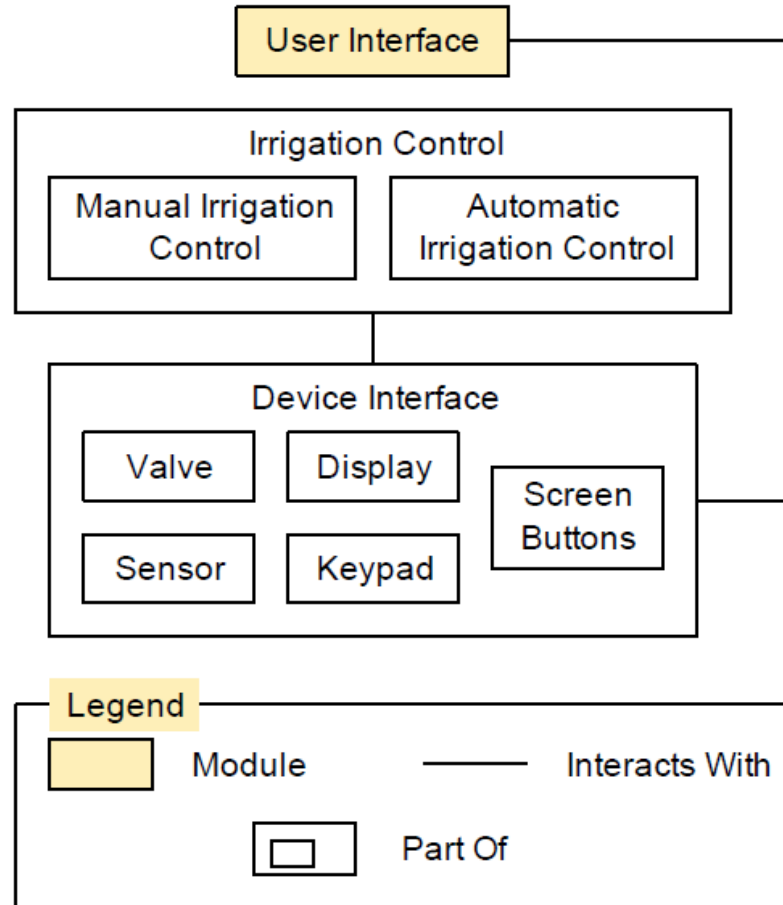
AquaLush Kalite Niteliklerine Göre Ayırıştırma 1



AquaLush Kalite Niteliklerine Göre Ayırıştırma 2



AquaLush Kalite Niteliklerine Göre Ayırıştırma 3



Alternatiflerin İyileştirilmesi

- **Alternatifleri Kombine Edin**—İki veya daha fazla alternatifin en iyi özelliklerini kombine edin
- **Bir Mimari Stilini Empoze Edin**—Bir tasarım stiline çok yakın olan bir mimaride değişiklik yaparak ilgili stile tam uygun olmasını sağlayın
- **Orta-düzey Tasarım Kalıplarını Uygulayın**—Orta-düzey tasarım kalıplarının avantajlarından yararlanmak üzere mimaride değişiklik yapın



Alternatiflerin Değerlendirilmesi

- Tasarımcılar bir mimari spesifikasyona göre oluşturulan bir programın gereksinimleri karşılayacağını program daha oluşturulmadan nasıl bilebilirler?
- Bunun nasıl garanti edileceğini kimse bilemez, ancak olasılığı arttırıcı bazı teknikler kullanılabilir.
- Mimarilerin değerlendirilmesi için senaryoların ve prototiplerin kullanımını inceleyeceğiz.



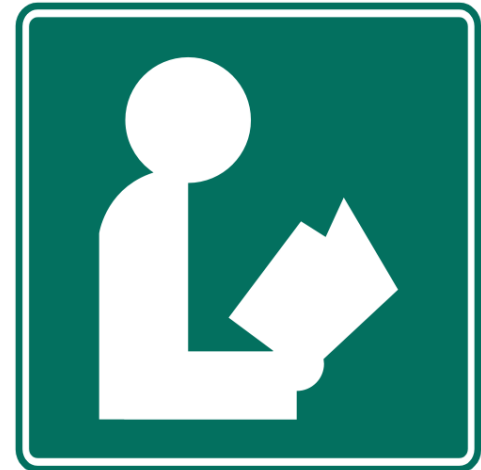
Senaryolar

Bir **senaryo** bir ürün ile belirli bireyler arasındaki bir etkileşimdir.

- Use-case örnekleri bir ürün ve aktörler arasındaki etkileşimlerdir
- Daha geniş bir bakış gerekli çünkü şimdi bir ürün ve herhangi bir birey arasındaki etkileşimleri dikkate alıyoruz.

Senaryo Yazma Kuralları 1

- Her bir senaryoyu açıklayıcı bir ifadeyle etiketleyin.
- Etken çatı ile basit, bildirimsel cümleler yazın.
- Senaryo açıklamalarını üç bölüm halinde yazın
 - Ürünün ve ortamının başlangıç (initial) durumu
 - Ürün ve bireyler arasındaki aktivite akışı (activity flow)
 - Ürünün ve ortamının son (final) durumu



Senaryo Yazma Kuralları 1

- Aktivite akışını tanımlayan cümlede, etkileşimin esas tarafını cümlenin öznesi yapın.
- Hem ürünün hem de ürünün ortamının başlangıç ve son durumlarını açıklayın.
- Mümkün olduğunca hedeflenen çıktı ölçütlerini belirtmeye çalışın.
- Açıklamayı tekrar okuyun ve hata varsa düzeltin.



Profiller

Bir **profil** bir ürünün bir grup gereksinimi karşılayıp karşılamayacağını değerlendirmek için kullanılan senaryo kümesidir.

Örneğin: kullanım profili, güvenilirlik profili

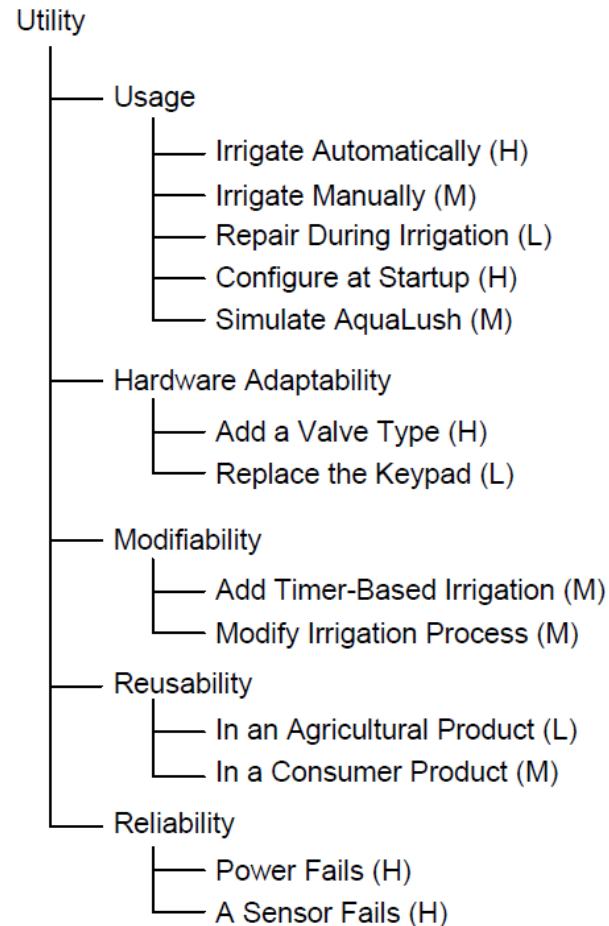
Profil içindeki senaryoların ağırlıkları olmalıdır

Profiller genellikle 3-10 arasında senaryodan oluşur.

Profil ve Senaryo Oluşturma

- Bir **fayda ağacı** (utility tree), dalları profiller ve yaprakları senaryolar olan bir ağaçtır.
 - Kökü “fayda” olarak etiketleyin.
 - Ürün gereksinimlerini yansıtacak şekilde isimlendirilmiş dallar ekleyin.
 - Her bir profil için senaryolar oluşturun.
 - Beyin fırtınası ile senaryolar oluşturun
 - Senaryo listesini rasyonelize edin
 - Her bir senaryoya bir ağırlık verin
 - Her bir profilde 3-10 arası senaryo kalana kadar düşük ağırlıklı senaryoları eleyin
 - Senaryo açıklamaları yazın.

Örnek Bir Fayda Ağacı



Senaryolarla Değerlendirme ve Seçim

- Her bir senaryonun üzerinden giderek
 - Tasarım alternatiflerinin senaryoyu ne kadar iyi desteklediğini değerlendirin.
 - Her bir senaryo için değerlendirme sonucunu kaydedin.

- Uygun bir teknik kullanarak alternatifler arasından seçim yapın.
 - Avantaj ve dezavantajlarına göre
 - Çok-boyutlu sıralama
 - Senaryo ağırlıkları normalize edilir
 - Değerlendirmeler sayısallaştırılır

Prototiplerle Değerlendirme ve Seçim

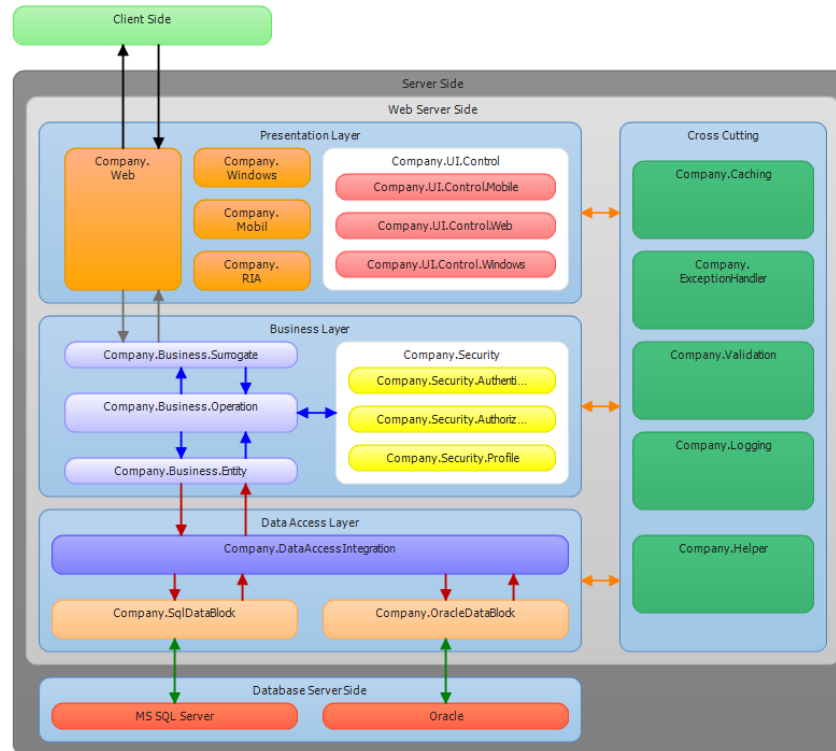
- Tasarım alternatiflerini sınamak için prototipler oluşturulabilir.
- Senaryoların üzerinden ilerlerken prototiplendirme ihtiyacı ortaya çıkabilir.
- Prototipler seçim yapabilmek için somut bir dayanak oluşturabilir.
 - Avantaj ve dezavantajlarına göre
 - Çok-boyutlu sıralama

Özet

- Mimari alternatifler oluşturmak ve iyileştirmek için birbirini tamamlayan çeşitli teknikler kullanılabilir.
- Ağırlıklandırılmış senaryolar içeren profiller oluşturmak ve bunlar üzerinden ilerlemek mimari alternatiflerini değerlendirmek için kullanılabilecek güvenilir bir tekniktir.
- Prorotipler de mimari değerlendirme için veri sağlayabilir.

KISIM 2

Yazılım Mimarilerinin Tamamlanması



Amaçlar

SAD kalite karakteristiklerini sunmak

Çeşitli gözden geçirme (review) türlerini araştırmak

Örnek bir mimari inceleme kontrol listesi sunmak

Aktif tasarım gözden geçirme sürecini ayrıntılı olarak sunmak

Tasarım sürecinde sürekli gözden geçirmenin (continuous review) önemini vurgulamak



İçerik

- SAD kalite karakteristikleri
- Gözden geçirmeler (Reviews)
- Gözden geçirme türleri
- Bir mimari inceleme kontrol listesi
- Aktif tasarım gözden geçirme süreci
- Sürekli gözden geçirme



SAD Kalite Karakteristikleri 1

- **Yapılabilirlik (Feasibility)**—SAD gerçekleştirilebilir (yapılabilir) bir programı belirtir.
- **Yeterlilik (Adequacy)**—SAD gereksinimleri karşılayacak bir programı belirtir.
- **İyi Biçimlendirilmişlik (Well-Formedness)**—SAD içindeki notasyonlar doğru şekilde kullanılmıştır.

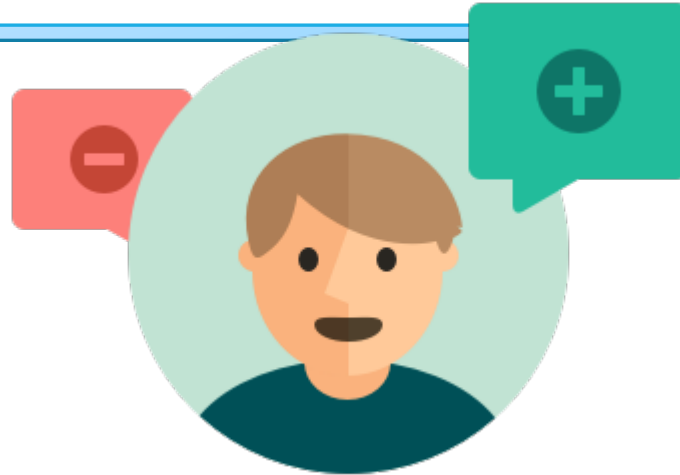


SAD Kalite Karakteristikleri 1

- **Tamlık/Bütünlük (Completeness)**—SAT gereken tüm bölümleri içerir; tasarımı açıklamak için gereken tüm modelleri içerir; ve önemli tüm bileşen karakteristiklerini, ilişkilerini, etkileşimlerini vb. belirtir.
- **Netlik (Clarity)**—Probleme ve notasyonlara aşina herkes SAD dokümanını anlayabilir.
- **Tutarlılık (Consistency)**—Tek bir program SAD ile belirtilenleri karşılayabilir.

Gözden Geçirmeler (Reviews)

Gözden geçirme (review), bir ürünün ya da sürecin kalifiye elemanlardan oluşan bir ekip tarafından incelenmesi ve değerlendirilmesidir.



Gözden Geçirme Türleri

- **Desk Check**—Tasarımın bir tasarımcı tarafından değerlendirilmesidir
- **Walkthrough**—Tasarımın bir tasarımcı ekibine resmi olmayan bir biçimde sunulmasıdır
- **Inspection**—Eğitimli bir inceleme ekibi tarafından yapılan resmi bir gözden geçirmedir
- **Audit**—Tasarım ekibinin dışından çeşitli uzmanlar tarafından gerçekleştirilen gözden geçirmedir
- **Active Review**—Tasarımla ilgili spesifik soruları cevaplayan uzmanlar tarafından yapılan incelemedir

Bir Mimari İnceleme Kontrol Listesi

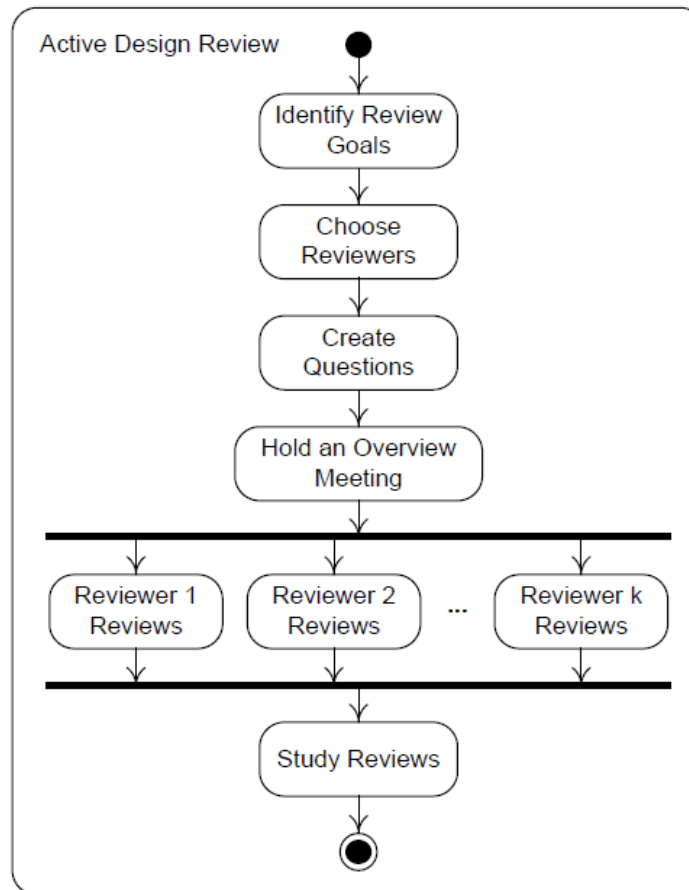
- ☐ The notations used for each model are correct.
- ☐ Every required section of the SAD is present.
- ☐ The SAD specifies the program's main components.
- ☐ The SAD specifies the states and state transitions for all components with important states.
- ☐ The SAD specifies important or complex component collaborations.
- ☐ The SAD specifies each component's responsibilities.
- ☐ The SAD specifies each component's interface.
- ☐ The SAD specifies each component's important properties.
- ☐ The SAD specifies each component's important relationships to other components.
- ☐ The SAD clearly states the connections between different architectural models.
- ☐ The SAD states the rationale for all important design decisions.
- ☐ Each design rationale states the problem to be solved and the constraints on the designer.
- ☐ Each design rationale summarizes the major design alternatives and their evaluations.
- ☐ Each design rationale explains why the final design was selected.
- ☐ All specifications are clear.
- ☐ No specification contradicts any other specification in the SAD.

Aktif Tasarım Gözden Geçirme

- Geleneksel gözden geçirmelerin problemlerine çare olur
 - Uzmanlık eksikliği
 - Gelişigüzel gözden geçirme
- Gözden geçirenleri dokümanın kendi uzmanlıklarıyla ilgili bölümlerini iyi anlamaya zorlar; bu kişiler tasarımın ayrıntılarıyla ilgili spesifik cevaplar verebilmelidir



Aktif Tasarım Gözden Geçirme Süreci



Gözden Geçirmeye Hazırlık

- **Gözden Geçirme Amaçlarını Belirleyin**—Tasarımcılar tasarımın kontrol edilmesi gereken yönlerini belirler.
- **Gözden Geçirme Ekibini Seçin**—Tasarımcıları 2-4 kişilik kalifiye ekip elemanı belirleyip inceleme yapmak üzere onaylarını alır.
- **Sorular Hazırlayın**—Tasarımcılar gözden geçiriciler tarafından cevaplanmak üzere sorular oluşturur.
 - Gözden geçiricileri tasarımı anlamaya zorlamak için
 - Gözden geçiricilere problem çözmeleri veya bir şeyi açıklamaları için vb. sorular sorulur.

Gözden Geçirmenin Gerçekleştirilmesi

- **Ön bilgilendirme Toplantısı Düzenleyin**—Tasarımcılar mimariyi, süreçleri vb. açıklarlar, son teslim tarihi belirlerler, vb.
- **Gözden Geçirme İşlemini Yapın**—Gözden geçiriciler işlerini kendi başlarına yaparlar.
 - Soru sormak, açıklama ve açıklık getirmek gibi amaçlarla tasarımcılarla görüşebilir, e-posta ile haberleşebilir vb.
 - Tamamlanınca sonuçları iletirler.

Gözden Geçirmenin Tamamlanması

- **Gözden Geçirme Sonuçlarını Çalışın**—Tasarımcılar gözden geçirme sonuçlarını çalışırlar.
 - Gözden geçiricilerle buluşarak ya da eposta ile sonuçlar üzerinde görüşebilirler.



Sürekli Gözden Geçirme

- **Bir hata (defect) ne kadar çabuk düzeltilirse o kadar ucuza düzeltilir.**
- Tasarım artefact'ları tamamlandığında son kalite kontrol için mutlaka gözden geçirme yapılmalıdır.
- Gözden geçirmeler olası hataları erken yakalamak için tasarım süreci boyunca da yapılmalıdır.
- Tasarım sürecinin farklı aşamalarında farklı tür gözden geçirmeler uygulanabilir.



Özet

- Bir SAD dokümanı büyük ölçüde tamamlandığında mutlaka gözden geçirilmelidir:
 - Yapılabilir ve yeterli bir mimari belirtiyor mu?
 - İyi biçimlendirilmiş modellerden oluşuyor mu?
 - Tam, açık ve tutarlı mı?
- Çeşitli gözden geçirme türleri kullanılabilir.
- Aktif gözden geçirme oldukça etkili bir gözden geçirme şeklidir.
- Tasarım artefact'ları tamamlandıkça ve tasarım süreci boyunca gözden geçirmeler mutlaka yapılmalıdır.

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Saridoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

YZM211 Yazılım Tasarımı – Yrd. Doç. Dr. Volkan TUNALI

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

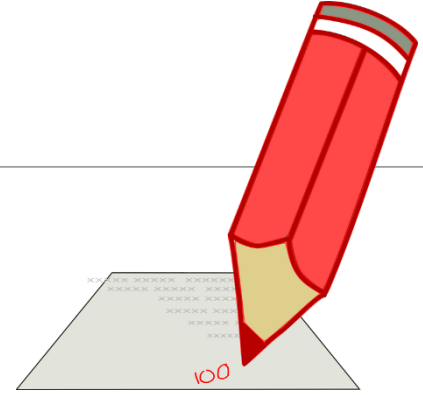
http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

<http://dsdmofagilemethodology.wikidot.com/>

<http://caglarkaya.piquequestion.com/2014/07/01/244/>

Ödev

?



Sorularınız





YMH311-Yazılım Tasarım ve Mimarisi

Mimari Tasarım

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Genel Bir Yazılım Mühendislik Tasarımı Süreci.....7

Mimari Tasarıma Etki Eden Faktörler.....11

Mimari Tasarım Süreci.....12

Mimari Spesifikasyon Notasyonları.....17

Kutu-ve-çizgi diyagramları25

Ortak UML Notasyonları.....28

Sağlanan ve Gereksinilen Arabirimler.....39

Mantıksal ve Fiziksel Mimari.....44

Amaçlar

Mimari tasarım, ürün tasarımı ve ayrıntılı tasarım arasındaki ilişkileri tartışmak

Mimari tasarıma etki eden faktörleri listelemek

Mimari tasarım sürecini gözden geçirmek

Mimari Tasarım Dokümanı'nın (SAD) içeriğini sunmak

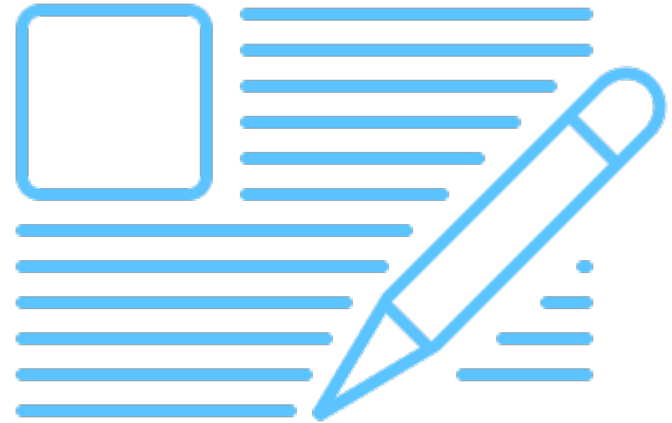
Kalite niteliklerini ve bunların mimari tasarımdaki rolünü açıklamak

Mimari tasarım spesifikasyonu notasyonlarını ve özellikle arabirimler (interface) için olanlarını araştırmak

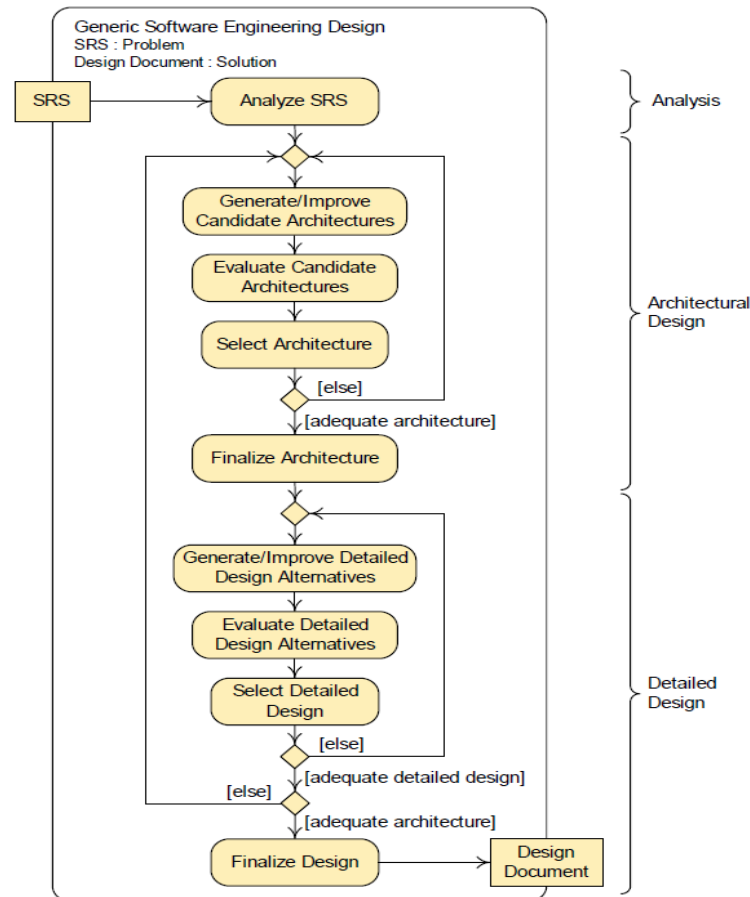


İçerik

- Mimari tasarım, ürün tasarımı, ve ayrıntılı tasarım
- Mimari tasarımı etkileyen faktörler
- Mimari tasarım süreci
- Kalite nitelikleri
- Mimari tasarım spesifikasyonları



Genel Bir Yazılım Mühendislik Tasarımı Süreci



Mimari Tasarım (Architectural Design)



Mimari tasarım, programın büyük parçalarının, bunların sorumluluklarının, özelliklerinin, ve arabirimlerinin; ve bu parçalar arasındaki ilişki ve etkileşimlerin belirlenmesi aktivitesidir.

Ürün Tasarımında Mimari Tasarım

Mimariye ürün tasarımı sırasında da gereksinim duyulur.

- Fizibiliteyi değerlendirmek için
- Paydaşları (stakeholders) gereksinimlerinin karşılanabileceğine ikna etmek için
- Fayda-maliyet analizi yapmak için
- Projeyi planlamak için



Mimari Tasarım ve Ayrıntılı Tasarım

- Mimari tasarım ve ayrıntılı tasarım arasındaki ayrım bazen yeterince belirgin olmayabilir.
- “Büyük” bir program parçası ne demek?
 - Mimari spesifikasyonları ne kadar soyut (abstract) olmalı?
 - Çok küçük bir programın mimarisi nedir/nasıldır?

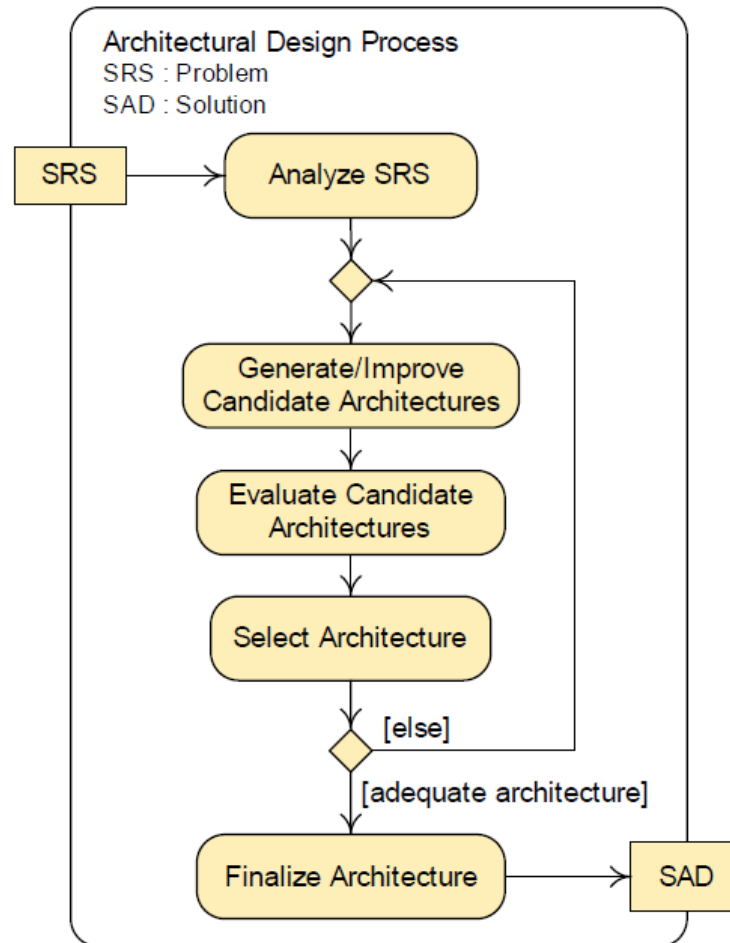


Mimari Tasarıma Etki Eden Faktörler

- Kod kütüphaneleri ve diğer kullanılabilir varlıklar (assets)
- Kurumsal yapı (Organizational structure)
- Tasarımcıların bilgi ve deneyimi
- Mimariler de kişileri ve kurumları etkileyebilir.



Mimari Tasarım Süreci



Mimari Tasarım Dokümanı (SAD)

- **Ürüne Genel Bakış (Product Overview)**—Ürün vizyonu, paydaşlar, hedeflenen pazar, vs.
- **Mimari Modeller (Architectural Models)**—Statik ve dinamik çeşitli modellere ilişkin spesifikasyonlar,
 - DeSCRIPTR
- **Modellerin Nasıl Eşleştiği (Mapping Between Models)**—Modellere ilişkin tablolar ve yazılı bilgiler
- **Mimari Tasarımın Gerekçesi (Architectural Design Rationale)**—Zor, çok önemli, kafa karıştııcı, ve değiştirilmesi zor tasarım kararlarına dair açıklamalar

Kalite Nitelikleri (Quality Attributes)

Bir **kalite niteliği**, müşteri gereksinimlerinin karşılanması bakımından önemli olan işlevlerinden bağımsız olarak, bir yazılım ürününün bir karakteristiği veya özelliğidir.

- İşlevsel olmayan gereksinimler
- Mimarilerin kalite nitelikleri üzerinde büyük etkisi vardır
- Geliştirmeye veya operasyona yönelik nitelikler
- Kalite niteliklerini kullanmak için teknikler (daha sonra)

Geliştirme Nitelikleri (Development Attributes)

- **Bakım Yapılabilirlik (Maintainability)**—Ürünün hatalarının düzeltilebilme, iyileştirilebilme ve başka platforma taşınabilme (port) kolaylığı,

Genellikle altbölümlere ayrılır.

- **Yeniden Kullanılabilirlik (Reusability)**—Ürünün parçalarının başka ürünlerin geliştirilmesinde kullanılabilirliğinin derecesi

- Diğer



Operasyonel Nitelikler (Operational Attributes)

- **Performans (Performance)**—Ürün işlevlerinin zaman ve kaynak limitleri içerisinde sağlanabilme becerisi
- **Uygunluk (Availability)**—Kullanıma hazır bulunma durumu
- **Güvenilirlik (Reliability)**—Normal çalışma koşullarında gereksinimlere uygun davranabilme becerisi
- **Güvenlik (Security)**—Kötü niyetli uygulamalar ve etkiler karşısında zarar görmeye ya da zarar vermeye karşı koyabilme becerisi
- Diğer



Mimari Spesifikasyon Notasyonları

Spesifikasyon Türü	Kullanışlı Notasyonlar
Decomposition	Box-and-line diagrams, class diagrams, package diagrams, component diagrams, deployment diagrams
States	State diagrams
Collaborations	Sequence and communication diagrams, activity diagrams, box-and-line diagrams, use case models
Responsibilities	Text, box-and-line diagrams, class diagrams
Interfaces	Text, class diagrams
Properties	Text
Transitions	State diagrams
Relationships	Box-and-line diagrams, component diagrams, class diagrams, deployment diagrams, text

Arabirimler (Interfaces)

Bir **arabirim** varlıklar arasındaki iletişim sınırırır.

Bir **arabirim spesifikasyonu**, bir varlığın içinde bulunduđu ortam ile haberleşmek için kullandığı mekanizmayı tanımlar.



Arabirim Spesifikasyonları

- Sözdizim (Syntax)—İletişim ortamının elemanları ve bunların mesaj oluşturmak için nasıl kombine edildiği
- Semantik (Semantics)—Mesajların anlamları
- Pragmatik (Pragmatics)—Mesajların ilgili bağlam içerisinde görevleri yerine getirmek için nasıl kullanıldığı

Arabirim spesifikasyonları bir modül ve onun içinde bulunduğu ortamla yaptığı iletişimin sözdizim, semantik, ve pragmatiklerini içermelidir.

Arabirim Spesifikasyonu Şablonu

1. Sağlanan Servisler (Services Provided)

Sağlanan her bir servis için belirtilmelidir:

- a) Sözdizim (Syntax)
- b) Semantik (Semantics)
- c) Pragmatik (Pragmatics)

2. Gereksinilen Servisler (Services Required)

Gereksinilen her bir servis adıyla belirtilmelidir.

Servis açıklaması da eklenebilir.

3. Kullanım Kılavuzu (Usage Guide)

4. Tasarım Gerekçesi (Design Rationale)



Kılavuz

Semantik (Semantic) Spesifikasyonu

- Bir **önkoşul (precondition)** bir aktivite veya operasyonun başlangıcında sağlanması (doğru olması) gereken bir koşuldur.
- Bir **sonkoşul (postcondition)** bir aktivite veya operasyonun bitiminde sağlanması (doğru olması) gereken bir koşuldur.
- Önkoşullar ve sonkoşullar bir operasyon gerçekleştiğinde ne olması gerektiğini yani operasyonun semantiklerini belirtirler.

Mimari Modelleme Notasyonları



Amaçlar

Mimari modellemede kullanılan çeşitli notasyonları göstermek

- Box-and-line diagrams
- UML package diagrams
- UML component diagrams
- UML deployment diagrams

Ortak UML notasyonlarını göstermek

- Notes
- Constraints
- Properties
- Stereotypes



İçerik

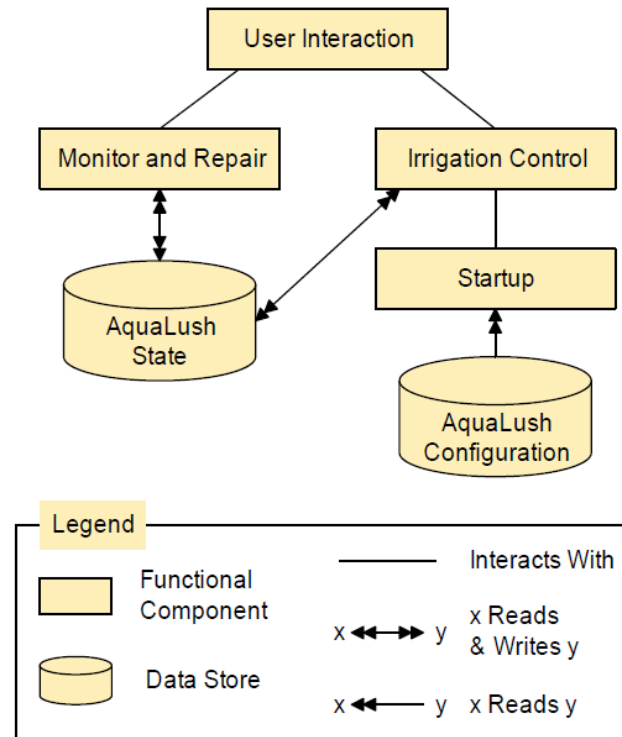
- Kutu-ve-çizgi diyagramları (Box-and-line diagrams)
- Ortak UML notasyonları
- Paketler ve paket diyagramları (Packages and package diagrams)
- Bileşenler ve bileşen diyagramları (Components and component diagrams)
- Nodes, artifacts, and deployment diagrams



Kutu-ve-Çizgi Diyagramları (Box-and-Line)

- Çizgilerle birleştirilen simgeler (kutular, icon'lar) şeklindedir
- Oluşuma ilişkin kurallar yoktur
- Hem statik hem de dinamik modelleme için kullanılırlar
- Lejant (gösterge) eklenmesi iyi olur

Kutu-ve-Çizgi Diyagramı Örneği



Kutu-ve-Çizgi Diyagramı Kuralları

- Kutu-ve-çizgi diyagramlarını yalnızca standart notasyonların yetersiz kaldığı durumlarda kullanın.
- Kutuları ve çizgileri basit ve sade tutun.
- Farklı şeyler için farklı semboller/simgeler kullanın.
- Farklı diyagramlarda sembolleri tutarlı bir şekilde kullanın.
- Elemanları isimlendirirken gramer kurallarına uyun.
- Statik ve dinamik elemanları birlikte kullanmayın.

UML'de Notlar ve Kısıtlar

Not (Note)—Model elemanlarına kesik çizgiyle bağlanan, bir köşesi kıvrılmış bir kutu

- İsteğe bağlı herhangi bir metin içerebilir
- Yorumlar ve belirtilmeler için kullanılır

Kısıt (Constraint)—Model elemanları tarafından belirtilen varlıklar için sağlanması gereken koşulu belirten bir ifade

- Küme parantezi içinde yazılır { }
- Model elemanlarının yanında
- Çeşitli model elemanlarını bağlayan kesik çizgilerin yanında

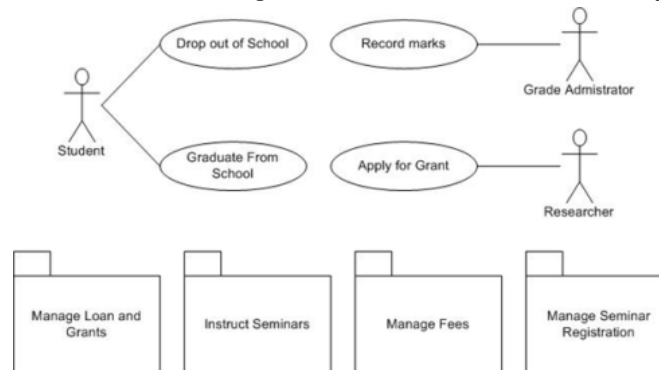
UML'de Özellikler ve Stereotipler

Özellik (Property)—Bir model elemanı tarafından belirtilen bir varlığın karakteristiği

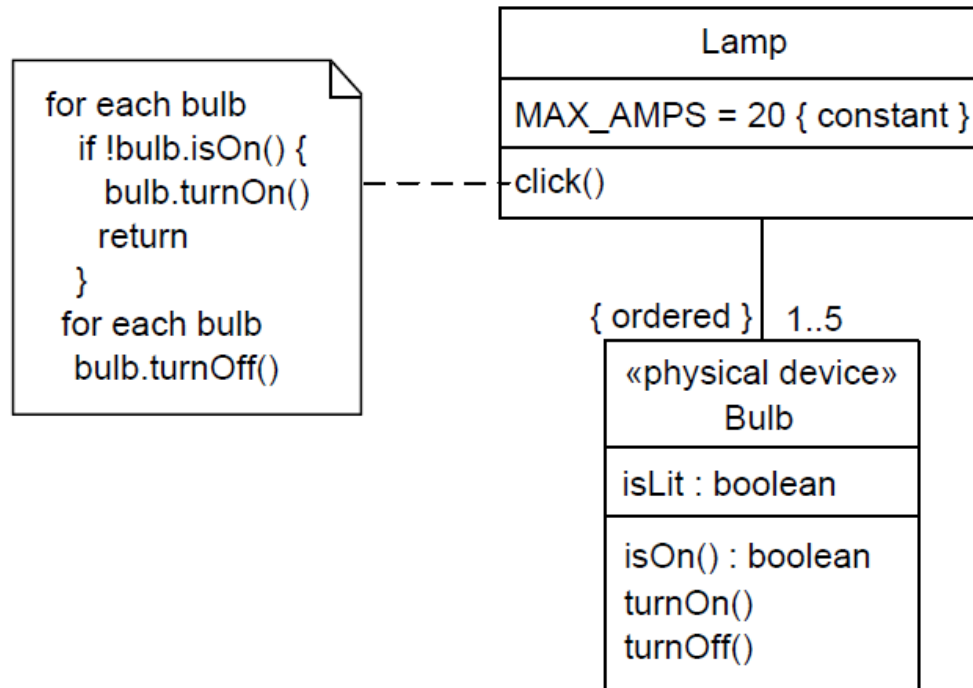
- Küme parantezi içinde etiketli değerlerin listesi
- Etiketli değer: etiket = değer
- True olan Boolean özellikler için değer ve eşittir simgesi yazılmayabilir.

Stereotip (Stereotype)—Daha spesifik anlam verilen bir model elemanı

- Simgeler, renkler, ve grafiklerle gösterilir
- Stereotip anahtar sözcükleri özel çift tırnaklar arasına yazılır, örneğin «interface»



Ortak Elemanlara Örnek

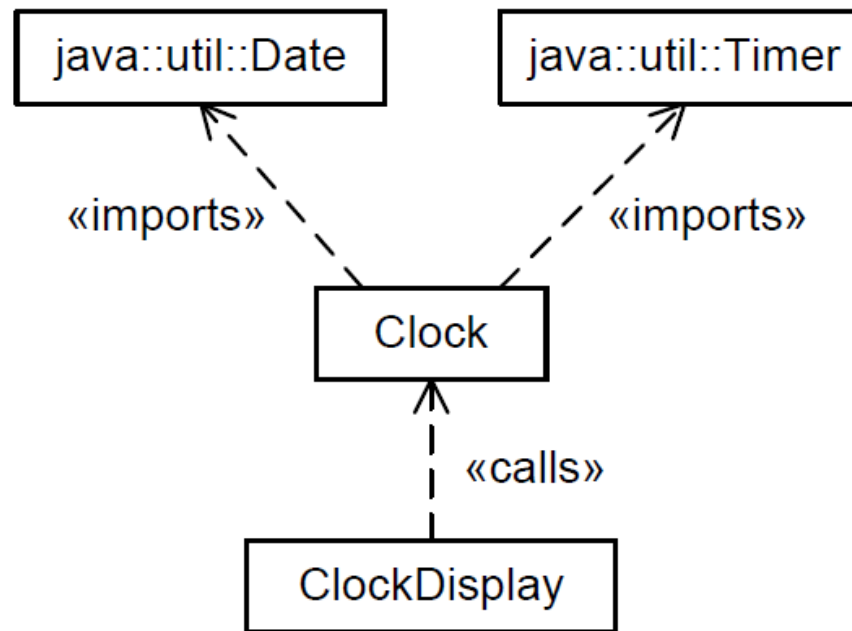


UML Bağımlılık (Dependency) İlişkileri

D ve *I* iki varlık olmak üzere; *I*'da (bağımsız - independent) olan bir değişiklik *D*'yi (bağımlı - dependent) etkiliyorsa, *D* ve *I* arasında bir **bağımlılık ilişkisi** (dependency relation) vardır.

- Örneğin: *D* *I*'yı kullanır, *D* derlenebilmek için *I*'ya bağımlıdır, *D* *I*'yı import eder.
- **Bağımlılık okları**yla (dependency arrow) temsil edilir: stereotipli kesik çizgili oklar.

Bağımlılık İlişkisine Örnek



UML Paketleri (Packages)

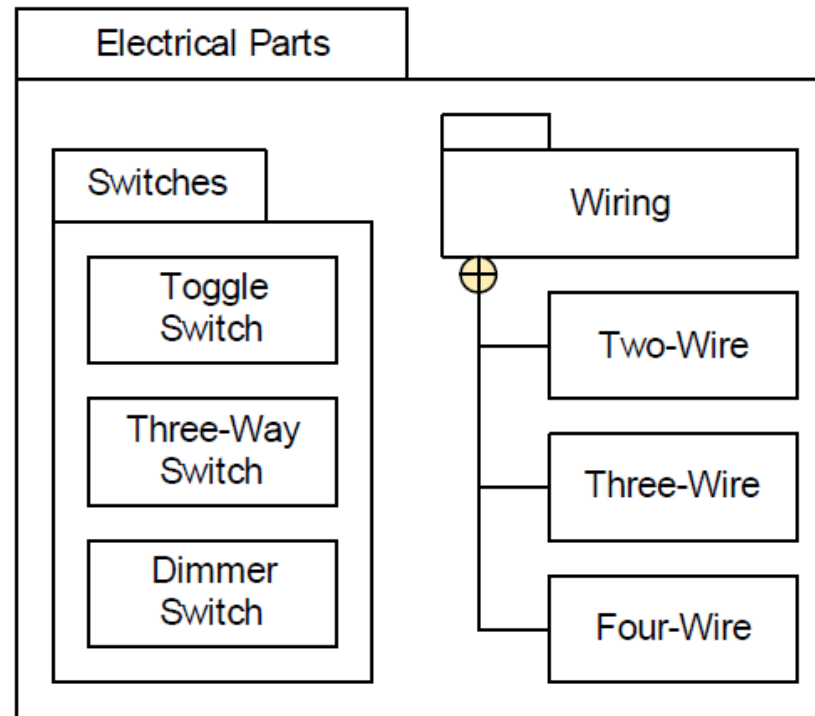
- Bir UML paketi (package), paket üyeleri (package member) olarak adlandırılan model elemanlarının bir koleksiyonudur.
- Paket sembolu olarak dosya klasörü kullanılır.
 - Eğer gövde kısmı doluysa paket adı sekmeye, gövde dolu değilse gövde içinde değilse gövdeye yazılır
 - Üyeler gövde içinde ya da bir kapsama sembolü (çember içinde artı işareti) kullanılarak gösterilir
 - Genellikle import veya export bağımlılık oklarıyla bağlanırlar.

Paket Diyagramları

- Temel olarak paket sembolleri kullanılarak oluşturulan diyagramlara UML paket diyagramı denir.
- Kullanım alanı:
 - Modüllerin, parçalarının, ve onların ilişkilerinin statik modellerinin gösterimi
 - Mimari modelleme

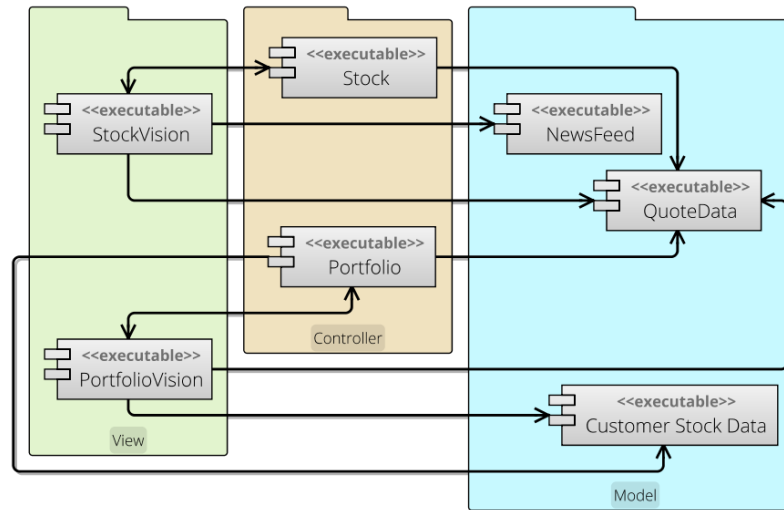


Paket Diyagramı Örneği



Yazılım Bileşenleri (Components)

- Bir **yazılım bileşeni** yeniden kullanılabilir ve değiştirilebilir bir yazılım parçasıdır.
- **Bileşen-tabanlı yazılım geliştirme**, ürünlerin satın alınabilir ya da özel olarak geliştirilmiş yazılım bileşenleri kullanılarak tasarlandığı ve geliştirildiği bir yaklaşımdır.



UML Bileşen Diyagramları

- Bir UML bileşeni, iyi tanımlanmış arabirimlere sahip, modüler, ve başkasıyla değiştirilebilir (replacable) bir birimdir.
 - Bileşen sembolü isim içeren bir dikdörgendir
 - «component» ile stereotipli veya sağ üst köşesinde bileşen sembolü olabilir
- Bir UML bileşen diyagramı bileşenleri, ortamlarıyla ilişkilerini, ve iç yapılarını gösterir.

UML Arabirimleri (Interfaces)

Bir UML **arabirimi** (interface) public tanımlı özellikler ve soyut (abstract) operasyonların adlandırılmış bir koleksiyonudur.

- Stereotipli bir sınıf sembolüyle temsil edilir (daha sonra)
- Özel top ve soket sembolleriyle temsil edilir.

Not: arabirim (interface) sözcüğünün buradaki anlamı daha önce gördüğümüz iletişim sınırı anlamından farklıdır.

Sağlanan ve Gereksinilen Arabirimler

- Bir sınıf veya bileşen bir arabirimin tüm özelliklerini kendine dahil eder ve arabirimin tüm operasyonlarını implemente ederse, bu sınıf veya bileşen bu arabirimi **gerçekliyor/gerçekleştiriyor** (realize) demektir.

Sağlanan (Provided) arabirim—Sınıf veya bileşen tarafından gerçekleştirilir

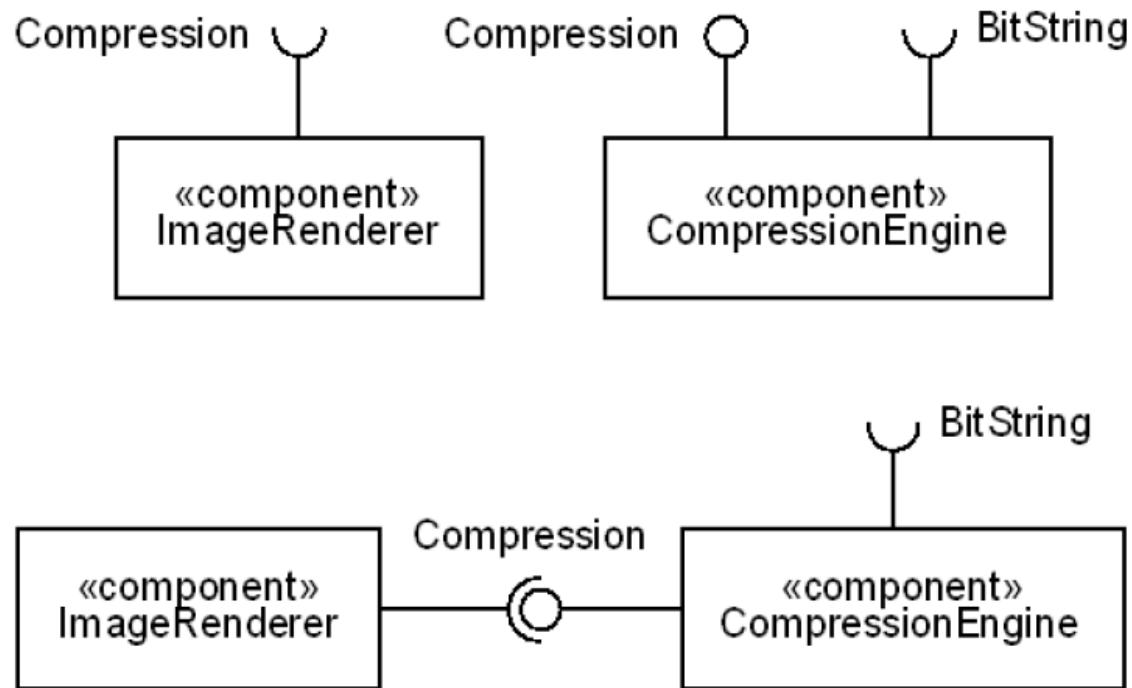
- Bir top veya lolipop sembolüyle temsil edilir

Gereksinilen (Required) arabirim—Bir sınıf veya bileşen tarafından ihtiyaç duyulur

- Bir soket sembolüyle temsil edilir

Montaj konnektörü (assembly connector) arabirimleri birbirine bağlar.

Arabirim Sembollerine Örnekler

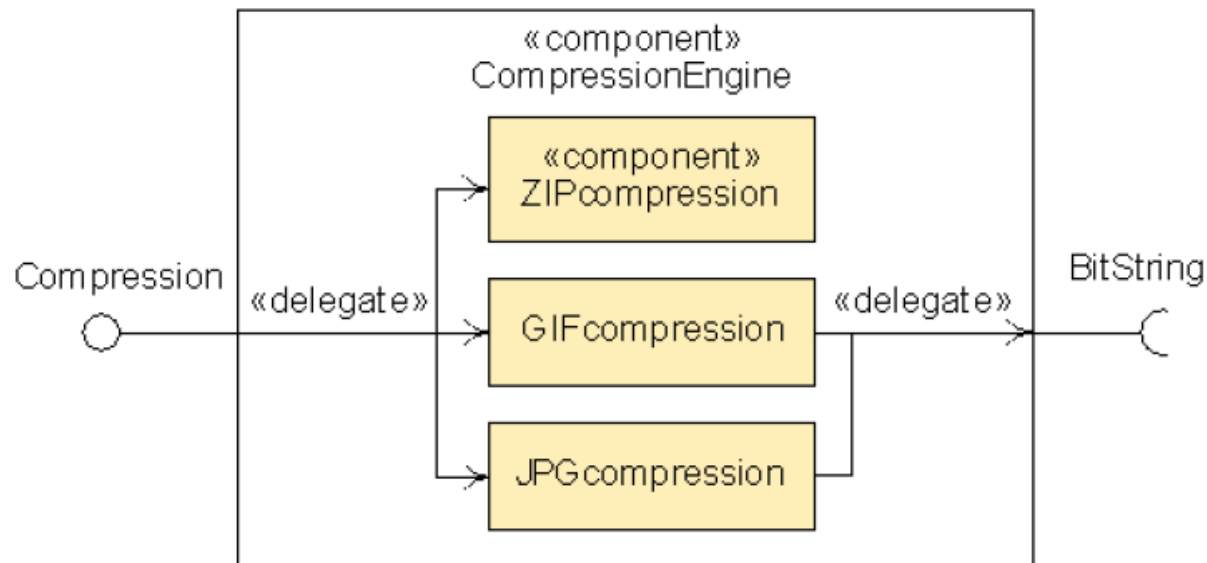


Bileşenlerin İç Yapısı

- Bileşenler başka bileşenleri veya sınıfları içerebilirler ve içsel yapılarını gösterebilirler.
- Bir **delegasyon konnektörü** (delegation connector), bir bileşen arabirimini bu arabirimi gerçekleyen veya kullanan bir veya daha fazla içsel sınıfa veya bileşene bağlar.
 - Düz çizgili oklar
 - «**delegate**» ile stereotipli



Bileşenlerin İç Yapısına Örnek

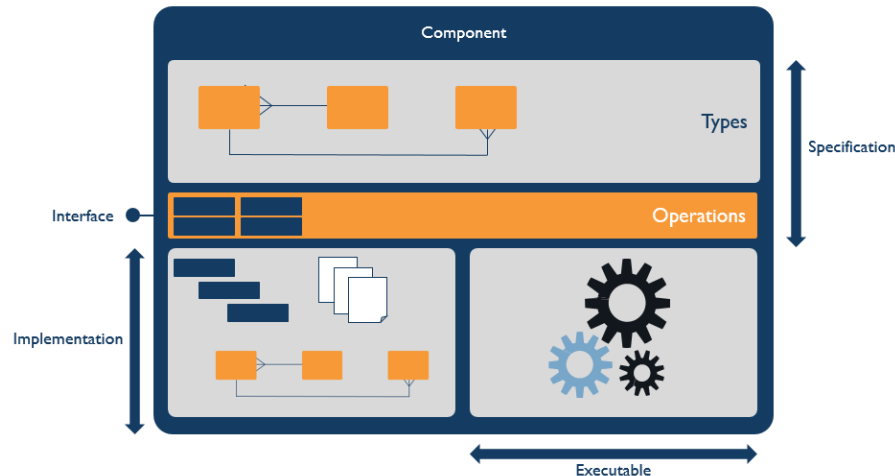


Bileşen Diyagramı Kullanımı

- Yazılım bileşenlerinin statik modellerinin gösteriminde (yeniden kullanılabilir ve başkasıyla değiştirilebilir parçalar)
- Program bileşenlerinin modellenmesinde
 - Mimari modeller
 - Ayrıntılı tasarım modelleri
 - Ortam ile ilişkiler
- Bileşenlerin içsel yapısının modellenmesinde kullanılır.

Mantıksal ve Fiziksel Mimari

- Mantıksal (Logical) mimari—Bir ürünün temel parçalarının ve onların ilişkilerinin çalışan kod halinde gerçek bir makinede implementasyonundan daha soyut bir konfigürasyonu
- Fiziksel (Physical) mimari—Bir ürünün işlemsel kaynaklar üzerinde kod ve veri dosyaları halinde bulunmasının ve çalışmasının gerçekleştirilmesi
- UML kurulum (deployment) diyagramı fiziksel mimariyi modeller.



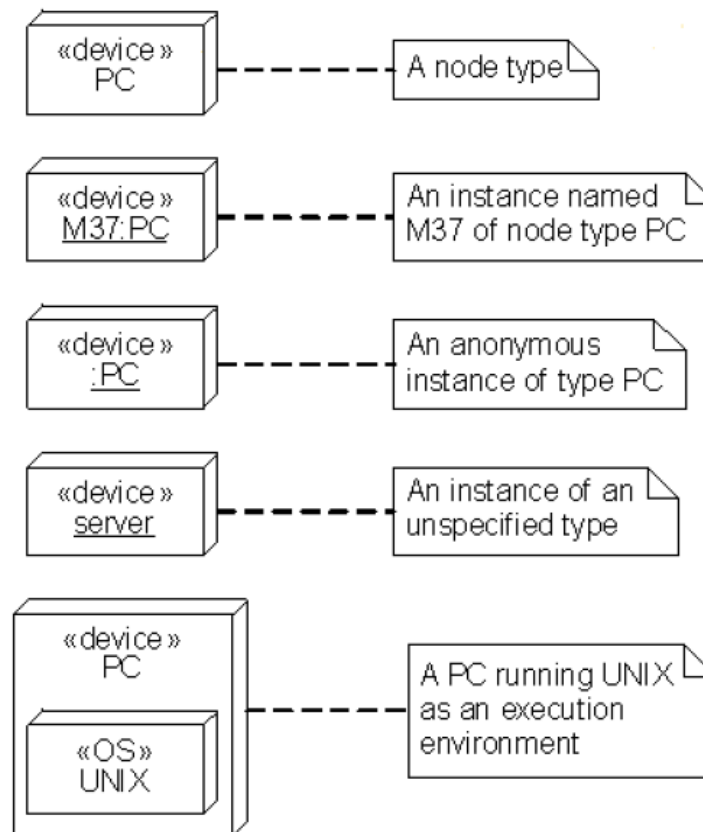
UML Artefaktları (Artifacts)

- Bir UML artefaktı (artifact) geliştirme veya işletim sırasında kullanılan veya üretilen verinin herhangi bir fiziksel temsidir.
 - Örneğin: dosyalar, dokümanlar, program kodları, veritabanı tabloları, vb.
- Artefaktların tipleri ve örnekleri (instance) vardır
 - İsim içeren dikdörtgenle temsil edilir
 - «**artifact**» ile stereotiplidir veya sağ üst köşesinde artefakt sembolü vardır
 - Örneklerin isimlerinin altı çizgilidir, tiplerin isimlerini altı çizgili değildir
- Artefaktlar mantıksal varlıkların gerçekleşmiş şekilleridir (sınıflar, bileşenler, vb.)

UML Düğümleri (Nodes)

- Bir UML **düğümü** (node) işlemsel (computational) bir kaynaktır.
 - **Cihaz (Device)**—Bir bilgisayar gibi fiziksel bir işlem birimidir
 - **İşletim ortamı (Execution environment)**—İşletim sistemi veya bir dil yorumlayıcısı gibi bir sanal makineyi implemente eden bir yazılım sistemidir
- UML’de kutu veya kütük (slab) sembolüyle temsil edilir
 - «**device**» veya «**execution environment**» ile stereotiplidir
 - Tipler ve örnekler (instance)
 - Tiplerin isimleri vardır
 - Örneklerin name : type formunda altı çizgili etiketleri vardır
 - İsim veya tipten biri yazılmayabilir, ama ikisi birden değil

Düğüm Sembolü Örnekleri



Kurulum (Deployment) Diyagramları

➤ Bir UML kurulum diyagramı, işlemsel kaynakları, aralarındaki iletişim yollarını, ve üzerlerinde bulunan ve işletilen artefaktları modeller.

Kullanım yeri:

- Sistemde kullanılan gerçek ve sanal makineleri göstermek
- Makineler arasındaki iletişim yolunu göstermek
- Sistemi oluşturan program ve veri dosyalarını göstermek
 - Yerleşim (Residence)
 - İşletim (Execution)

Kurulum Diyagramı Kuralları

İşlemsel kaynaklar düğümlerdir

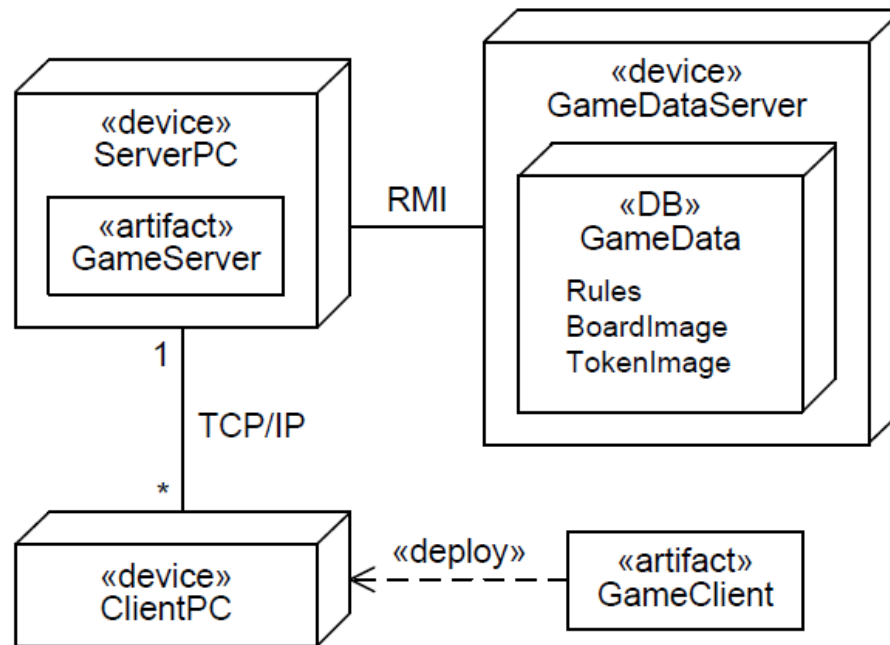
İletişim yolları düğümler arasındaki düz çizgilerdir

- Etiketlenebilir
- Çoklukları ve rol isimleri olabilir

Artefakt sembolleri

- Düğüm sembollerinin içinde görünebilir
- Düğüm sembollerinin içinde listelenebilir
- «**deploy**» stereotipli bağımlılık oklarıyla düğüm sembollerine bağlanabilir

Kurulum Diyagramı Örneği



Özet

- Mimari tasarım ürünün fizibilitesinin değerlendirilebilmesi amacıyla genellikle ürün tasarımı sırasında başlamalıdır.
- Mimari tasarımdan ayrıntılı tasarıma doğru soyutlamanın (abstraction) derecesi azalır.
- Bir mimari tasarım dokümanı (SAD) ürüne genel bir bakış, mimari spesifikasyonlar, ve tasarım gerekçesi gibi bölümlerden oluşur.
- Mimari kararların kalite nitelikleri üzerinde büyük etkisi vardır.
- Mimari modelleme için kullanılan çeşitli notasyonlar mevcuttur.
- Arabirim spesifikasyonları, iletişim ortamına dair tanımlamalar içermelidir
 - Sözdizim (Syntax),
 - Semantik (Semantics), ve
 - Pragmatik.

Özet

- Kutu-ve-çizgi diyagramları statik ve dinamik mimari model oluşturmak için kullanılır.
- Notlar, kısıtlar, özellikler, ve stereotipler herhangi bir UML diyagramında kullanılabilir.
- Paket diyagramları modulleri ve modüllerin parçalarını modellemek için kullanılır.
- Bileşen diyagramları yazılım bileşenlerini modellemek için kullanılır.
- Kurulum diyagramları fiziksel mimarileri modellemek için kullanılır.

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

YZM211 Yazılım Tasarımı – Yrd. Doç. Dr. Volkan TUNALI

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

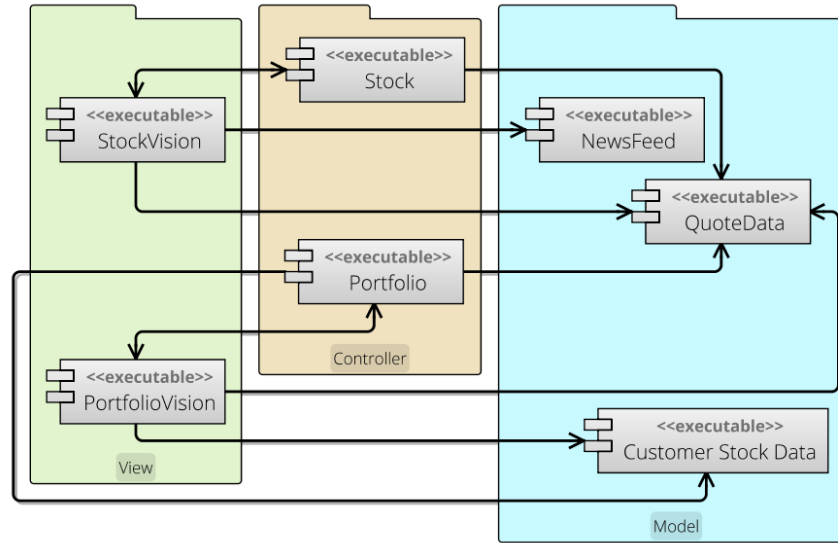
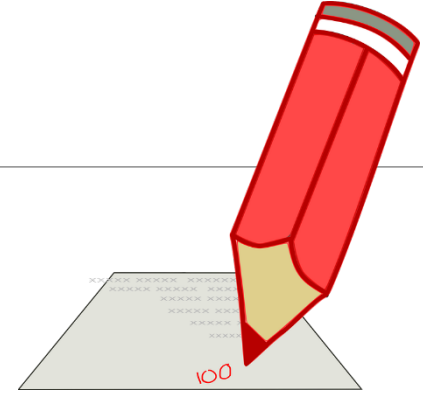
http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

<http://dsdmofagilemethodology.wikidot.com/>

<http://caglarkaya.piququestion.com/2014/07/01/244/>

Ödev

Mimari Tasarım Hakkında Araştırma Yapınız.
Yazılım Mimarisinde Kullanılan Stilleri Araştırınız.



Sorularınız





YMT311-Yazılım Tasarım Ve Mimarisi

Tasarım Süreci: Alt Düzey Tasarım

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

İsimler ve Görünürlük.....	7
Bilgi Saklama ve Erişim.....	16
Operasyon Spesifikasyonu (Op-Spec).....	24
Bildirimsel Spesifikasyonun Avantajları.....	26
Sözleşme ile Tasarım (Design by Contract).....	27
Algoritma Spesifikasyonu.....	36
Tasarımın Tamamlanması.....	43

GENEL BAKIŞ

KISIM 1 – Görünürlük, Erişilebilirlik, ve Bilgi Saklama

KISIM 2 – Operasyon, Algoritma ve Veri Yapısı Spesifikasyonu, ve
Tasarımın Tamamlanması



KISIM 1

Görünürlük, Erişilebilirlik, ve Bilgi Saklama (Visibility, Accessibility, and Information Hiding)



Amaçlar

- Görünürlük kurallarının program varlıklarının isimleri üzerinden nasıl erişilebilir yaptığını açıklamak
- Referans ve takma-adların görünürlüğün ötesinde nasıl erişilebilirlik sağladığını göstermek
- Bilgi saklama açısından görünürlüğün sınırlandırılmasının ve erişimin genişletilmemesinin önemini vurgulamak
- Erişimin genişletilmesine izin verilebilecek durumları açıklamak



İçerik

- İsimler ve görünürlük
- Referanslar ve takma-adlar
- Erişilebilirlik ve bilgi saklama
- Bilgi saklama için kurallar
- Bilgi saklama pratiklerinin ihlali



Varlıklar, İsimler, ve Görünürlük

- Bir **program varlığı** (program entity) genel olarak program içerisinde bir birim olarak ele alınan herhangi bir şeydir.
- Bir isim (name) bir program varlığıyla bağlantılı bir belirteçtir (identifier).
- Bir program varlığına eğer program içinde herhangi bir noktada ismiyle başvurulabiliyorsa, o varlık o noktada **görünür** (visible) demektir.

Görünürlük Örneği

```
File: package1/PublicClass.java
package package1;
public class PublicClass{
    private String privateAttribute;
    String packageAttribute;
    public void method() {
        String localVariable;
        ...
        // point A
        ...
    }
    ...
    // point B
    ...
} // end package1.PublicClass

File: package1/PackageClass.java
package package1;
class PackageClass{
    ...
    // point C
    ...
} // end package1.PackageClass

File: package2/PackageClass.java
package package2;
import package1.*;
class PackageClass{
    ...
    // point D
    ...
} // end package2.PackageClass
```

Görünürlük Türleri

- **Yerel (Local)**—Yalnızca tanımlandığı modül içerisinde görünürdür.
- **Yerel-olmayan (Non-local)**—Tanımlandığı modül dışında da görünürdür, ancak programın her yerinde değil.
- **Global (Global)**—Programın her yerinde görünürdür.
- Eğer bir varlık tanımlandığı modül dışında da görünür durumda ise o varlık tanımlandığı modülden **ihraç edilmiş** (exported) demektir.

Nesne Yönelimli Nitelik Görünürlüğü

Private—Yalnızca tanımlandığı sınıf içerisinde görünürdür

- Yerel görünürlüğün bir türü

Package—Yalnızca tanımlandığı sınıf ve bu sınıfla birlikte aynı paket (veya namespace) içerisinde bulunan sınıfların içerisinde görünürdür

- Yerel-olmayan görünürlüğün bir türü

Protected—Yalnızca tanımlandığı sınıf ve bu sınıftan türeyen alt-sınıflar içerisinde görünürdür

- Yerel-olmayan görünürlüğün bir türü

Public—Sınıfın görünür olduğu her yerde görünürdür

- Yerel-olmayan ve global görünürlüklerin bir türü

Erişilebilirlik (Accessibility)

Bir program varlığı eğer program metni içinde herhangi bir noktada kullanılabiliyorsa o noktada **erişilebilir** (accessible) demektir.

Bir program varlığı görünür olduğu yer yerde aynı zamanda erişilebilirdir.

Bir program varlığı görünür olmadığı yerlerde de erişilebilir olabilir.

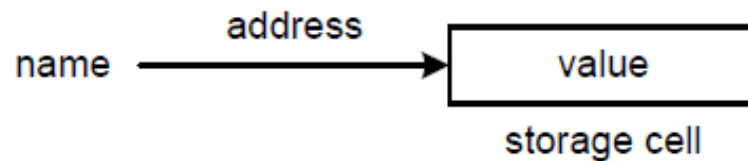


Değişkenler (Variables)

Bir **değişken** (variable) değerleri saklamak için kullanılan bir programlama aygıtıdır.

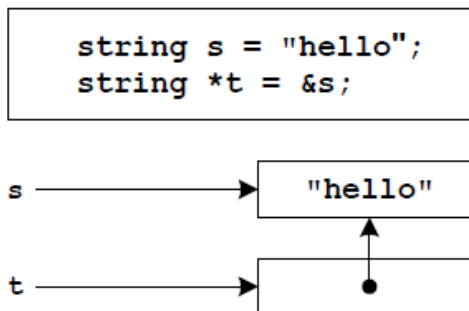
Değişkenlerin nitelikleri:

- İsim (Name)
- Değer (Value)
- Adres (Address)



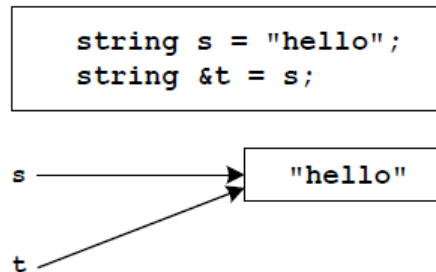
Referanslar (References)

Bir **referans** (reference) bir değerin saklandığı bellek adresini içeren bir ifadedir.



Takma-adlar (Aliases)

Bir **takma-ad** (alias) başka bir değişkenle aynı adrese sahip bir değişkendir.



Görünürlüğün Ötesinde Erişilebilirlik

Referanslar ve takma-adlar değişkenleri görünür olmadıkları yerlerde de erişilebilir yapabilirler

- Parametre olarak bir referans geçirilmesi
- Bir parametrenin referans ile geçirilmesi (takma-ad ile)
- Bir alt-programdan bir referans döndürülmesi

Bu kullanım şekline **erişilebilirliğin görünürlüğün ötesine genişletilmesi** (extending access beyond visibility) denir.

- Genellikle iyi bir programlama pratiği değildir (bad practice)

Bilgi Saklama ve Erişim

Bilgi saklamada anahtar teknik program varlıklarına erişimi mümkün olduğunca kısıtlamaktır.

- **Görünürlüğü kısıtlamak** (Limiting visibility)—Kapsam ve görünürlük işaretleri kullanarak görünürlüğü sınırlandırmak
- Erişimi genişletmemek (Not extending access)—Görünürlüğü genişletmek için referanslar ve takma-adlar kullanmaktan sakınmak

Defansif kopya (defensive copy) tekniği, bir operasyondan normalde görünür olmayan bir nesnenin referansı yerine nesnenin tam bir kopyasının referansının döndürülmesi taktiğidir.

Bilgi Saklama Kuralları 1

Görünürlüğü sınırlandırın.

- Program varlıklarını yalnızca mümkün olan en küçük program bölgesinde görünür yapın.
- Deklarasyonların kapsamını mümkün olan en küçük program bölgesiyle sınırlı tutun.
- **Özellikleri (attribute) en azından protected, tercihan da private yapın.**
- Yardımcı operasyonları en azından protected, tercihan da private yapın
- **Global görünürlükten uzak durun.**
- **Paket görünürlüğünden uzak durun.**

Bilgi Saklama Kuralları 2

Erişimi genişletmeyin.

- Özellikleri sınıfa parametre geçirilen referanslarla ilişkilendirmeyin—bunun yerine defansif kopyalarını oluşturun.
- Özelliklere referans durumunda olan değişkenleri parametre olarak geçirmeyin veya geri döndürmeyin—bunun yerine defansif kopyalarını geçirin veya döndürün.
- Parametreleri referansla geçirmeyin.
- Takma-adlar kullanmayın.

Erişimin Genişletilmesinde İstisnalar

Şu iki durumda erişimi görünürlüğün ötesine genişletmek gerekebilir:

- Modüller işbirliği için bir varlığı paylaşmak zorunda olabilir
 - Örneğin: bir paylaşılan kuyruk (a shared queue)
- Birtakım tasarım hedefleri bilgi saklamadan daha yüksek öneme sahip olabilir
 - Örneğin: performans kısıtları

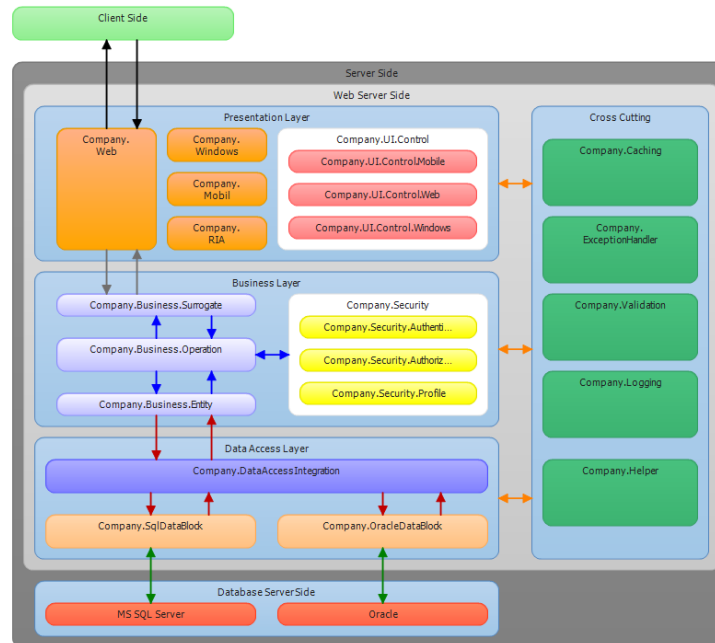


Özet

- Program varlıkları genellikle programın çeşitli bölgelerinde görünür olarak isimleriyle erişilirler.
- Varlıklar ayrıca referanslar ve takma-adlarla da erişilebilirler.
- Bilgi saklama prensibine göre görünürlük kısıtlı olmalı ve erişim görünürlüğün ötesine genişletilmemelidir.
- Bazı özel durumlarda bu prensip ihlal edilebilir.

KISIM 2

Operasyon, Algoritma ve Veri Yapısı Spesifikasyonu, ve Tasarımın Tamamlanması



Amaçlar

Operasyon spesifikasyonlarını ve bunların içeriklerini sunmak

Operasyon davranışlarının bildirimsel spesifikasyonu için sözleşme ile tasarım kavramını sunmak

Algoritma spesifikasyonu için minispecs ve sözde-kodu (pseudocode) tanıtmak

Veri yapısı spesifikasyonu için veri yapısı diyagramlarını tanıtmak

Tasarım sonlandırmayı araştırmak



İçerik

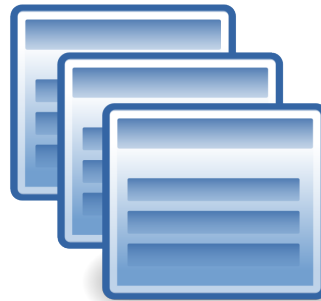
- Operasyon spesifikasyonları
- Bildirimsel ve prosedürel davranış spesifikasyonu
- Sözleşme ile tasarım (Design by contract)
- Assertion, precondition, postcondition, ve class invariant
- Algoritma spesifikasyonu
- Veri yapısı spesifikasyonu
- Tasarımın tamamlanması



Operasyon Spesifikasyonu (Op-Spec)

Operasyonun arabirimini ve sorumluluklarını gösteren yapısal metin

- **Sınıf veya modül** (Class or module)—Operasyonu belirtir
- **İmza** (Signature)—Operasyonun adı, parametrelerin adları ve tipleri, dönüş tipi, ve belki daha fazlası (syntax)
- **Açıklama** (Description)—Bir iki cümle ile
- **Davranış** (Behavior)—Semantikler ve pragmatikler
- **İmplementasyon** (Implementation)—İsteğe bağlı

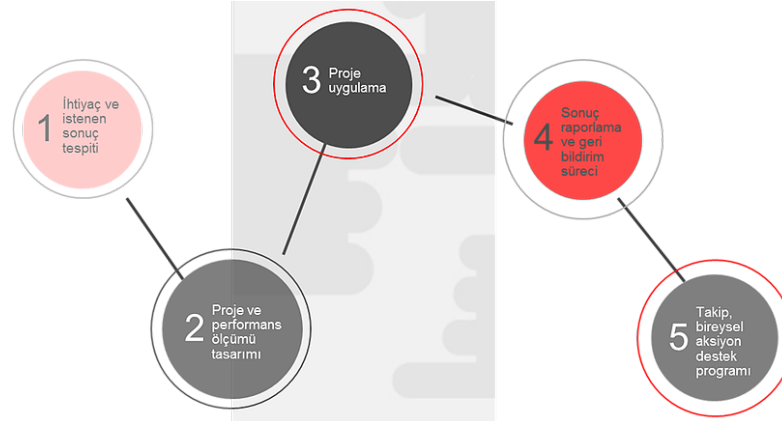


Davranış Spesifikasyonu

Prosedürel (Procedural)—Girdileri çıktılara dönüştüren bir algoritmayı açıklar

- Bir **algoritma** (algorithm) bir bilgisayar tarafından gerçekleştirilebilen işlem adımları sırasıdır.

Bildirimsel (Declarative)—Algoritma belirtmeden girdileri, çıktıları, çağırma kısıtlarını, ve sonuçlarını açıklar



Bildirimsel Spesifikasyonun Avantajları

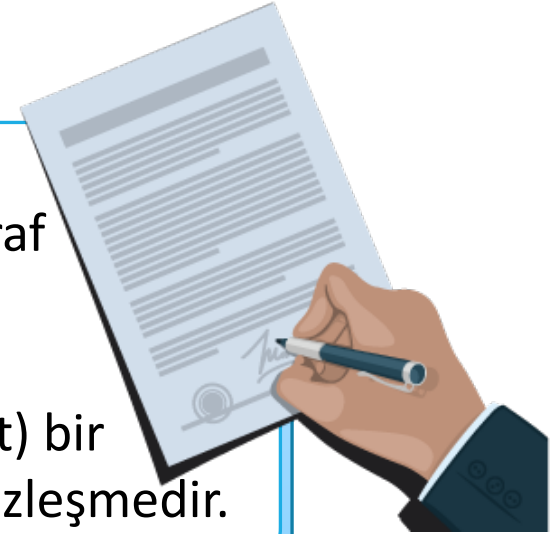
- Daha soyuttur çünkü implementasyon ayrıntılarını gözardı eder—daha öz/özet
- Arabirime odakların, içsel işleyişe değil
- Prosedürel spesifikasyonun yaptığı gibi programcıyı belirli bir implementasyonun uygulanması için yönlendirmez, zorlamaz.



Sözleşme ile Tasarım (Design by Contract)

Bir **sözleşme** (contract) iki veya daha fazla taraf arasında bağlayıcı bir anlaşmadır.

Bir **operasyon sözleşmesi** (operation contract) bir operasyon ve onu çağıranlar arasındaki bir sözleşmedir.



Sözleşme Hakları ve Yükümlülükleri

➤ Çağırın

- Uygun şartlar altında geçerli (valid) parametreler geçirmekle yükümlüdür.
- Duyurulan işlemsel hizmetleri alma hakkına sahiptir.

➤ Çağrılan operasyon

- Duyurulan hizmetleri sağlamakla yükümlüdür.
- Uygun koşullar altında geçerli parametrelerle çağrılma hakkına sahiptir.



Bildirimler (Assertions)



Bir **assertion** bir program içinde belirlenen noktada doğru (true) olması gereken bir ifadedir

- Bu bildirimler çağırان ve çağrılan operasyonun hakları ve yükümlölüklerini belirtir.

Önkoşul ve Sonkoşullar

Bir **önkoşul** (precondition) bir operasyonun en başında doğru olması gereken bir assertion'dır.

Bir **sonkoşul** (postcondition) bir operasyon tamamlandığında doğru olması gereken bir assertion'dır.

- Önkoşullar çağıranın yükümlülüklerini ve çağrılan operasyonun haklarını belirtirler.
- Sonkoşullar ise çağıranın haklarını ve çağrılan operasyonun yükümlülüklerini belirtirler.

Operasyon Spesifikasyonu Örneği

Signature	<code>public static int findMax(int[] a) throws IllegalArgumentException</code>
Class	Utility
Description	Return one of the largest elements in an int array.
Behavior	pre: (a != null) && (0 < a.length) post: for every element x of a, x <= result post: throws IllegalArgumentException if preconditions are violated

Sınıf Değişmezleri (Class Invariants)

Bir **önkoşul** (precondition) bir operasyonun en başında doğru olması gereken bir assertion'dır.

Bir **sonkoşul** (postcondition) bir operasyon tamamlandığında doğru olması gereken bir assertion'dır.

- Sınıf değişmezleri bütün ihraç edilen operasyonların sözleşmelerini tamamlayıcı niteliktedir.

Bildirimlerde (Assertion) Neler Olmalı 1

Önkoşullar (Preconditions):

- Parametreler üzerindeki kısıtlamalar
- Operasyon çağrılmadan önce sağlanması gereken koşullar

Sonkoşullar (Postconditions):

- Parametreler ve sonuçlar arasındaki ilişkiler
- Sonuçlar üzerindeki kısıtlamalar
- Parametrelerdeki değişiklikler
- İhlal edilen önkoşullara karşı verilecek karşılıklar

Bildirimlerde (Assertion) Neler Olmalı 2

Sınıf değişmezleri (Class invariants)

- Özellikler (attribute) üzerindeki kısıtlamalar
- Özellikler (attribute) arasındaki ilişkiler

Boş bildirimleri “true” veya “none” olarak ifade edin.

Op-Spec'lerin Oluşturulması

Orta-düzey tasarımın ilk evrelerinde ayrıntılı op-spec'ler oluşturmayın

- Çünkü tasarım henüz çok değişken; ayrıntılar çok fazla değişikliğe uğrayabilir

Tasarımın sonunu beklemeyin

- Ayrıntılar unutulabilir
- Muhtemelen çok yetersiz kalır

Op-spec'leri tasarım sırasında kademeli olarak oluşturun, kesinleştikçe de ayrıntıları ekleyin

Algoritma Spesifikasyonu

- İyi-bilinen algoritmaları yalnızca adıyla belirtin.
- **Minispec** kullanarak bir algoritmanın, girdileri çıktılara nasıl dönüştürdüğünü adım adım anlatın.
- Minispec'leri **sözde kod** (pseudocode) ile yazın.

Pseudocode Örneği

```
Inputs: array a, lower bound lb, upper bound ub,  
        search key  
Outputs: location of key, or -1 if key is not  
        found  
lo = lb  
hi = ub  
while lo <= hi and key not found  
    mid = (lo + hi) / 2  
    if      ( key = a[mid] ) then key is found  
    else if ( key < a[mid] ) then hi = mid-1  
    else                                     lo = mid+1  
if key is found then return mid  
else                 return -1
```

Veri Yapıları

Bir **önkoşul** (precondition) bir operasyonun en başında doğru olması gereken bir assertion'dır.

Bir **sonkoşul** (postcondition) bir operasyon tamamlandığında doğru olması gereken bir assertion'dır.

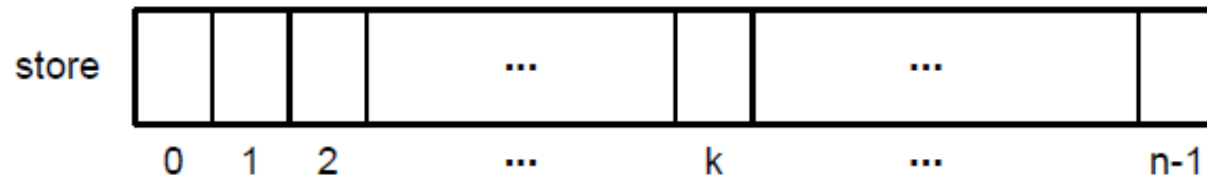
Ardışık (Contiguous) implementasyon—Değerler birbirine bitişik bellek hücrelerinde saklanır

Bağlı (Linked) implementasyon—Değerler referanslar veya işaretçilerle erişilebilen rastgele bellek hücrelerinde saklanır

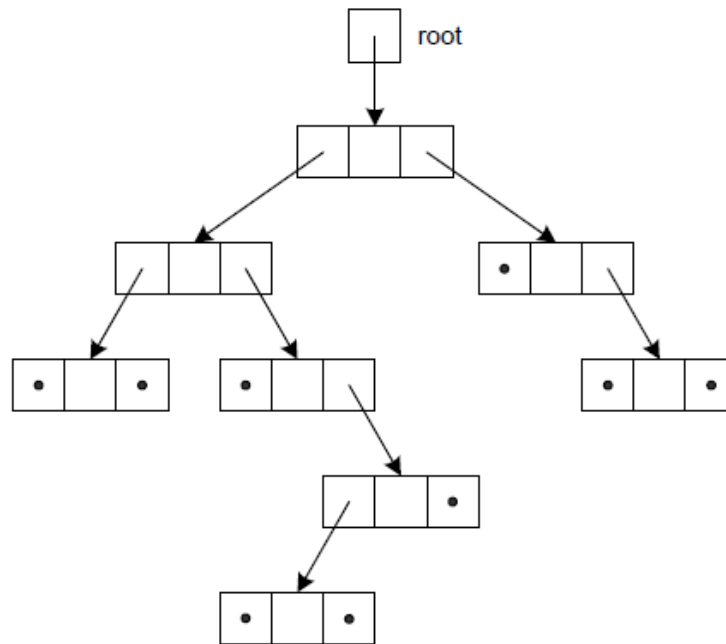
Veri Yapısı Diyagramı

- Dikdörtgenler bellek hücrelerini temsil eder. Genellikle isimlendirilirler.
- Bitişik hücreler bitişik dikdörtgenlerle temsil edilir. Hücrelerin indisleri olabilir.
- Tekrar eden elemanlar ... (ellipsis) ile gösterilir.
- Bağlı hücreler işaretçileri veya referansları temsil eden, bir hücreden diğerine giden oklarla gösterilir.
- Null işaretçiyi göstermek için nokta kullanılır.

Veri Yapısı Diyagramı Örneği 1

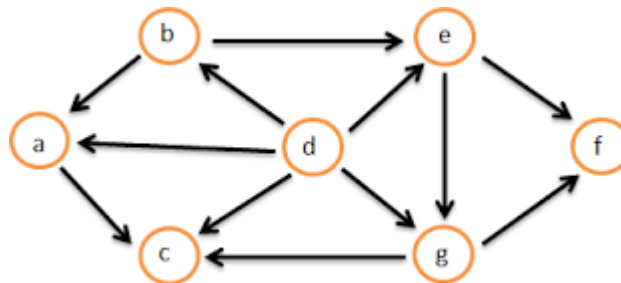


Veri Yapısı Diyagramı Örneği 2



Veri Yapısı Diyagramı Kuralları

- Record alanlarını yalnızca bir kez etiketleyin.
- Büyük ve tekrar eden yapıları ellipsis (...) ile basitleştirin.
- Bağlı yapıları çizerken işaretçilerin okları yukarıdan aşağıya veya soldan sağa doğru ilerlesin.
- Az bilinen veya ilave semboller için lejant (gösterge) kullanın.



Tasarımın Tamamlanması

- Alt-düzey spesifikasyonlarla birlikte tasarım dokümanı tamamlanır.
- Tasarımın tamamlanması (Design finalization) aşamasında tasarımın kalitesi ve iyi dokümante edildiğinden emin olmak için kontroller yapılır.
- Mühendislik tasarımı sürecinin son adımıdır.



Tasarım Dokümanı Kalite Karakteristikleri 1

Fizibilite (Feasibility)—Tasarım gerçekleştirilebilir, yapılabilir olmalıdır

Yeterlilik (Adequacy)—Tasarım gereksinimleri karşılayacak bir programı belirtmelidir

Ekonomi (Economy)—Tasarım zamanında ve bütçe sınırları içinde yapılabilecek bir programı belirtmelidir

Değişebilirlik (Changeability)—Tasarım kolayca değiştirilebilecek bir programı belirtmelidir.

Tasarım Dokümanı Kalite Karakteristikleri 2

İyi yapılanmışlık (Well-Formedness)—Tasarımda notasyonlar doğru kullanılmalıdır

Tamlık (Completeness)—Programcıların programı yazabilmeleri için gereken herşeyi belirtmelidir

Açıklık/Netlik (Clarity)—Mümkün olduğunca anlaşılması kolay olmalıdır

Tutarlılık (Consistency)—Tek bir ürün tarafından gerçekleştirilebilecek bir tasarım içermelidir.

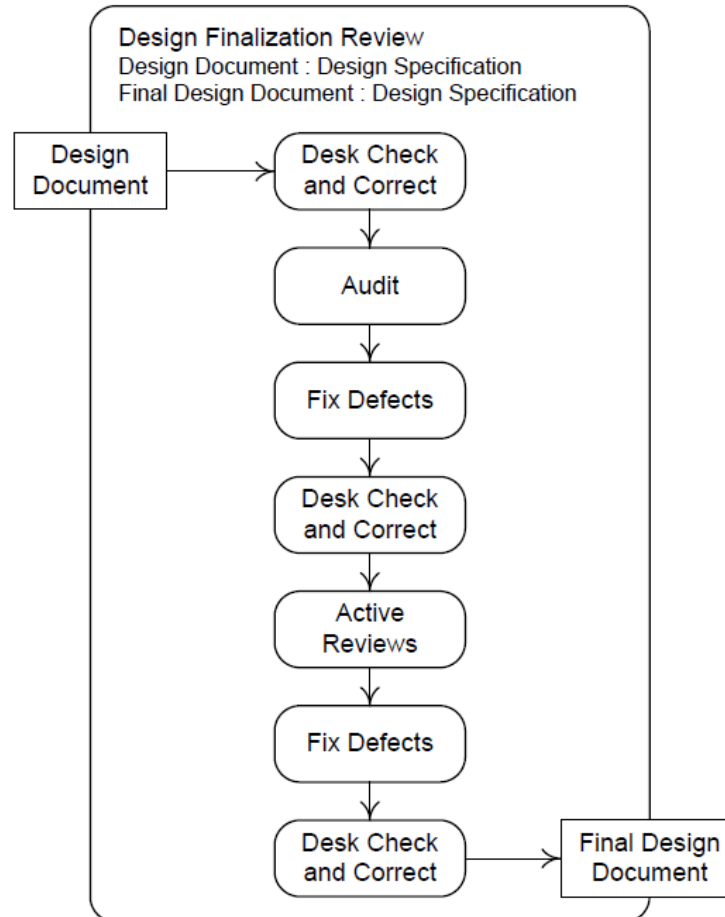
Eleştirel Gözden Geçirme (Critical Review)

Bir **eleştirel gözden geçirme** (critical review) tamamlanmış bir ürünün kabul edilebilir bir kalitede olup olmadığına dair değerlendirilmesidir.

Gözden geçirmeler şu şekillerde olabilir:

- Desk checks,
- Walkthroughs,
- Inspections,
- Audits, and
- Active reviews.

Eleştirel Gözden Geçirme Süreci



Sürekli (Continuous) Gözden Geçirme

- Eleştirel gözden geçirme sayesinde tasarımın erken safhalarına dönülmesini gerektirecek çok ciddi tasarım kusurları bulunabilir. Bu da;
 - Pahalıdır
 - Zaman kaybettiricidir
 - Sinir bozucudur
- Sürekli gözden geçirme (continuous review) politikası sayesinde tasarımın her aşamasında kusurlar daha erken fark edilebilir ve geç fark edildiğinde yaşanabilecek sıkıntıların önüne geçilebilir.

Özet

- Operasyon spesifikasyonları operasyonlarla ilgili tasarım ayrıntılarını belirtir
 - Sınıf veya modül adı
 - İmzası
 - Açıklaması
 - Davranışı
 - İmplementasyonu
- Davranışlar bildirimsel veya prosedürel olarak belirtilebilir.
- Bildirimsel spesifikasyon assertion'lar ile belirtilen operasyon sözleşmeleriyle yapılır.
 - Önkoşullar
 - Sonkoşullar

Özet

- Algoritmalar minispec veya pseudocode ile belirtililebilirler.
- Veri yapıları veri yapısı diyagramı kullanılarak gösterilirler.
- Tasarımın tamamlanması, mühendislik tasarımının son adımıdır.
- Tasarım dokümanı kalite karakteristiklerini karşılaması bakımından bir eleştirel gözden geçirme süreci ile kontrol edilir.
- Eleştirel gözden geçirmelerin kurum politikası olarak sürekli gözden geçirme şeklinde uygulanması geç fark edildiğinde felakete yol açabilecek kusurların erken safhalarda fark edilmesini sağlar.

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

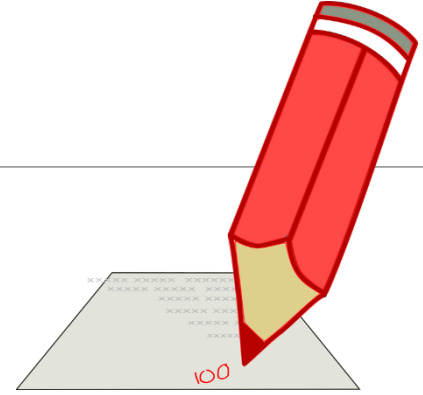
http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

<http://dsdmofagilemethodology.wikidot.com/>

<http://caglarkaya.piququestion.com/2014/07/01/244/>

Ödev

?



Sorularınız





YMH 311-Yazılım Tasarım ve Mimarisi

TASARIM Süreci

Nesneye Yönelik Çözümleme ve Tasarım

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Nesneye Yönelik Kavramlar.....	5
Nesne Yönelimli Sistemleri Destekleyen Kavramlar.....	10
Nesneye Yönelik Çözümlemenin Temelleri.....	15
Gereksinim Belirleme Çalışmaları.....	18
Nesneye Yönelik Çözümleme Teknikleri.....	24
Alt Sistem Modellemesi.....	28
Nesneye Yönelik Tasarım.....	39
Nesneye Yönelik Tasarım Metodolojileri.....	47
NY Metodolojilerinin Tasarım Yöntemleri.....	49
Genel Sistem Tasarımı.....	57

Amaçlar



Nesneye Yönelik Kavramlar Hakkında Bilgilenmek

Kalıtım ve Çoklu Kalıtım Kavramlarını Öğrenmek

Nesne Yönelimli Sistemleri Destekleyen Kavramları Bilmek

Gereksinim Belirleme Çalışmalarını Öğrenmek

Nesneye Yönelik Çözümleme Tekniklerini Öğrenmek

Alt Sistem Modellemesini Kavramak

Nesneye Yönelik Tasarım Katmanlarını Öğrenmek

Nesneye Yönelik Metodolojilerinin Tasarım Yöntemlerini Öğrenmek

Genel Sistem Tasarımı Yapmak

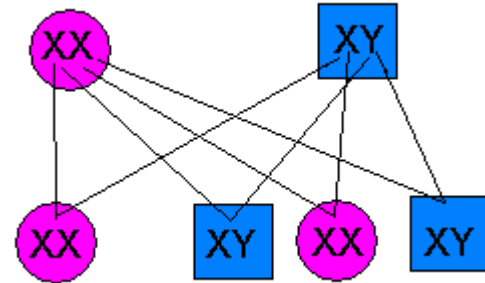
Giriş

- Yapısal tekniklerle güçlenen yazılım mühendisliğine, programlama düzeyinden yukarıya doğru bir gelişme ile Nesneye Yönelik (NY) yaklaşımlar kazandırıldı.
- NY programlama dillerinin gördüğü büyük ilgi, çok geçmeden bu yaklaşımın tasarım ve çözümleme konularına da yansması ve daha sonra tüm süreç modeli ve metodoloji uyarlamaları ile izlendi.
- Bugün yeni projeler için geleneksel/NY tartışması, NY lehinde kararlarla sonuçlanma yönünde değişmektedir.
- Geçerli neden olmadığı durumlarda artık NY ortamlarda geliştirme çalışması yapmak gerekmektedir.
- Bu bölümde önce NY kavramlar kısaca özetlenecek ve sonra çözümleme teknikleri sunulacaktır.
- Geleneksel yaklaşımların geldiği noktaya, NY yaklaşımlar çabucak gelerek, kabul görmüş metodolojiler yaygınca bilinen standartlar halinde gelişmiştir.
- Hatta bu yaklaşımlardaki benzerlikler ve piyasanın da motivasyonu sonucunda değişik yaklaşımlar arasında uyum çalışmaları yapılmıştır.
- Fusion ve UML, bu birleştirme/standartlaşma çabalarının başlıca iki örneğini oluştururlar.

Nesneye Yönelik Kavramlar

Doğal çevremizdeki nesnelerin özellikleri ve ilgili oldukları süreçler bir arada düşünülür. Bir nesnenin özellikleri olduğu gibi onun yapacağı işlemler de vardır. NY modellerin ilk özelliği olarak, veri ve süreçlerin bir nesne için bir arada değerlendirilmesi tanımlanır. NY kavramların temeli olarak, aşağıdaki dört özelliği sıralayabiliriz:

1. Kimlik (Identity)
2. Sınıflama (Classification)
3. Kalıtım (inheritance)
4. Çok Şekillilik/Biçimlilik(Polymorphism)



Kimlik

Kimlik

Kimlik: Nesne denilen, farklı, ayırt edilebilen varlıkların tanımlanmasıdır. Her nesnenin kendi kimliği vardır. Örneğin; Ali'nin bisikleti, renkli TV.

Sınıf

Kalıtım

Çok şekillilik

<i>Poligon Nesneleri</i>	<i>Poligon Sınıfı</i> <i>Özellikler (Attributes)</i>	<i>İşlemler (Operations)</i>
üçgen	gerçeve rengi	çiz
dikdörtgen	dolgu rengi	sil
yamuk	kenar sayısı	taşı

Sınıf

Kimlik

Sınıf

Kalıtım

Çok şekillilik

Sınıf: Aynı veri yapılı (aynı özellikli) ve aynı davranışlı (işlemli, operations) nesneler bir sınıfta gruplandırılır. Bir sınıftaki her bir nesneye o sınıfın örneği (instance) denir.

Bir sınıfın her bir örneği, o sınıfın her bir özelliği için kendi değerine sahiptir, fakat aynı özellik isimlerini ve işlevleri paylaşırlar.

Kalıtım

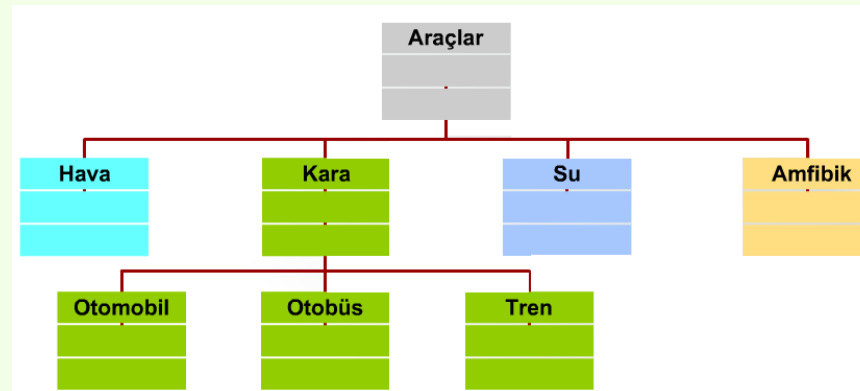
Kimlik

Sınıf

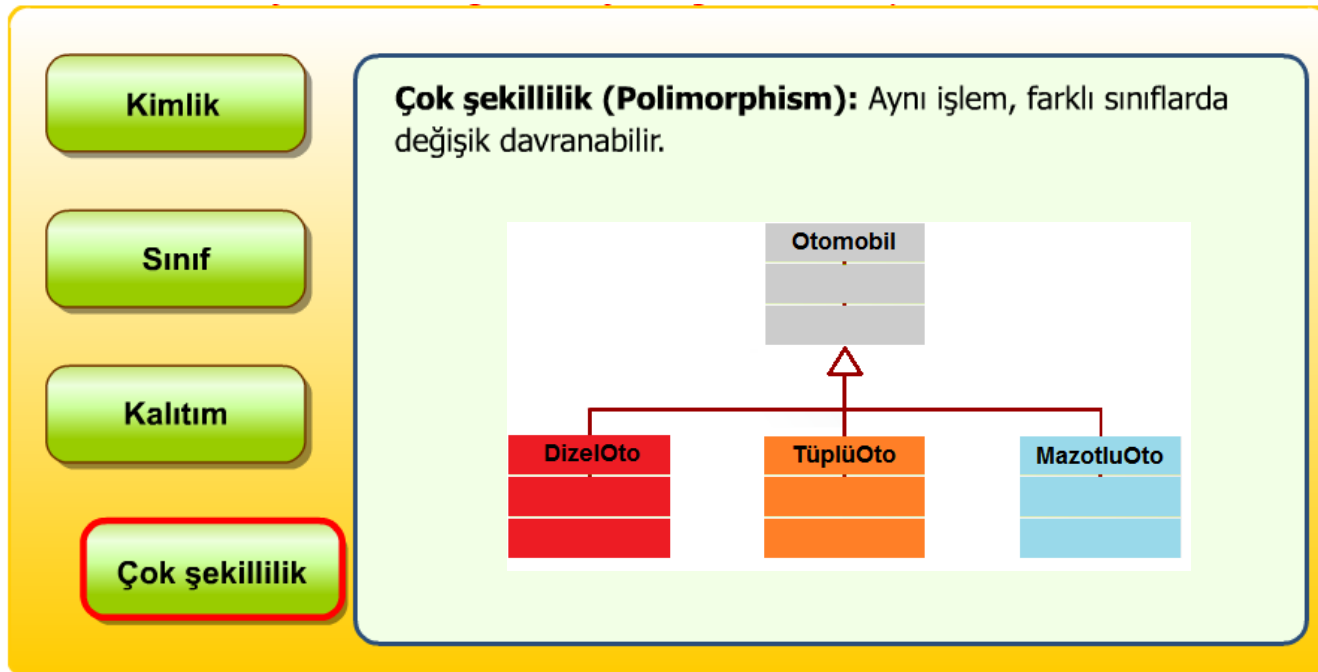
Kalıtım

Çok şekillilik

Kalıtım (Inheritance): Hiyerarşik ilişkiye dayanan sınıflar arasında özellik ve işlevlerin paylaşılmasıdır.



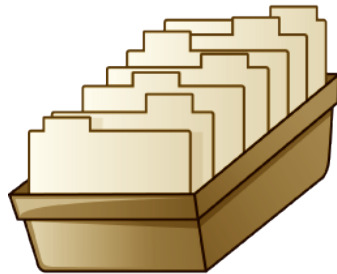
Çok Şekillilik



Nesne Yönelimli Sistemleri Destekleyen Kavramlar

Sadece nesne yönelimli sistemlerde olmayan ama nesne yönelimli sistemleri destekleyen kavramlar;

- Soyutlama (Abstraction),
- Bilgi Gizleme (Encapsulation),
- Paylaşım (Sharing),
- Altsınıf (Subclass) ve "Aggregation" dır.



Soyutlama

Soyutlama

Bilgi Gizleme

Paylaşım

Altsınıf

Soyutlama (Abstraction):

Soyutlama kullanımı, analiz süresince sadece uygulama alanı kavramları ile ilgilenmek ve problemi anlamadan önce tasarım ve uygulama kararları vermemektir.

Bilgi Gizleme

Soyutlama

Bilgi Gizleme

Paylaşım

Altsınıf

Bilgi Gizleme (Encapsulation):

Bir nesnenin iç ayrıntılarının ve ilişkilerinin, eriştiği diğer nesnelerden gizlenmesidir. Diğer tasarım yöntemleri ve programlama dillerinde de bu özellik vardır, ancak burada veri yapısı ve davranış bir arada olabildiği için daha güçlüdür.

Paylaşım

Soyutlama

Bilgi Gizleme

Paylaşım

Altsınıf

Paylaşım (Sharing):

Nesne yönelimli dillerde kodun paylaşımı sağlanabilir, ayrıca tasarım ve kodlama, gelecek projelerde de kullanılabilir.

Altsınıf

Soyutlama

Bilgi Gizleme

Paylaşım

Altsınıf

Altsınıf (Subclass) ve "Aggregation":

"Aggregation": İki sınıf arasındaki ilişkide bir sınıfın örneklerinin, diğer sınıfın örneklerinin parçaları (üyeleri) ve içeriği olmasıdır.

Altsınıf (Subclass): Bir sınıf (superclass veya parentclass), diğer sınıftan seçilen örneklerden yapılır.

Nesneye Yönelik Çözümlemenin Temelleri

Çözümleme: Bir şeyi anlayabilmek için parçalarına ayırmak.

Sistemi anlamaya yönelik çalışmalardan ve üst düzey planlama eylemlerinden oluşur.

- Uygulama/problem alanının anlaşılması.
- Kullanıcı gereksinimlerinin anlaşılması.
- Koddaki sınıflar ve nesneler ile bunların arasındaki üst düzey etkileşimlerin belirlenmesi: Çözümleme modelinin oluşturulması.

“Bir sorunu anlamadan çözemezsiniz.”



Uygulama Alanının Çözömlenmesi (Domain Analysis)

- Amaç, uygulama alanını anlamak ve elde edilen bilgileri analiz modeline taşımaktır.
 - Uygulama alanı hakkında bilgi edinilebilecek kaynaklar:
 - Teknik literatür
 - Mevcut uygulamalar
 - Müşteri anketleri
 - Uzman tavsiyeleri
 - Mevcut ve gelecekteki gereksinimler
- Problem alanı hakkında bilgi edinmeden “müşterinin dilinden konuşamazsınız”.

Genel Olarak NY Metodolojiler

- Sözü geçen değişik modeller, temelde genellenebilecek ortaklıklar göstermektedir.
- Bu özellikten faydalanmak için önce Fusion ve sonra da UML metodolojileri ortaya çıkmıştır.
- İki girişimde de önceki yöntemlerin iyi taraflarının alınarak, standart ve daha iyi bir metodolojiye ulaşmak isteği yer almıştır.
- Genellenmiş şekilde işlemleri aşağıdaki gibi özetleyebiliriz:
 1. Gereksinimler belirlenir (kullanım durumları ve senaryolar kullanılır).
 2. Sınıf ve nesneler belirlenir.
 3. Özellikler ve işlemler tanımlanır.
 4. Sınıfları ve nesneleri organize edecek hiyerarşiler ve yapılar belirlenir.
 5. Nesneler arası ilişkiler modellenir.
 6. Nesneler arası etkileşim modellenir.
 7. Ortaya çıkan modeller, kullanım örnekleri ve senaryolar ile denenir ve iyileştirilir.

Gereksinim Belirleme Çalışmaları

- Bu bölümde sözü geçecek olan iki teknik;
 - Kullanım Durumları ve
 - Sınıf Sorumluluk İşbirlikçi modelleridir.
- Kullanım Durumları, NY olmayan ortamlara da uyarlanabilir.
- Ancak bu tekniğin ortaya çıkışı NY metodolojilerle birlikte olmuştur.

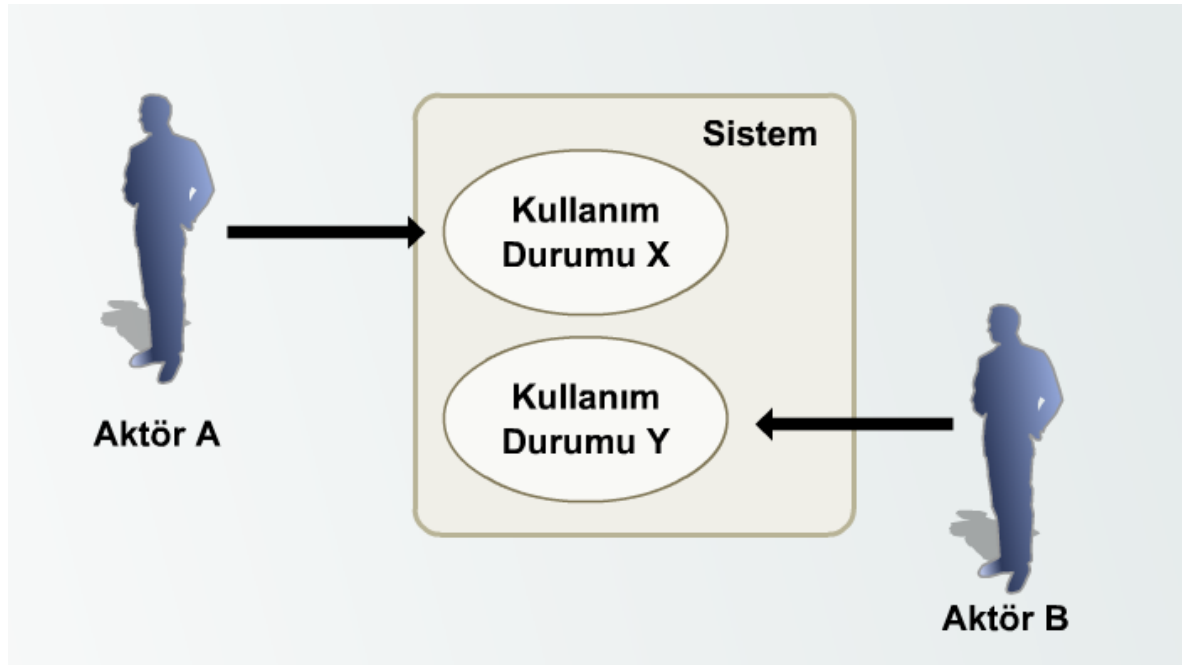


Kullanım Durumları

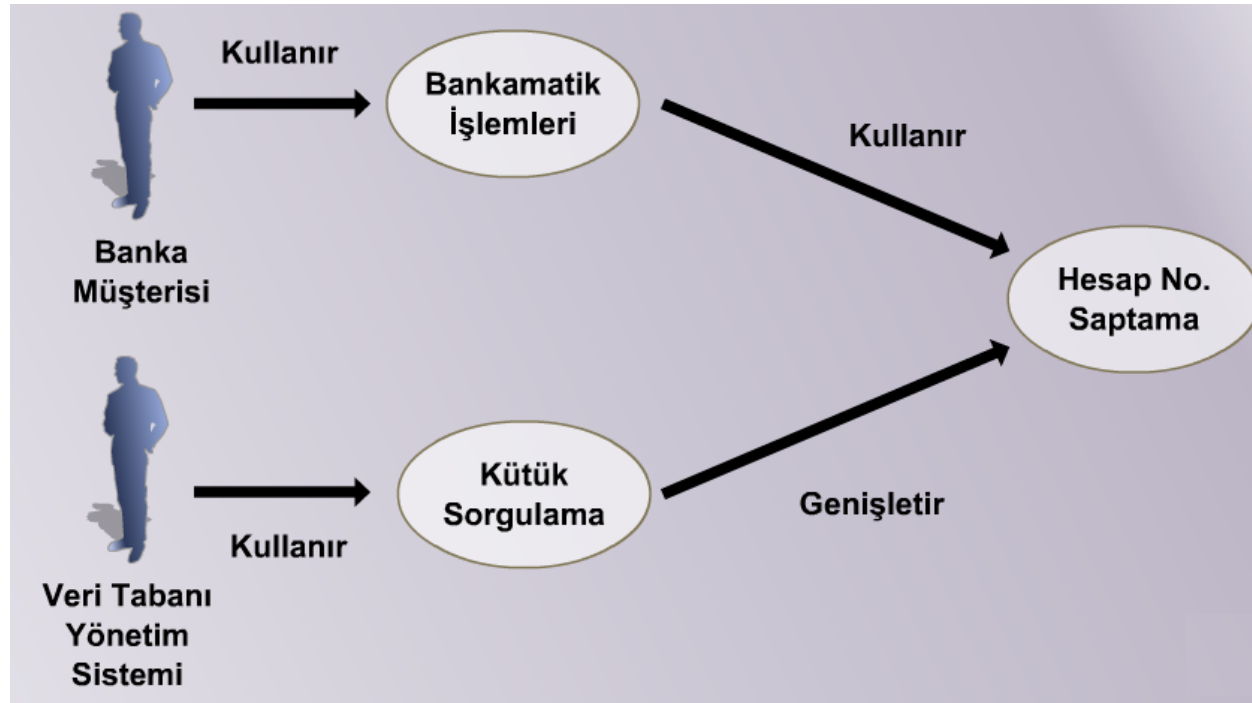
- İlk gereksinim derleme çalışmalarından sonra, sistemin kullanımıyla ilgili senaryoları denemeye yönelik modeller çizilir. Bunlarda 'aktörler' sistemin işleminde temel bazı süreçlerin harekete geçirilmesini sağlarlar.
- Aktörler, insan veya sistemin bir ögesi olabilir (işletim sistemi, bir dış etken vb.).
- Yalnızca bir süreci başlatmak için değil, sistem ile bir işlem boyunca herhangi bir etkileşimde bulunmak üzere de yer alabilirler.



Kullanım Durumu Diyagramı



Bankamatik Sistemi Örneğinde Kullanım Diyagramı Modellemesi



Sınıf/Sorumluluk/İşbirlikçi Modeli

- Kullanım senaryoları irdelendikten sonra sistemi oluşturan temel sınıflar, bunların sorumlulukları ve etkileşimleri ortaya çıkarılmalıdır.
- Sınıf/Sorumluluk/İşbirlikçi (SSI) modeli, sınıfları temsil eden kartlar kümesidir.
- Bu kartların başına sınıfın adı yazılır.
- Altta ise sınıfın sorumlulukları, solda ve sağda da işbirliği yapacak diğer sınıfların adları yazılır. Kartlar sanal olarak da tutulabilir.
- Sınıflar, dış varlıklar, doğal nesneler, olaylar, roller, kurum yapı birimleri, yerler ve yapılar gibi değişik kavramları temsil edebilirler.
- Doğal dil ile yapılmış sistem tanımından ilk önce dilbilimce isim olarak nitelendirilebilecek olanlar sınıf olmaya adaydırlar.
- Genelde bir seçimin sınıf olması için altı özelliği bulundurması istenir.

Sınıf/Sorumluluk/İşbirlikçi Modeli

Örnek

[illegible]

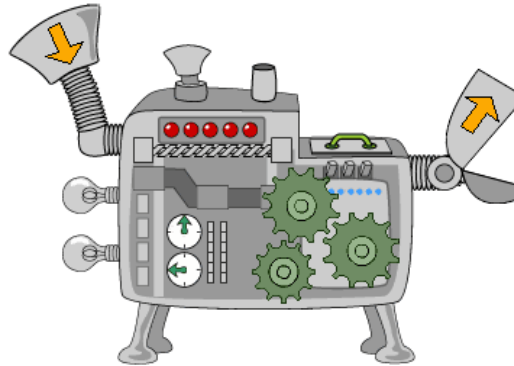
Bu özellikler;

- Bilgi saklama gereği,
- İşlem beklentisi,
- Birden çok özellik,
- Ortak özellikler,
- Ortak işlemler ve
- Temel ihtiyaç olma

biçimindedir.

Nesneye Yönelik Çözümleme Teknikleri

- Önceki sayfalarda söz edilen teknikler, bu bölümde biraz daha ayrıntılı bir şekilde incelenmektedir.
- Ancak örnek olarak verilecek diyagramların değişik metodolojilerde farklılıklar gösteren çizimlerinin olacağı hatırlanmalıdır.
- Bu ayrılıklar konunun anlatımı açısından çok da önemli değildir.



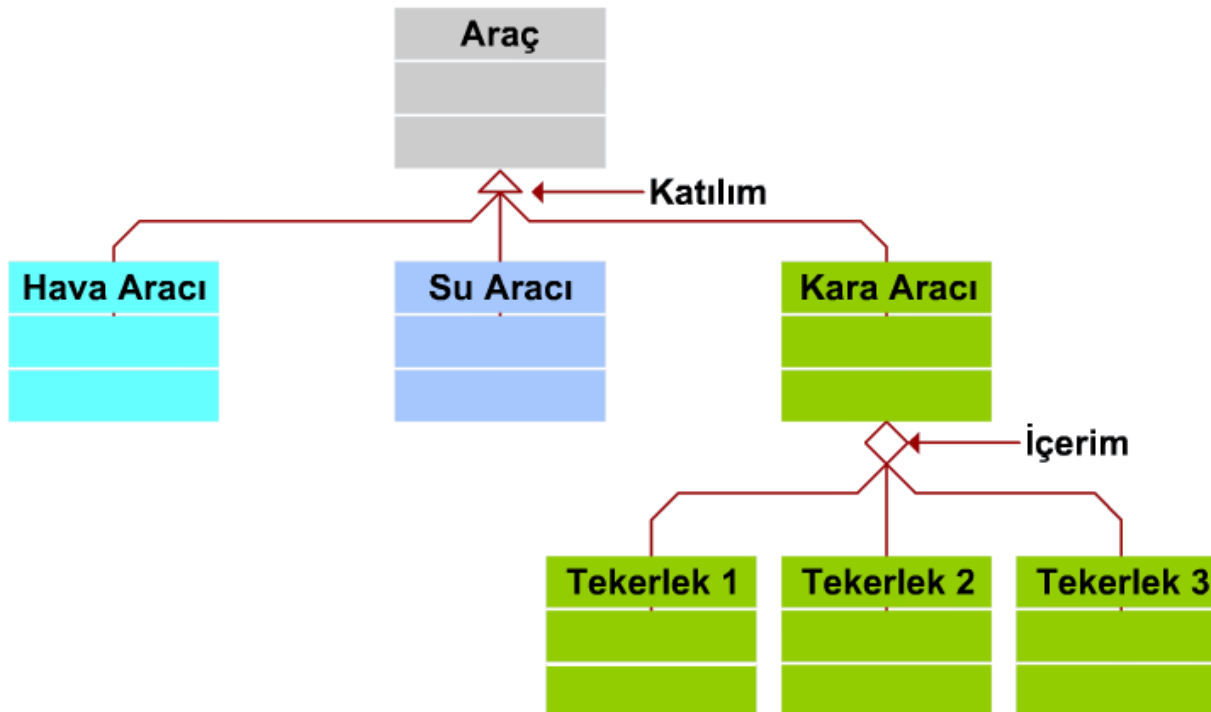
Yapısal İlişkiler

- Gereksinim çalışması sırasında sistemi tanımlamada kullanılabilecek bir çok yazılı belge oluşacaktır.
- Bu belgelerde, nesne olmaya aday olan varlıklar ve sınıf olmaya aday olan varlıklar önce belirlenir ve bunların arasında yapısal ve ilişkisel bağlantılar kurularak modelleme işlemi başlar.
- Yalnızca sınıflar ya da yalnızca nesneler ile başlamak zorunlu değildir.
- Yapısal ilişkiler, başlıca kalıtım ve içerim kavramlarına dayalıdır.
- Sistemde bulunacak nesne sınıflarında genelleştirme ve özelleştirme ilişkileri aranarak bunlar modele kalıtım bağlantısı olarak yansıtılır.

Yapısal İlişkiler Örnek

- Araçlar daha genel olup, su veya hava ortamlarında da yol alabilirler, yük taşıma amacıyla kullanılabilirler.
- Bir aracın taşıma kapasitesi, boş ağırlığı ve boyutları, onun durağan özellikleri olarak modellenenebilir. İşlemler olarak da yükleme ve gitme gibi örnekler verebiliriz.
- Sözü geçen özellik ve işlemler ile sarmalanmış olan araç sınıfının bir özelleştirmesi olarak otomobil sınıfını tanımlayabiliriz.
- Otomobil sınıfı, sözü geçen özellik ve işlemleri kalıtım yolu ile edindiğinden kendisininmişçesine içerir. Ayrıca otomobil sınıfına özel eklemeler yapılabilir.
- Örneğin dört tekerlek gibi. Bu dört tekerlek, basit veri yapısı olarak tanımlanabileceği gibi, birer nesne olarak da modellenenebilir.
- Otomobil sınıfının özelliği olarak yer alan tekerlek nesneleri, aslında 'içerim' yolu ile yapısal bir bağ kurmuşlardır.
- Bir otomobil, araç sınıfına bağlı olduğu gibi tekerlek sınıfına da bağlı olabilir (programcılar bu kolaylığa kaçıcı yolu bazen yeğlerler), ancak bu, modelleme açısından doğru bir yaklaşım olmaz.
- Çünkü kalıtım, bir genelleme/özelleşme ilişkisidir.
- Otomobiller araçtırlar, ama tekerlek değildirler. Bir sonraki slaytta Kalıtım ve İçerim bağlarını birlikte gösteren bir örnek içermektedir.

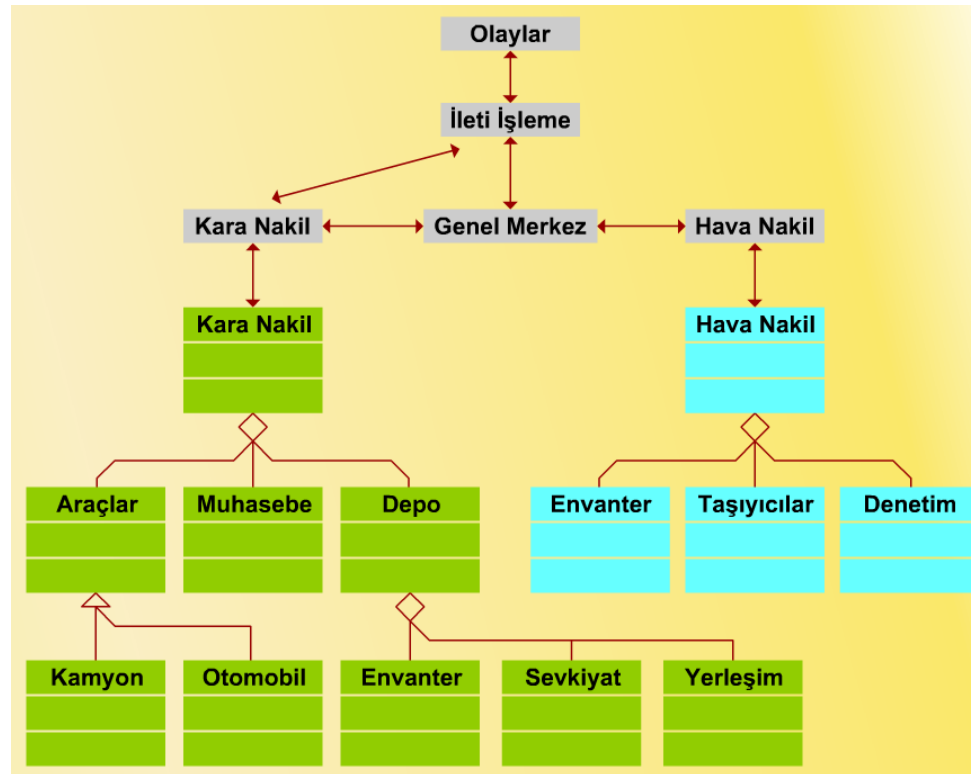
Kalıtım ve İçerim Örnek



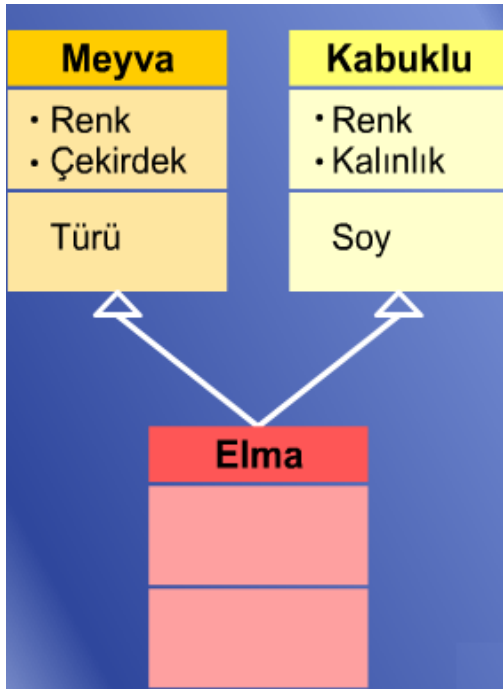
Alt Sistem Modellemesi

- Çözümleme aşamasında modelin mantıksal olarak anlaşılabilirliği için bu anlamsal bağların doğru taşınmasında yarar vardır.
- Tasarım aşamasında ise, verimlilik gibi bazı etkenlerin çok önemli olmaları durumunda bazı değişiklikler sonradan yapılabilir.
- Sözü geçen yapısal ilişkiler, sistem modelinin sıra düzensel olarak ayrıştırılması kavramına karşı düşerler. Bu önemli kavramı destekleyen bir üçüncü araç ise 'alt sistem' öğeleridir.
- Yapışıklık prensibinden hareketle bir işlem grubunun çalıştırılmasında emeği geçecek nesnelerin bir araya toplanarak yapay bir büyük nesne gibi değerlendirilmeleri, bir alt sistemin tanımlanmasıdır.
- Bazı yöntemlerde alt sistemlere 'özne' veya 'öge' gibi isimler verilmiştir. NY tekniklerde alt sistem anlayışı ve içerim kavramının da kullanımının yanı sıra en ağırlıklı olarak kalıtım kavramı ile çözümleme modelleri oluşturur.

Alt Sistem Modellemesi Örnek



Çoklu Kalıtım

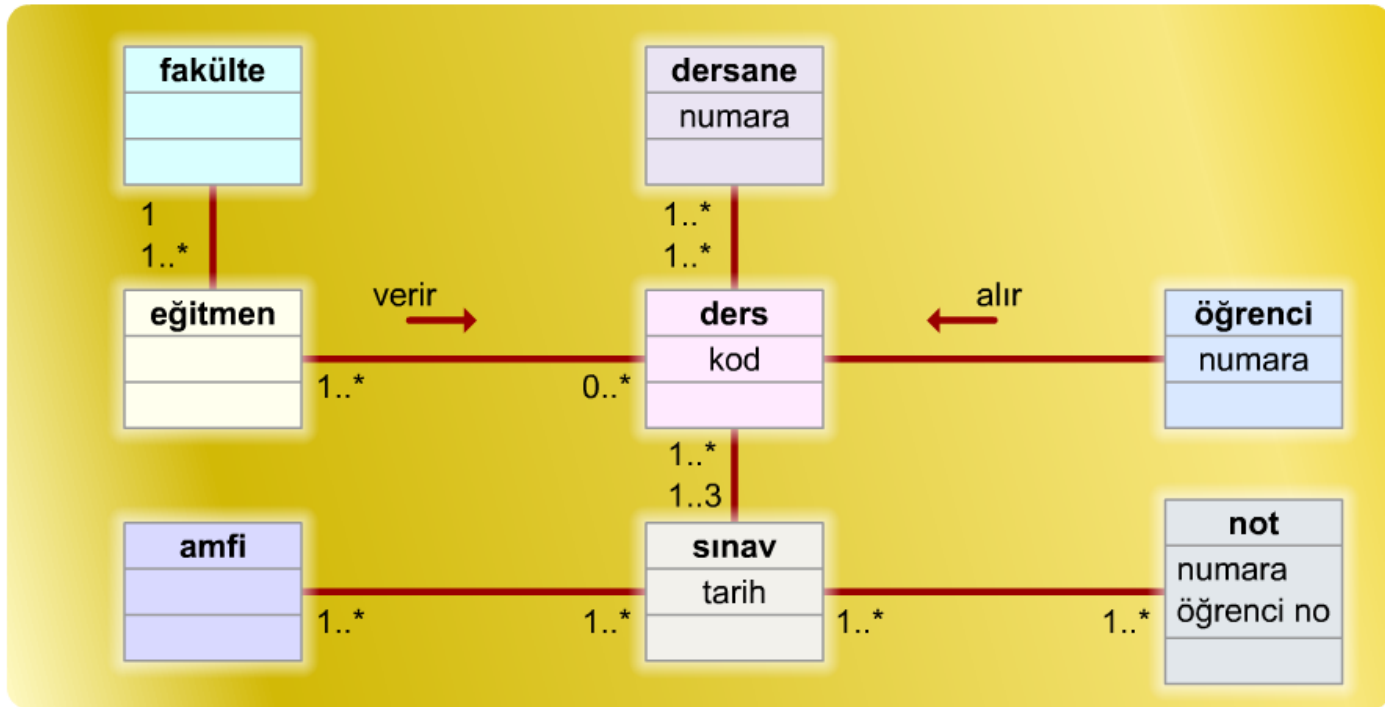


- Bazı durumlarda birden çok sınıfın özelliklerini kalıtım yolu ile almak uygun olabilir.
- Kalıtımı alan sınıfa, **özelliklendirilmiş sınıf**, kalıtım alınanlara da **genel sınıflar** denebilir.
- Bu durumda özelliklendirilmiş sınıf, kavramsal olarak her iki genel sınıfın içinde de bulunabilmelidir (bir elmanın hem meyva, hem de kabuklu cisim sınıflarına girebileceği gibi.)
- Genelde kalıtım için geçerli olan kural, bu durumda iki kere uygulanabilir olmalıdır:
- "**Özelliklendirilmiş sınıf, genel bir sınıfın üyesidir.**" ifadesi geçerli ise kalıtım anlamlıdır.
- Bu ifadeyi elma örneğine uygularsak, "**Elma bir çeşit meyvedir.**" ve "**Elma bir çeşit kabuklu cisimdir.**" ifadeleri anlamsal olarak geçerli ifadelerdir.
- Yan tarafta çoklu kalıtım örneği modellenmektedir.

Çoklu Kalıtım İlişkilerin Gösterimi

- Geleneksel çözümleme tekniği olarak karşımıza çıkmış olan Nesne İlişki Diyagramları, NY teknolojisine uyarlanarak az değişiklikle kendisini kabul ettirmiştir.
- Yine nesneler arası ilişkiler isimleri konarak ve çoğullama belirtimi kullanılarak çizilmektedir.
- Genelde ikili ilişkiler kullanılmakla birlikte üçlü (üç nesne arasında bir ilişki) ve daha çoklu olanlarını da tanımlamak mümkündür. NY tekniklerinde ilişkiler yalnızca bir isim olmanın yanında, ayrı bir nesne olarak da tanımlanabilir. İlişkilerin iç özellik ve işlemleri olabilir.
- Ayrıca dönüşlü ilişkiler de tanımlanabilir.
 - Örneğin 'personel' sınıfı olarak modellenen sınıf içerisinde bir amir/memur ilişkisi tanımlanabilir ve bu, diyagramda aynı sınıftan çıkıp aynı sınıfta sonlanan bir oklu çizgi üzerine 'memur' ilişki tanımı yazılarak yapılabilir.

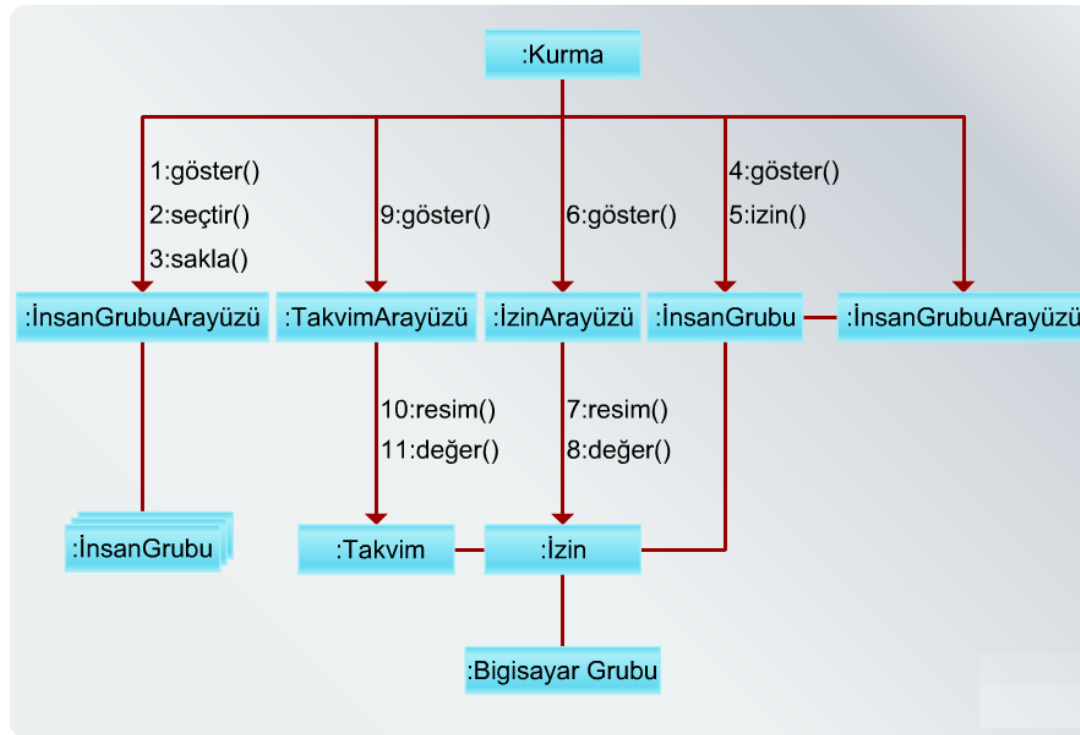
Sınıf Diyagramında İlişkiler



İşbirliği Diyagramları

- Kullanım durumu diyagramları ile irdelenen sistem düzeyindeki işlemlerin gerçekleştirilmeleri için birlikte çalışması gereken nesnelerin bu işbirliklerini yansıtır.
- İlk tekniklerdeki genelde sınıf ve nesne diyagramı ile yola çıkılma anlayışında nesne diyagramları, ileti trafiğini yansıtmakta idi.
- Bu nesne diyagramlarına karşı düşen yeni tekniklerde ise işbirliği diyagramı kullanılmaktadır.
- Biraz daha organize olarak hangi sistem işleminin yansıtıldığı sınırları ile belirtilerek ve kullanım örneği, SSİ gibi tekniklerle modellenmiş gereksinimlere dayandırılarak çizilir.
- Modeller arasındaki iletilerin sıralandırması ve biçimi belirtilerek sistem işleminin gerçekleştirilmesi modellenir.

İşbirliği Diyagramı



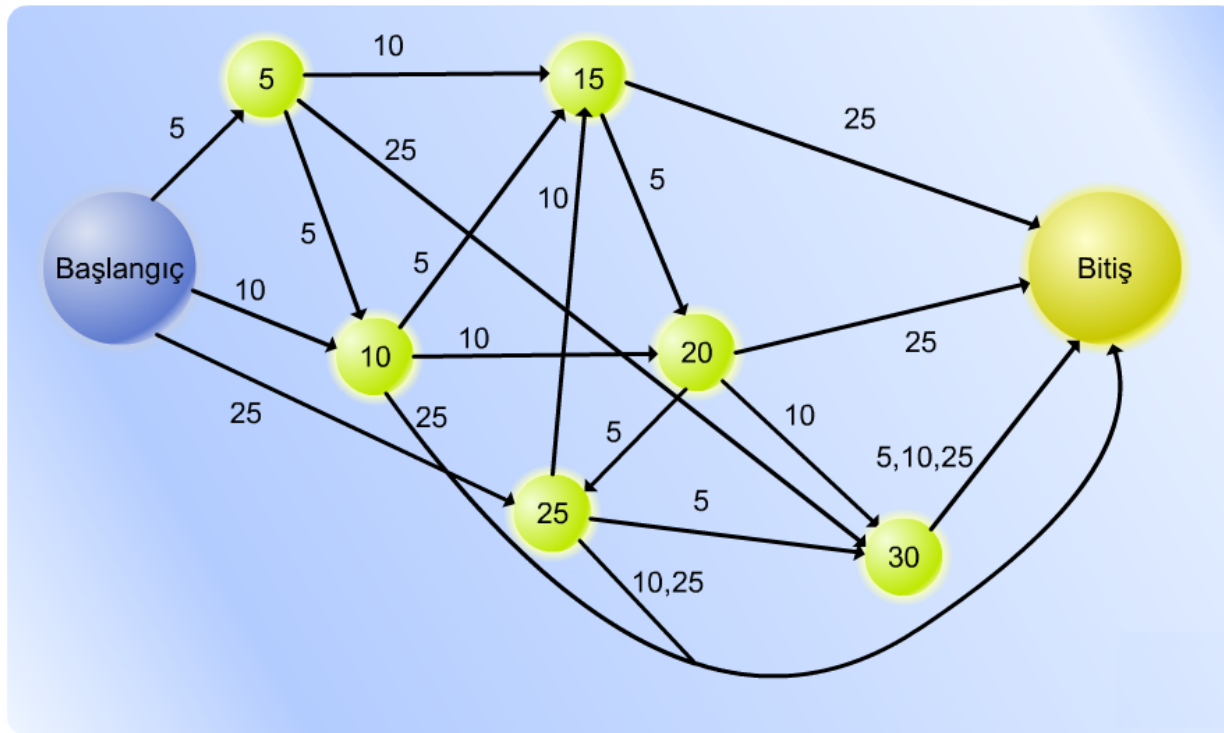
Davranış Modellemesi

- Çözümleme sırasında ayrıntıya girildikçe tasarımı ilgilendiren düşünceler söz konusu olur.
- Sistem davranışı, işbirliği diyagramları ile üst düzeyde ortaya konmuştur.
- Ancak bu işbirliği diyagramlarının bazı karmaşık kontrol gerektiren durumlarda ileti sıralandırması gibi kararları neye dayanarak yaptığının belirtilmesi istenebilir.
- Nesneler, daha önceki işlemlerin sonucu olarak bir duruma gelirler ve davranışları bu durumlara bağlı olarak ortaya konulur.
- Bu noktada 'durum modellemesi' konusuna girilmesi gerekir.
- Bazı metodolojiler, durum modellemesi konusunun çok ayrıntılı olduğunu ve daha çok tasarıma bırakılacak bir uğraşı olduğunu savunurlar.
- Durum sayısı artınca model karmaşıklaşır ve anlaşılması zorlaşır.

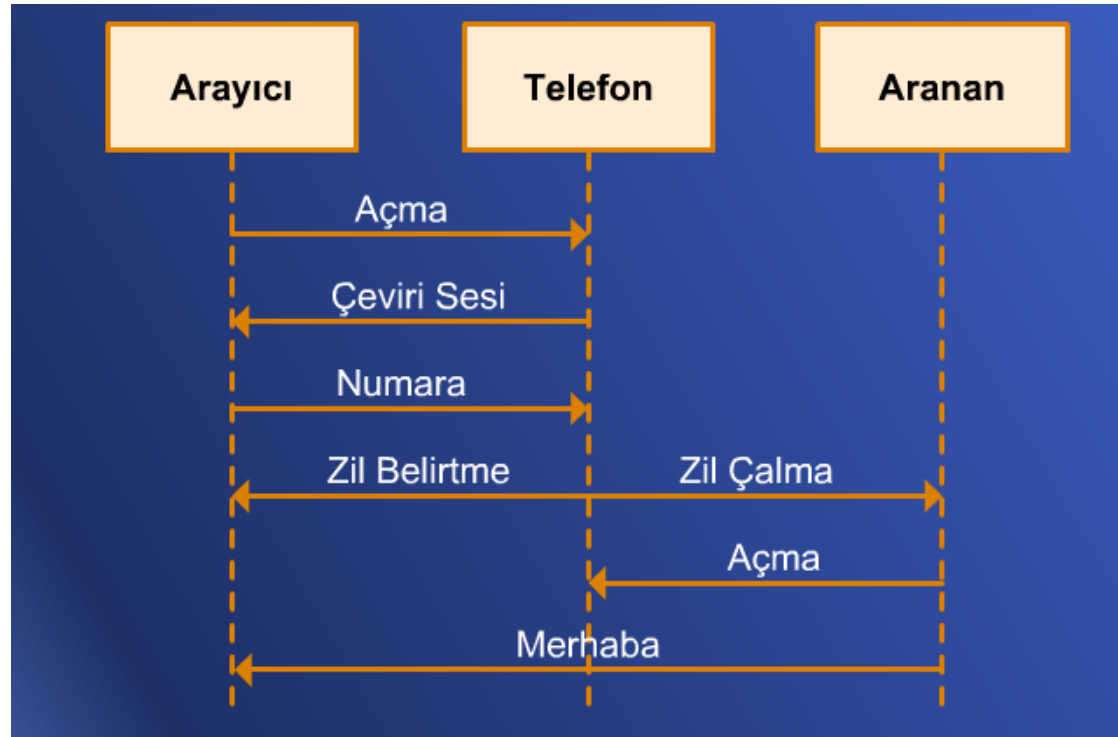
Davranış Modellemesi

- 'Durum Makinesi' modelleri, sistemin tümü için çok karmaşık olacaktır.
- Bu yüzden çok işlemliliği de yansıtabilen 'Petri Net' gibi modeller, yapısal teknolojiler zamanında sıkça kullanılmışlardır.
- Genelde de çözümleyicinin bu gibi diyagramları çizmesini istemektense bir şekilde serbestçe tanımladığı bilgilerden BDYM araçlarınca iç belirtim olarak kullanılmak üzere otomatik olarak elde edilmişlerdir.
- Ancak çok işlemliliği sağlamanın bir yolu da bilinen durum makineleri tabanlı modelleri, değişik nesneler içerisinde ayrı ayrı oluşturup (her biri tek işlemlili modeller olarak) bunların işbirliğine imkan tanımaktır.
- Durum diyagramlarına destek olacak ardışık diyagramlar gibi araçlar da kullanılmaktadır.
- Sonraki slaytlarda durum diyagramı ve ardışık diyagramı örnekleri verilmektedir.

Durum Diyagramı



Ardışık Diyagramı



Nesneye Yönelik Tasarım



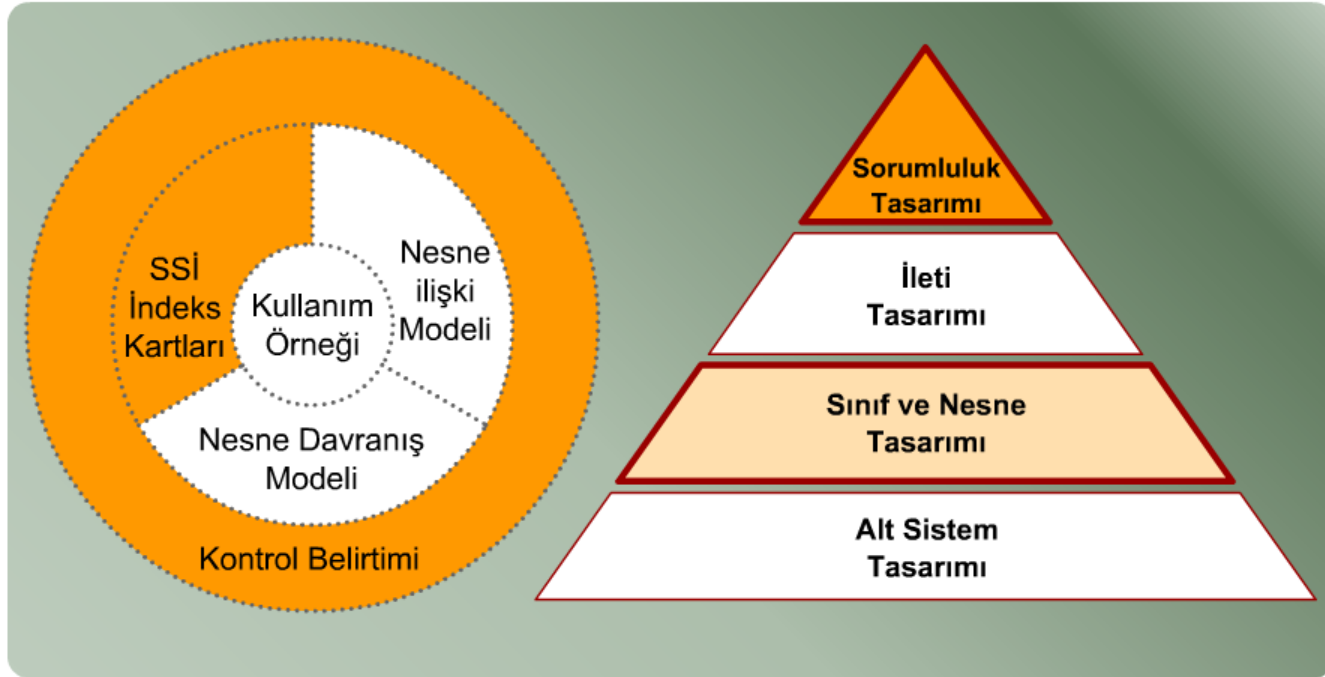
Nesneye Yönelik Tasarımda Katmanlar

- Nesneye Yönelik (NY) Tasarım, NY Çözümleme ile elde edilen modelin, yazılımın oluşturulması için yeterli tanımları sağlayacak şekile dönüştürülmesidir.
- Kavram olarak geleneksel tasarım ile benzerlikler gösterir.
- İki yaklaşımda da veri ve süreç belirtimi benzer şekilde yapılabilir.
- Algoritmik düzeyde süreç tanımları da farklı değildir.
- NY teknikler, yazılım tasarımının ilkeleri olan soyutlama, bilgi saklama, işlevsel bağımsızlık ve modülerliğe çok yatkındır.
- NY Tasarımı yanda görüldüğü gibi dört katman olarak inceleyebiliriz:
- Ayrıca bütün bu katmanlar bir 'saha nesneleri' bilgisinin oluşmasına dayalıdır.
- Belirli bir konuda yazılım geliştirilirken o konu ile ilgili genel modeller ve bunların uzantısı olan bir nesneler kitaplığı kurulmuş olmalıdır.
- Böyle bir yapı hazır değil ise eldeki projeler için gerekli altyapı olarak bir taraftan da saha altyapısının tasarımı yapılır.

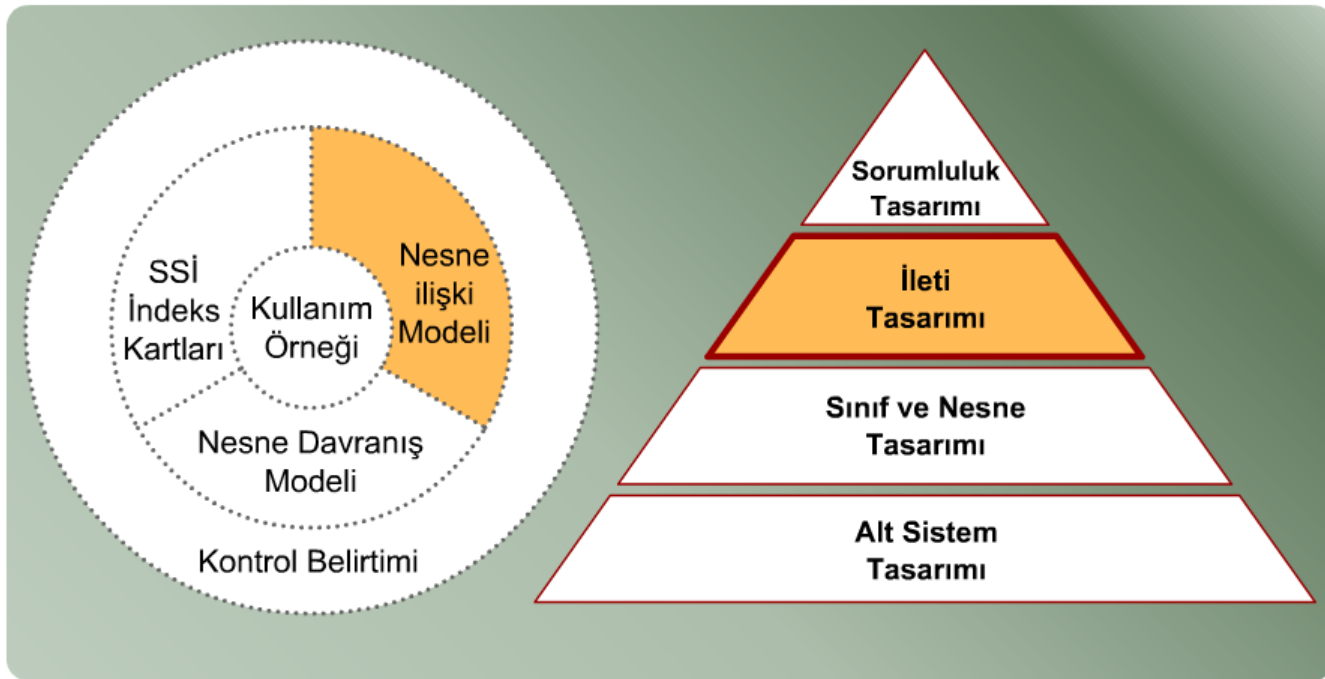
Nesneye Yönelik Tasarım

- Geleneksel tasarımın tersine, NY tasarımda NY çözümlemeden devir alınacak yapılar oldukça sadık kalınarak kullanılır.
- Ancak ihtiyaçlar çalışmasında kullanılan NY olması gerekmeyen modeller bir şekilde NY yapılara yansımalıdır.
- Sonraki slaytlarda NY çözümlemeden NY Tasarıma geçişte kullanılan kavram ve ortamları gösterilmektedir.

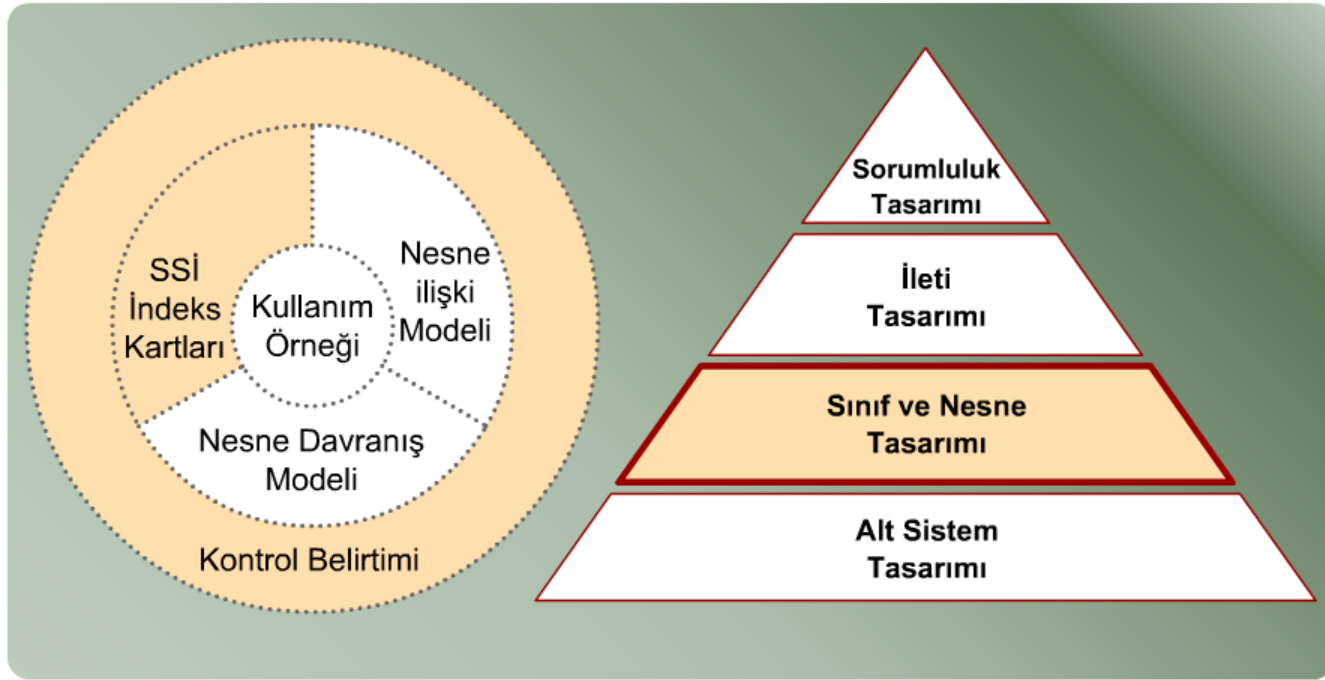
Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş



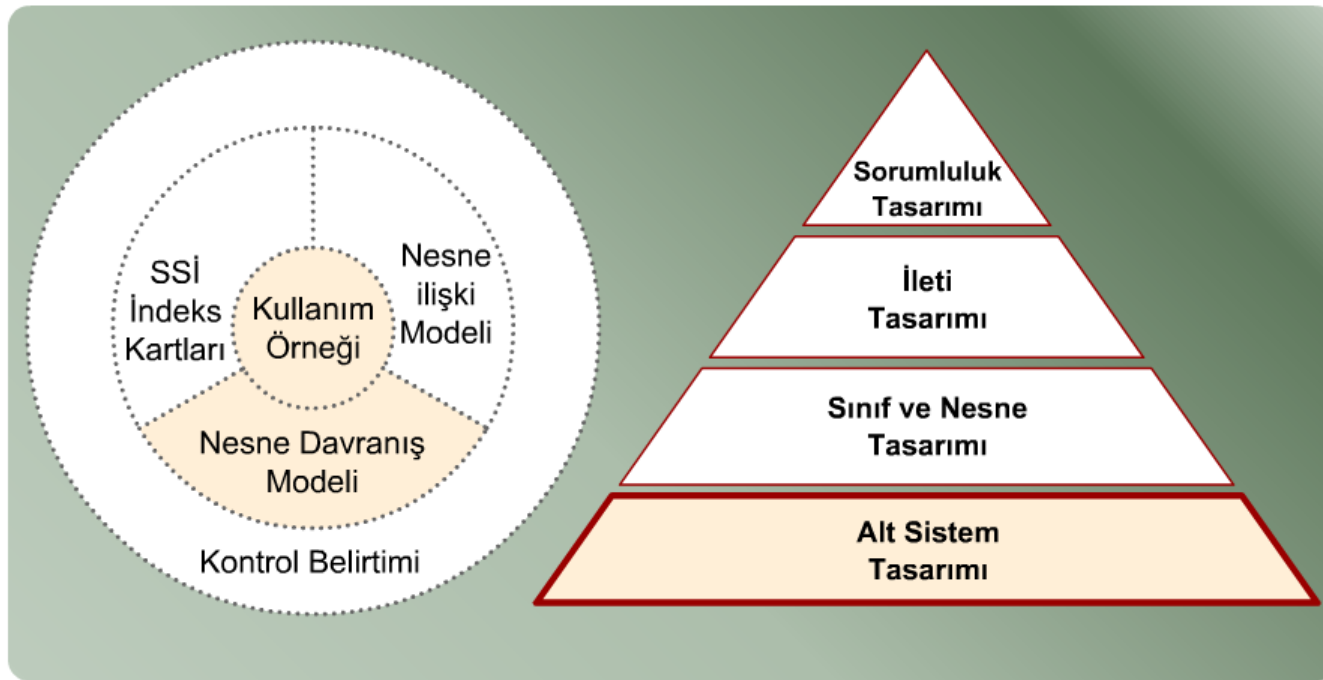
Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş



Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş



Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş



Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş

- Geleneksel ve NY Tasarım yaklaşımlarında en büyük fark, yapısal modellemelerde ortaya çıkar.
- Bütüncül bir yaklaşımla geleneksel tasarımda merkezi yapı ve kontrol hiyerarşisi kurulurken NY yaklaşımda dağıtık bir yapı söz konusudur.
- Sistem, etkileşimleri belirtilen bağımsız nesnelerin tümü olarak yapılır.
- Kontrol kavramı da benzer şekildedir.
- Sistemin durumu ve dolayısıyla kontrol mekanizması, geleneksel yaklaşımda merkezidir.
- NY yaklaşımda ise iletilerle bağlantı kurulan dağıtık nesneler, sistemin genel durumunu belirler ve kontrol de dağıtık olarak sağlanır.

Nesneye Yönelik Yaklaşımda Çözümlemeden Tasarıma Geçiş

Birbirinden farklı olan geleneksel ve NY tasarım metodolojilerinin karşılaştırılmasında kullanılmak üzere 10 özellikten söz edilebilir:

1. Modül sıradüzen gösterimi
2. Veri tanımları belirtimi
3. İşlem mantığı belirtimi
4. Sondan sona süreç sıralamalarının gösterimi
5. Nesne durumları ve durum geçişlerinin gösterimi
6. Sınıf ve sıradüzen tanımları
7. Sınıflara işlemlerin tayin edilmesi
8. İşlemlerin detaylı tanımı
9. İleti bağlantılarının belirtimi
10. Sistem işlemlerinin tanımlanması

Nesneye Yönelik Tasarım Metodolojileri

Çözümlemede olduğu gibi tasarımda da bir çok farklı gibi görünen yöntem kullanılmaktadır. Aslında metodolojiler genelde çözümleme ve tasarım safhalarını diğer safhalar ile birlikte içerirler. Modüler yapının tasarım için öneminden daha önce söz edilmişti.

Bir tasarım metodunun modülerlik ve ilişkili NY kavramlarını desteklemesi için beş kriter ortaya atılmıştır:

1. **Çözünürlük:** Büyük bir problemin, çözülmesi daha kolay olan küçük problemlere ayrıştırılması yöntemidir.
2. **Bütünleştirme:** Geliştirilecek modüllerin ne kolaylıkla değişik sistemlerce kullanılabileceğidir.
3. **Anlaşılabilirlik:** Bir bileşenin, diğerlerine başvurmadan ne kolaylıkla anlaşılabilirliği.
4. **Devamlılık:** Programda yapılacak değişikliğin yalnızca bir veya birkaç modülü etkilemesidir.
5. **Koruma:** Programda yapılacak değişikliğin yalnızca bir veya birkaç modülü etkilemesidir.

Nesneye Yönelik Tasarım Metodolojileri

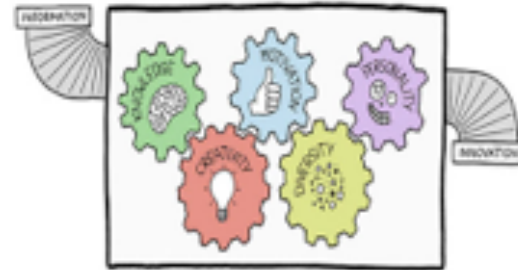
Aynı şekilde modüler yapının sağlanması için de beş prensipten söz edilebilir:

1. Modüler yapıyı destekleyici dil öğelerinin bulunması,
 2. Az arayüz,
 3. Arayüz içinde sınırlı bilgi akışı,
 4. Arayüzün bağımsız olarak açıkça anlaşılabilmesi (örneğin merkezi yapılara başvurulmaması),
 5. Bilgi saklama.
-
- Her ne kadar bu prensipler geleneksel tasarım için de geçerli ise de NY yaklaşımların daha etkili olarak bu prensipleri elde ettiği görülmektedir.
 - Özellikle arayüz tanımları, bileşen tabanlı yaklaşımlarda iyice önemli olmaktadır.

NY Metodolojilerinin Tasarım Yöntemleri

Bazı Nesne Yönelimli metodolojilerinin tasarım yöntemlerini aşağıdaki şekilde maddelersek bunlar;

1. Booch Metodu
2. Coad ve Yourdon Metodu
3. Jacobson (OOSE) Metodu
4. Rumbaugh (OMT) Metodu



Booch Metodu

Yapısal Planlama:

- Benzer nesneler ayrı kısımlarda toplanır.
- Nesneler soyutlama düzeylerine göre katmanlara ayrılır.
- Senaryolar tanımlanır.
- Tasarım öntipi oluşturulur.
- Öntip, kullanım senaryoları ile sınanır.

Taktik Tasarım:

- Özellik ve işlem kullanım kurallarının tanımını,
- Bellek idaresi, hata mesajları gibi altyapı işlevleri ile ilgili kuralların tanımını,
- Kuralların anlamını tanımlayan bir senaryonun geliştirilmesini,
- Her kural (politika) için bir öntipini,
- Öntip iyileştirmesini,
- Her politikanın, yapısal vizyonu iletmesi açısından gözden geçirilmesini kapsar.

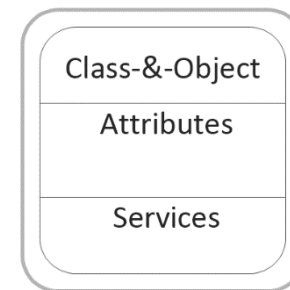
Coad ve Yourdon Metodu

Problem Ortamı Bileşeni

- Ortamı oluşturan bütün sınıflar gruplandırılır.
- Uygulama sınıfları için hiyerarşi oluşturulur.
- Uygun düştükçe kalıtım basitleştirilir.
- Verimlilik için tasarım uyarlanır.
- Alt düzey nesneleri, gerektiğinde eklenir ve iyileştirilir.
- Tasarım gözden geçirilir ve çözümlemeye yapılacak ilaveler sorgulanır.

İnsan Etkileşim Bileşeni

- Kullanıcılar belirlenir.
- Görev senaryoları geliştirilir.
- Kullanıcı komutları sıradüzeni tanımlanır.
- Kullanıcı etkileşim sıralandırması iyileştirilir.
- İlgili sınıflar ve sıradüzenleri tasarlanır.
- Grafik arayüz sınıfları ile bütünleştirilir.



Coad ve Yourdon Metodu

Görev Yönetimi Bileşeni

- Görev çeşitleri (olay veya saat ile tetiklenen gibi) tanımlanır.
- Öncelikler oluşturulur.
- Bir görev, diğerlerinin yöneticisi olarak atanır.
- Her görev için uygun nesneler tasarlanır.

Veri Yönetimi Bileşeni

- Veri yapıları tasarlanır.
- Veri yapılarının idaresi için gerekli servis işlemleri tasarlanır.
- Veri idaresi için araçlar tanımlanır.
- Uygun sınıf ve hiyerarşiler tasarlanır.

Jacobson (OOSE) Metodu

1. Çözümleme modelinde gerçek dünyaya uygunluk ayarlamaları.
2. Temel tasarım nesnesi olarak 'öbek'lerin tanımlanması:
 - İlgili çözümleme nesnelerinin uygulanacağı bir blok tanımlanır.
 - Arayüz, varlık ve kontrol blokları tanımlanır.
 - İşletim sırasında blokların nasıl haberleşeceği açıklanır.
 - Bloklar arası uyarı işaretleri ve bunların haberleşeceği açıklanır.
3. Bloklar arası uyarı iletilerini gösteren etkileşim diyagramları çizilir.
4. Bloklar, alt sistemler altında düzenlenir.
5. Tasarım gözden geçirilir.

Rumbaugh (OMT) Metodu

1. Sistem Tasarımı

- Çözümleme modeli alt sistemlere ayrılır.
- Problemin gerektirdiği eşzamanlılıklar tanımlanır.
- Altsistemler ve süreçleyiciler görevlere dağıtılır.
- Veri idaresi için temel strateji seçilir.
- Ortak kaynaklar ve bunlara erişim kontrol edecek yapılar tanımlanır.
- Sistem için uygun kontrol mekanizması tasarlanır,
- Sınır koşulları göze alınır.
- Parametreler arası çıkar çatışmaları gözden geçirilir.

2. Nesne Tasarımı

- Çözümleme modelinden işlemler seçilir.
- Her işlem için yordam tanımlanır.
- Yordamlar için uygun veri yapıları tanımlanır.
- İç sınıflar tanımlanır.
- Nesne düzenlemesi veri ulaşımı ve hesaplama verimi açısından gözden geçirilir.
- Sınıf özellikleri tasarlanır.

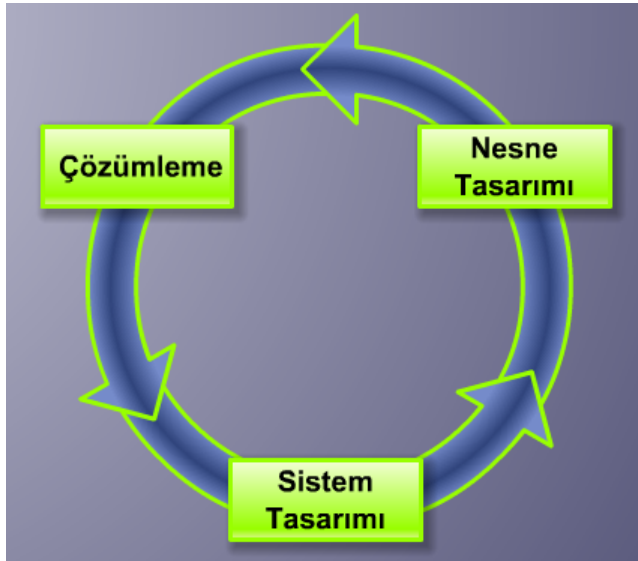
3. Sistem tasarımındaki kontrol mekanizmaları uygulanır.

4. Sınıf yapıları kalıtımı kuvvetlendirici yönde ayarlanır.

5. Nesne ilişkileri için ileti tasarımı yapılır.

6. Sınıflar modüller altında toplanır.

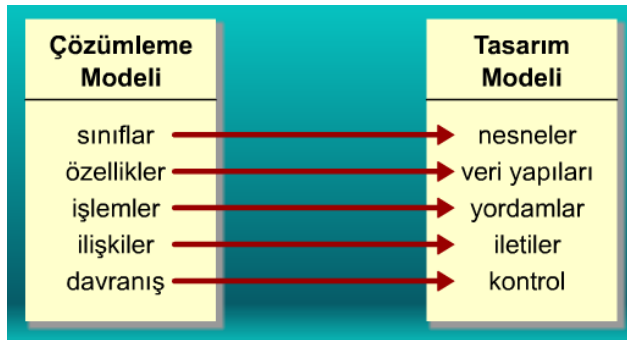
Genel Olarak NY Metodolojiler



Nesneye Yönelik Çözümlemeden
Tasarıma Geçiş Çevrimi

- NY çözümlemede olduğu gibi, NY tasarımda da değişik yaklaşımlar olmasına rağmen benzer işlemler uygulamaktadır.
- NY yaklaşımlarda bazen çözümleme ve tasarım arasındaki ayrımı yapmak zorlaşır.
- NY Çözümleme, bir sınıflandırma işlemidir.
- Problem yapısı, nesne sınıfları olarak organize edilir ve nesne ilişkileri ile problemin davranışı modellenir.
- Çözümlemeden tasarıma geçiş süreci, genel hatlarıyla yanda gösterildiği gibi bir çevrim içerir.
- Tasarım ise tanımlanmış sınıflardan çıkarılacak nesneleri tanımlar ve bu nesnelerin birbirleri ile ilişkisini gösterir.
- Ayrıca nesnelerin davranışları ve haberleşmeleri belirtilir.

Genel Olarak NY Metodolojiler



Nesne Tasarımında
Çözümleme Modelinden Geçiş

- Tasarımcı, önce sistemin bütününe ele almalıdır.
- Kullanıcı istekleri ve bunların gerçekleştirilmeleri için gerekli alt sistem karakteristiklerinin tanımlanması gerekir.
- Alt sistemlerin tasarımı sırasında Coad Yourdon metodolojisinde sözü geçen dört önemli tasarım bileşeni tanımlanır.
- Daha sonra nesne tasarımına sıra gelir.
- Yanda gösterildiği gibi, Sınıf Sorumluluk İşbirliği (SSİ) modelindeki öğeler, tasarıma dönüştürülür.

Genel Sistem Tasarımı

- Değişik yaklaşımlardan söz edilmiş ise de sistem tasarımı konusunda Rumbaugh yaklaşımı burada tercih edilmiştir.
- Daha önce tanımlanan sistem tasarımı adımlarında yapılacak ayrıntılı işlemler, bu bölümde açıklanmaktadır.
- Çoğu yaklaşım, önce sistem/alt sistem ayrıştırması ve bu ayrıştırılan bileşenler arası arayüzün tanımı ile tasarıma başlar.

Çözümleme Modelinin Ayrıştırılması

- Alt sistemleri tanımlarken, sistemin sınıf, ilişki ve davranışlarını 'yapışık gruplar' olarak bir araya getirmek gerekir.
- Bir alt sistem içerisinde bulunacak bileşenlerin ortak yanı bulunmalıdır.
- Aynı sistem işlemini yerine getirmek, aynı donanım birimi içerisinde bulunmak veya aynı kaynakları idare etmek gibi.
- Sistem açısından, alt sistemler sorumluluklarına göre tanımlanırlar.
- Bu tanımlama yapılırken de aşağıdaki tasarım kriterleri göz önünde bulundurulur:
 - Sistemin gerisi ile yapılacak haberleşmenin tümü, iyi belirlenmiş bir arayüz üzerinden olmalıdır.
 - Sınıflar, yalnızca aynı alt sistem içerisinde işbirliği yapmalıdırlar (Yalnızca küçük sayıdaki haberleşme sınıfları dışında).
 - Alt sistem sayısı küçük tutulmalıdır.
 - Karmaşıklığı azaltmak üzere alt sistemler kendi içlerinde bölünebilirler.
 - Haberleşen alt sistemler istemci/sunucu gibi bağlantı protokollerine sahip olabilirler.

Eşzamanlılık ve Alt Sistem Belirleme

- Nesne davranış modeli açısından, iki nesne aynı zamanda aktif değilse bir eşzamanlılık yoktur.
- Bu iki nesne aynı donanım (süreçleyici) üzerinde uygulanabilirler.
- Ancak eğer iki nesne aynı zamanda ve asenkron olarak olaylara karşılık veriyorlarsa eşanlılık vardır.
- Eşanlı alt sistemler için iki yerleştirme seçeneği vardır:
 1. Ayrı süreçleyicilere yerleştirme,
 2. Aynı süreçleyiciye yerleştirme ve işletim sisteminin eşzamanlılık desteğini kullanma.



Görev Yönetimi Bileşeni

➤ Eşzamanlı görevleri yöneten nesnelerin tasarımı için Coad ve Yourdon aşağıdaki stratejiyi önermektedirler:

1. Görevin özellikleri ortaya çıkarılır:
 - başlatılma (olay veya saat ile tetiklenme..)
 - öncelik (görevlerin birbirinden önce başlatılması)
 - önemlilik (kritik bir görevin kesilmemesi)
2. Bir yönetici 'görev' ve ilişkili nesneler tanımlanır.
3. Yönetici ve diğer görevler bütünleştirilir.

➤ Görevler, standart tasarım modeline girmeden önce bir şablon ile şekillendirilirler.

➤ Burada görevin ismi, açıklanması, önceliği, sorumlu olduğu işlemler, nasıl başlatıldığı ve haberleşmesindeki girdi/çıktı veri değerleri bulunur.

Veri Yönetim Bileşeni

- Sistem açısından bakıldığında tek bir veri tabanı yönetim sistemi ile her türlü ihtiyaç karşılanmaktadır.
- Ancak sistem düzeyindeki ihtiyaçlar ile veri yapılarının işlemleri için gereken alt düzeydeki ihtiyaçların ayırt edilmesi gerekir.
- Nesnelerin özelliklerinin kaydedilebilmesi ve yeniden okunabilmesi gerekir.
- Nesneye yönelik veritabanı yönetim sistemleri artık ticari ürünler olarak piyasaya sürülmüşlerse de henüz fazlaca kullanılmamışlardır.
- Çoğu zaman uygulamada nesneler bir şekilde ilişkisel veri tabanlarında saklanmaktadır.
- Nesnelerin kayıt ve yeniden okunma işlemlerini yönetmek üzere önlemler alınmalıdır.

Kaynak Yönetimi Bileşeni

- Genel olarak, alt sistemler kaynaklar için yarışır.
- Kaynakların kontrolsüz bir şekilde, isteyen birime atanması sorun çıkarır.
- 'Gözetleyici Nesneler' ile bir kaynağın kullanımı için hangi alt sistemin nasıl ve ne zaman erişime kavuşabileceği düzenlenebilir.
- Alt sistemler önce bu gözetleyiciler ile haberleşerek izin aldıktan sonra kaynağa erişebilirler.
- Bu erişim politikası çizilmeli, gözetleyici nesnelerin bu politikalar doğrultusunda işlemleri tanımlanmalıdır.

Kullanıcı Arayüzü Bileşeni

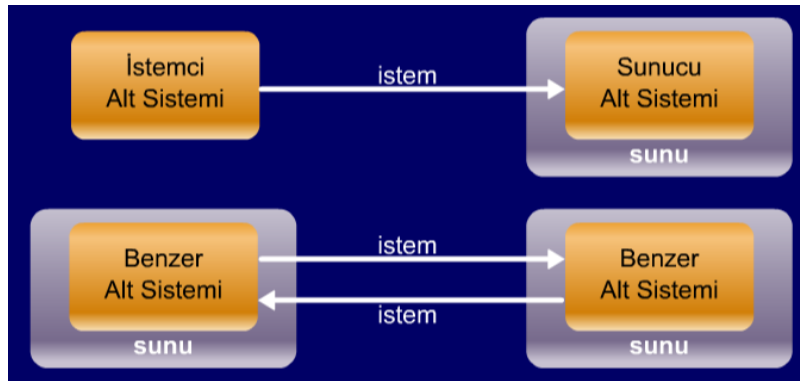
- Kullanım örnekleri gibi ortamlardan yola çıkarak kullanıcı ve sistem etkileşimlerini tanımlamak doğaldır.
- Roller ve kullanım senaryoları tanımlandıktan sonra, komut hiyerarşisi ortaya çıkarılır.
- Komut hiyerarşisi, bütün kullanım örnekleri kapsanana kadar ana menüden başlayarak alt menülerin ve daha alt kumanda yapılarının tanımlanması ile tasarlanır.
- Bu konuda yardımcı olabilecek hazır nesneler çoğu geliştirme ortamında mevcuttur; örneğin Visual BASIC veya Delphi.

Alt Sistemler Arası İletişim

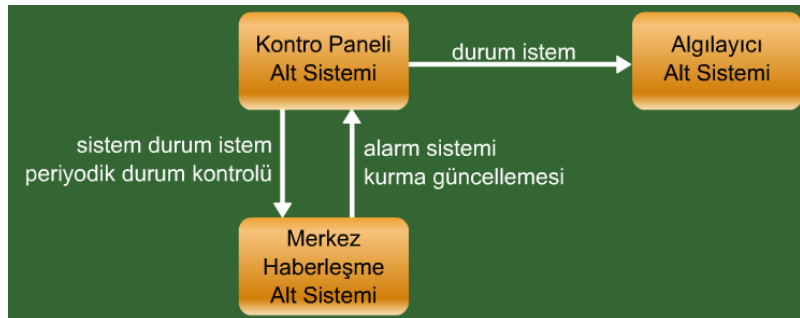
Alt sistemler tanımlandıktan sonra bunlar arasındaki iletişim yöntemlerinin de belirtilmesi gerekir. Alt sistemlerde genelde iki tür haberleşme tipi yaygındır. Bu haberleşme tipleri, istemci/sunucu veya benzerler arası bağlantı tiplerinden oluşur. Alt sistemler arasındaki haberleşmeyi de istem ve karşılığında sunulacak bir hizmet şeklinde genellersek, bir hizmet sunusu için aşağıdaki tasarım adımları uygulanmalıdır:

1. Alt sistem ile işbirliğinde bulunanlarca yapılabilecek istemler sıralanır. İstemler alt sistemlere göre gruplanır ve sunular ile ilişkilendirilir.
2. Her sunu için nesne işlemleri belirlenir ve bunlar alt sistem dışından olmamalıdır.
3. Her sununun yer alacağı Şekil 9.16'daki gibi bir tablo oluşturulur. Bu tablodaki sütunlar:
 - Tip (İstemci/sunucu vb.)
 - İşbirlikçiler: Sunu ile ilgili alt sistemler
 - Sınıf: Sunu için gerekli 'nesne işlemleri'ni içeren sınıf isimleri
 - İşlem: Belirtilen sınıf içerisinde sunuyu gerçekleştiren işlemler
 - İleti biçimi şeklindedir. Eğer etkileşim karmaşık ise, bir çeşit '**olay akışı**' olan diyagram çizilir.

Alt Sistem İşbirliği



Alt Sistem İşbirliği Modeli Örnekleri



Alt Sistem İşbirliği İçin Olay Akış Diyagramı Örneği

Nesne Tasarımları-Nesne Tanımı

Tasarımda bir nesnenin tanımı için iki öge vardır:

Protokol Tanımı

Nesnenin alabileceği her ileti ve sonucunda yapması gereken işlemin tanımlanması ile oluşan nesne arayüzüdür.

Uygulama Tanımı

İleti sonucunda başlatılacak her nesne işlemi için süreç belirtimi yapılması ve nesne içi özel veri yapılarının (nesne özelliklerinin) ayrıntılarının belirtimidir.

Burada daha önce karşılaşılan 'ne' ve 'nasıl' soruları yine geçerlidir. Protokol tanımı, '**ne**' sorusuna, uygulama tanımı ise '**nasıl**' sorusuna cevap verir.

Nesne Tasarımları- Yordam ve Veri Yapısı Tasarımı

- Veri yapıları ve yordamlar birlikte tasarlanırlar.
- Sistem düzeyinde yordam, bir bütünlük içerisindeki işlemi gerçekleştiren süreç olarak tanımlanabilir.
- Ancak alt sistem ve nesneler düzeyinde yordamlar biraz daha farklı ele alınırlar.
- Üst düzeydeki işlemlerin daha küçük işlemlere ayrıştırılarak gerçekleştirilmeleri söz konusu olur.
- Burada yine Rumbaugh'dan alınan eniyileme önerilerinden söz etmek yerinde olur:
 - Nesne ilişki modelini verimlilik ışığında gözden geçirin, gerekirse tekrarlanan yapılar tanımlayın.
 - Nesne içi veri yapılarını ve işlem yordamlarını yine verimlilik ışığında gözden geçirin
 - Türetilen bilgiyi saklamak üzere ve fazla işlemi önlemek için gerekiyorsa ek özellikler tanımlayın.

Tasarım Kalıpları

- Diğer alanlarda da tasarımcılar 'kalıp' fikrini başarı ile uygulamaktadırlar.
- Örneğin ev mimarisinde sıkça kullanılacak bir kapı ve belirli bir mesafeden sonra bir pencere kümesi, bir kalıp olarak tanımlanabilir ve her oda tasarımında bu kalıp kullanılmaya çalışılabilir. Böylece yeniden kullanılabilirlik arttırılmış olur.
- Yerleşmiş bir kalıp kullanımı için kalıplar, dört özellik ile tanımlanabilir:
 - Kalıbın ismi
 - Genelde kalıbın uygulandığı problem türü
 - Kalıbın karakteristiği (tasarımda hangi özelliklerin değiştirilmesi ile kalıbın değişik problemlere uyarlanacağı)
 - Kalıbın uygulanmasının etkileri
- NY tasarımda kalıpları oluşturmak için iki mekanizma yararlıdır.
- Kalıtım yolu ile bir kalıp tanımlandıktan sonra değişik problemlere uygulanabilir.
- Diğer mekanizma ise nesneleri bir araya getirerek oluşturulacak 'içerim'dir.
- Karmaşık bir işlemi (alt sisteme karşı düşebilir) bir nesneler bütünü ile uygulama yöntemidir.
- Seçenek olduğu taktirde içerim, kalıtıma karşı tercih edilmektedir.

Özet

- NY kavramların temeli olarak, aşağıdaki dört özelliği sıralayabiliriz:
 - Kimlik (Identity)
 - Sınıflama (Classification)
 - Kalıtım (inheritance)
 - Çok Şekillilik/Biçimlilik(Polimorphism)
- Kalıtım: Hiyerarşik ilişkiye dayanan sınıflar arasında özellik ve işlevlerin paylaşılmasıdır.
- Çok şekillilik: Aynı işlem, farklı sınıflarda değişik davranabilir mantığına dayanır.
- Nesne yönelimli sistemleri destekleyen kavramlar;
 - Soyutlama (Abstraction),
 - Bilgi Gizleme (Encapsulation),
 - Paylaşım (Sharing),
 - Altsınıf (Subclass) ve "Aggregation" dır.
- Uygulama Alanının Çözümlemesinde Amaç, uygulama alanını anlamak ve elde edilen bilgileri analiz modeline taşımaktır.

Özet

Gereksinim Belirleme Çalışmalarında iki teknik vardır;

- Kullanım Durumları ve
- Sınıf Sorumluluk İşbirlikçi modelleridir.

Nesneye Yönelik Tasarımda Katmanlar şu şekildedir:

- Sorumluluk Tasarımı
- İleti Tasarımı
- Sınıf ve Nesne Tasarımı
- Alt Sistem Tasarımı

- NY tasarım metodolojilerinin karşılaştırılmasında kullanılmak üzere 10 özellikten söz edilebilir:
- Bir tasarım metodunun modülerlik ve ilişkili NY kavramlarını desteklemesi için beş kriter ortaya atılmıştır.
- Nesne Yönelimli metodolojilerinin tasarım yöntemleri; Booch Metodu, Coad ve Yourdon Metodu, Jacobson (OOSE) Metodu ve Rumbaugh (OMT) Metodu'dur.

Sorular

1. Nesneye Yönelik Kavramlar nelerdir? Açıklayınız.
2. Kalıtım ve Çoklu Kalıtım arasındaki farkı açıklayınız.
3. Nesne Yönelimli Sistemleri Destekleyen Kavramlar nelerdir? Açıklayınız
4. Sınıf/Sorumluluk/İşbirlikçi Modelini açıklayınız.
5. Alt Sistem İşbirliğini açıklayınız.
6. Nesne Yönelimli metodolojilerinin tasarım yöntemleri kaç tanedir?
Bunları açıklayınız.
7. Tasarımda Nesne kavramı nedir?

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

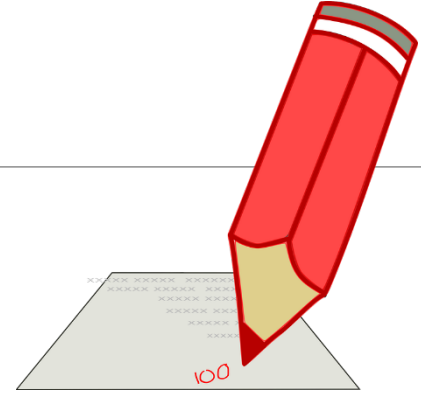
Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları
(<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Ödev

Mimari Tasarım Hakkında Araştırma Yapınız.
Yazılım Mimarisinde Kullanılan Stilleri Araştırınız.



Sorularınız





YMH311-Yazılım Tasarım Ve Mimarisi Gerçekleştirme/Kodlama

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Yazılım Geliştirme Ortamları.....	6
Veri Modelleri.....	17
Kodlama Stili	26
Program Karmaşıklığı.....	33
Olağan Dışı Durum Çözümleme.....	37
Kod Gözden Geçirme.....	38

Amaçlar

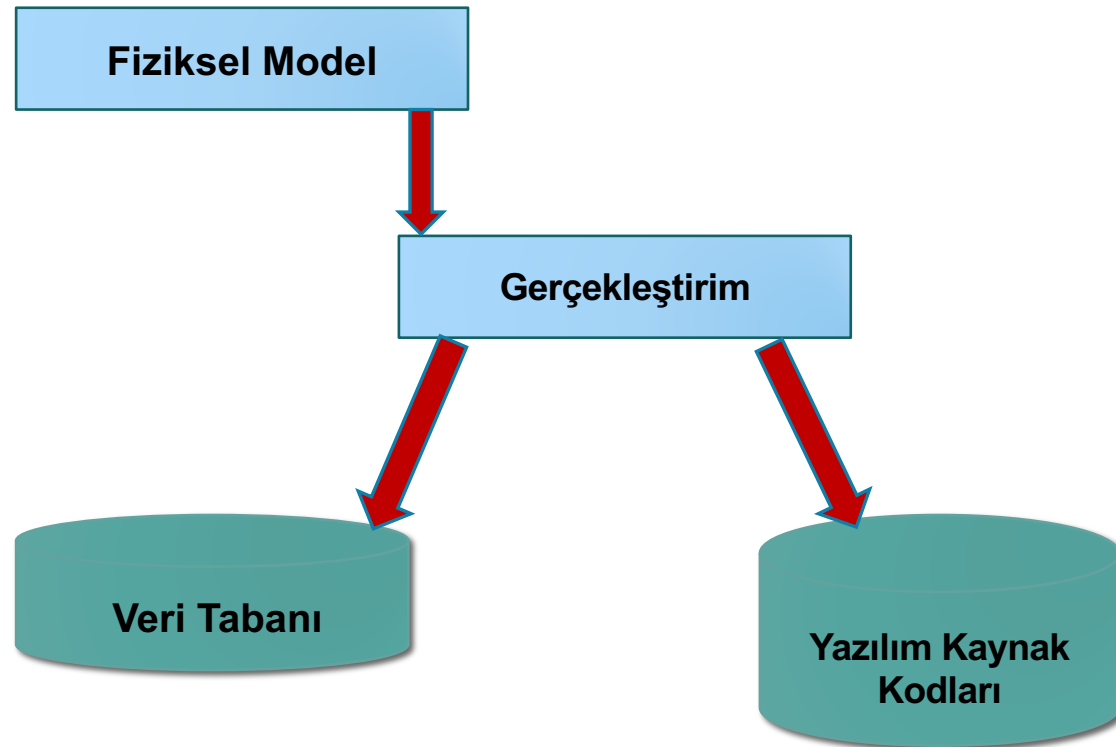
- Yazılım Gerçekleştirimini Kavramak
- Yazılım Geliştirme Ortamları Hakkında Bilgilenmek
- Veri Tabanı Sistemlerini Anlamak
- Veri Modelleme Yöntemlerini Öğrenmek
- Kodlama Stilleri Hakkında Bilgi Edinmek
- Yapısal Programlama Yapılarını Öğrenmek
- Program Karmaşıklığı Hesaplamasında Kullanılan Yöntemler



Giriş

- Tasarım sonucu üretilen süreç ve veri tabanının fiziksel yapısını içeren fiziksel modelin bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışmasıdır.
- Her şeyden önce bir yazılım geliştirme ortamı seçilmelidir (programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları (CASE)).
- Kaynak kodların belirli bir standartta üretilmesi düzeltme için faydalıdır.

Gerçekleştirim Çalışması



Yazılım Geliştirme Ortamları

- Yazılım geliştirme ortamı, tasarım sonunda üretilen fiziksel modelin, bilgisayar ortamında çalıştırılabilmesi için gerekli olan:

Programlama Dili

Veri Tabanı Yönetim Sistemi

Hazır Program Kitapçıkları

CASE Araçları

bileşenlerinden oluşur.

- Günümüzde söz konusu bileşenler oldukça farklılık ve çeşitlilik göstermekte ve teknolojinin değişimine uygun olarak gelişmektedir. Bu bileşenler aşağıda açıklanmaktadır.

Programlama Dilleri

- Geliştirilecek Uygulamaya Göre Programlama Dilleri seçilmelidir.

- **Veri İşleme Yoğunluklu Uygulamalar**

- Cobol,
- Görsel Programlama Dilleri ve Veri Tabanları

- **Hesaplama Yoğun Uygulamalar**

- Fortran
- C
- Paralel Fortran ve C

- **Süreç ağırlıklı uygulamalar**

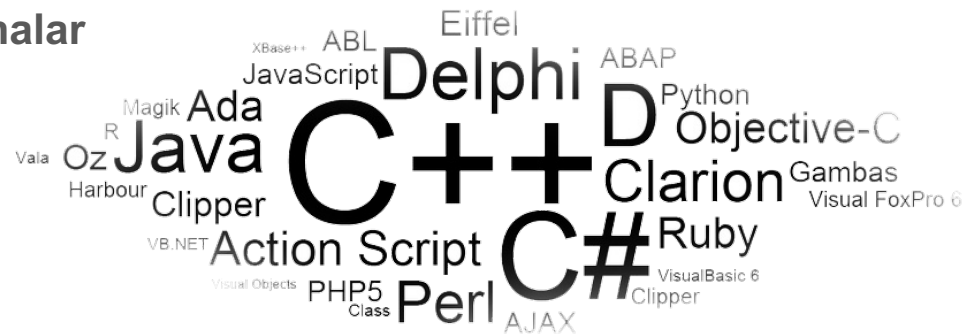
- Assembly
- C

- **Sistem programlamaya yönelik uygulamalar**

- C

- **Yapay Zeka Uygulamaları**

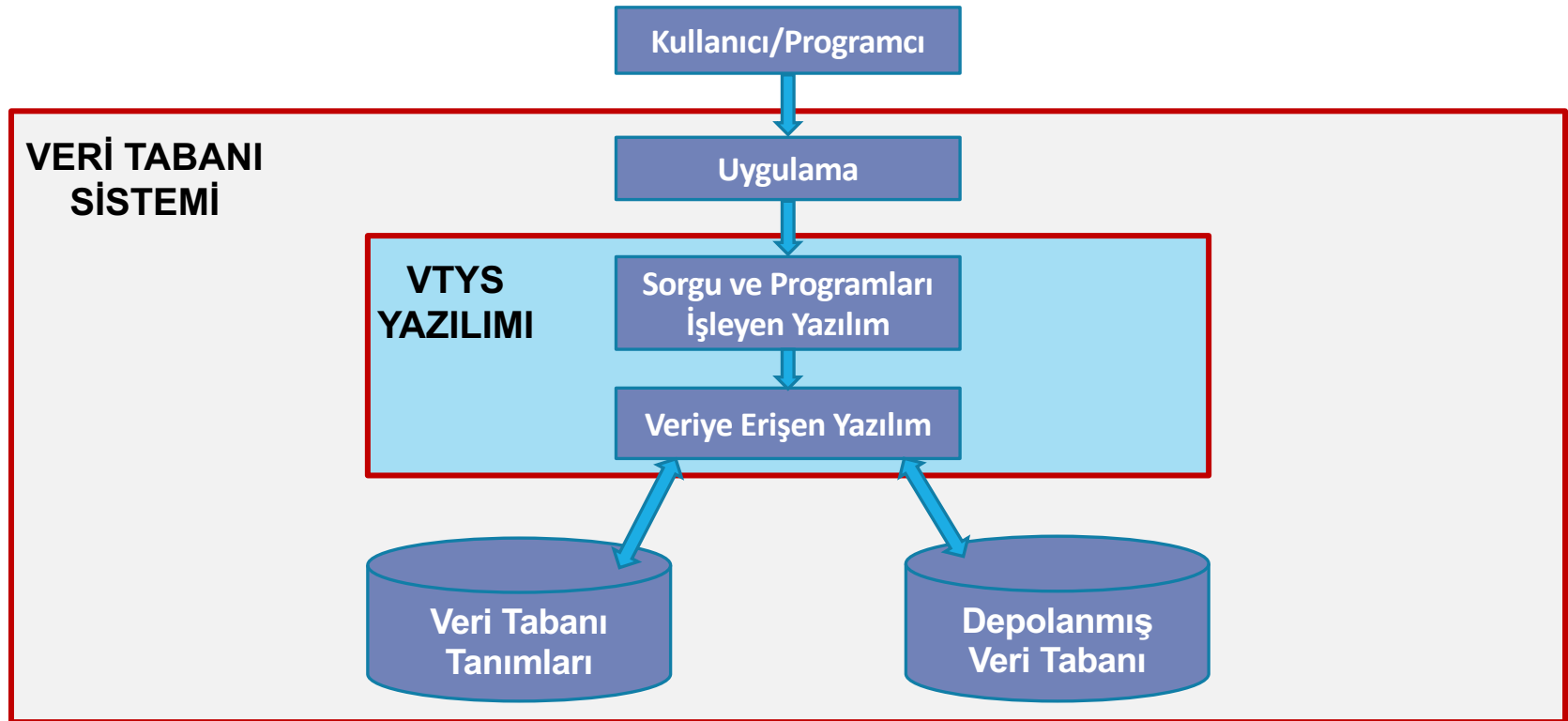
- Lisp
- Prolog



Veri Tabanı Yönetim Sistemleri

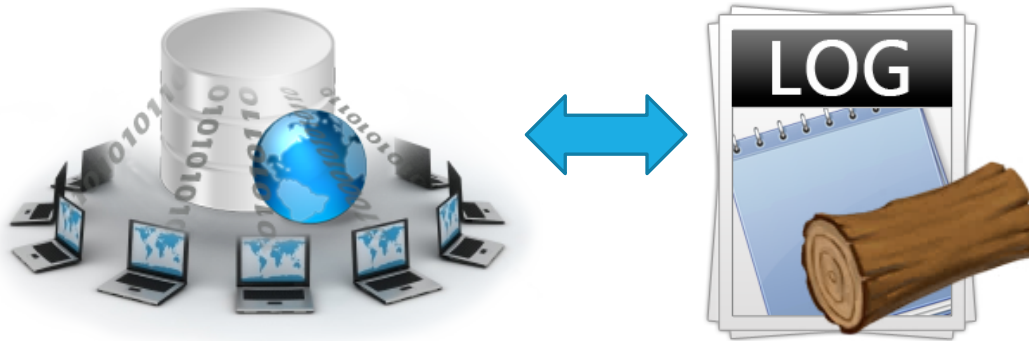
- Birbiri ile ilişkili veriler topluluğu veri tabanı olarak tanımlanmaktadır. Veri tabanı herhangi bir boyutta ya da karmaşıklıkta olabilir.
- Kişisel telefon rehberinizdeki adres bilgileri bir veri tabanı örneği oluşturduğu gibi, maliye bakanlığı bünyesinde saklanan ve vergi ödemesi gereken tüm kişilerin bilgilerinin saklandığı uygulama da bir başka veri tabanı örneğidir.
- Veri tabanını oluşturan veriler birbiriyle ilişkili verilerdir. Bir veritabanında veriler arası ilişkiler ile veri değerleri bulunur.
- Kullanıcıların veritabanındaki verileri soruşturmasını, veritabanına yeni veriler eklemesini, varolan verilerde değişiklik yapmasını sağlayan yazılım Veri Tabanı Yönetim Sistemi (VTYS) olarak tanımlanır. VTYS, genel amaçlı bir yazılımdır.
- Bütün VTYS yazılımları (Oracle, Informix, Sybase vb), kullanıcıya, veri tabanı yapısı tanımlama, veri tabanını sorgulama, değiştirme ve raporlama olanakları verirler.

Veri Tabanı Sistemleri



Veri tabanı Yaklaşımının Geleneksel Kütük Kavramından Ayıran Özellikler

1. Veri Sözlüğü
2. Veri Soyutlama
3. Program-Veri ve Program-İşlem Bağımsızlığı
4. Birden Çok Kullanıcı Desteği
5. Verinin Birden Fazla İşlem Arasında Paylaşımı



Veri Sözlüğü

- Uygulama yazılımında kullanılan tüm veri tanımlarının yapısal ve ayrık bir biçimde saklanması sağlayan bir katalog bilgisi veya veri sözlüğünün varlığıdır.



fields in Student file

Field Name	Data Type	Description
Student ID	AutoNumber	Student's ID Number
First Name	Text	Student's First Name
Last Name	Text	Student's Last Name
Address	Text	Student's Address
City	Text	City Student Lives
State	Text	State Student Lives
Postal Code	Text	Student's Postal Code
E-mail Address	Hyperlink	Student's E-mail
Date Admitted	Date/Time	Date Student Admitted to School
Major	Text	Student's Major Code
Photo	Attachment	Digital Photo of Student

primary key

field name

data type for State field

default value

metadata about State field

Field Properties

General

Field Size: 2

Format:

Input Mask:

Caption:

Default Value: "IN"

Validation Rule:

Validation Text:

Required: Yes

Allow Zero Length: No

Indexed: No

Unicode Compression: No

IME Mode: No Control

IME Sentence Mode: None

Smart Tags:

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

Veri Soyutlama

- Veri tabanı yaklaşımının bir temel karakteristiği kullanıcıdan verinin saklanması konusundaki detayları gizleyerek veri soyutlamasını sağlamasıdır. Bu soyutlamayı sağlayan ana araç veri modelidir. Veri modeli şu şekilde tanımlanabilir.



Bir veri modeli bir veri tabanının yapısını açıklamakta kullanılan kavramlar setidir.

- Bir veri tabanının yapısı ile veri tipleri, ilişkiler ve sınırlamalar kastedilmektedir.
- Pek çok veri modeli ayrıca veri tabanı üzerinde belirtilen getiri (çağrı) ve güncellemeler için temel işlemler setini içerir.
- Davranışı belirlemek üzere veri modelinin içine kavramları da eklemek mümkündür.
- Bu da, temel işlemlere ek olarak kullanıcı tarafından tanımlanmış işlemlerin de veri modeline eklenmesiyle mümkündür.
- Bir veri modelindeki genel işlemlere örnek olarak Ekle, Sil, Değiştir, Eriş verilebilir

Program-Veri ve Program-İşlem Bağımsızlığı

- Geleneksel dosya işleme yönteminde, veri dosyalarının yapısı bu dosyalara erişim programları içine gömülmüştür.
- Bundan dolayı da dosyanın yapısındaki herhangi bir değişiklik bu dosyaya erişen tüm programlarda değişiklik yapılmasını gerektirir.
- Tersine VTYS erişim programları veri dosyalarından bağımsız olarak tasarlanmışlardır. VTYS de saklanan Veri dosyalarının yapısı erişim programlarından ayrı olarak katalogda tutulduğundan program ve veri bağımsızlığı sağlanır.
- Örneğin bir veri dosyasına yeni bir alan eklenmek istendiğinde, bu veri dosyasını kullanan tüm programlara bu alanın eklenmesi gerekir. VTYS yaklaşımında ise sadece katalogdaki veritabanı tanımlarına bu alan eklenerek sorun çözülebilir.
- Nesne-kökenli veritabanları ve programlama dillerindeki son gelişmeler kullanıcının veri üzerindeki işlemleri veritabanı tanımının bir parçası olarak tanımlamasına olanak tanımaktadır.

Birden Çok Kullanıcı Desteği

- Bir veritabanının birçok kullanıcısı vardır ve bunların herbiri veritabanının farklı bir görüntüsüne gereksinim duyabilir.
- Bir görüntü veritabanının alt kümesi olabilir veya veritabanından elde edilmiş fakat kesin olarak saklanmamış sanal veri içerebilir.
- Bazı kullanıcılar, kullandıkları verilerinin veritabanından türetilmiş veriler mi? Yoksa veritabanında gerçekte saklanan veriler mi olup olmadığı ile ilgilenmezler.
- Kullanıcıları farklı uygulamalara sahip çok kullanıcılı bir VTYS çoklu görüntü tanımlama olanaklarını sağlamalıdır.

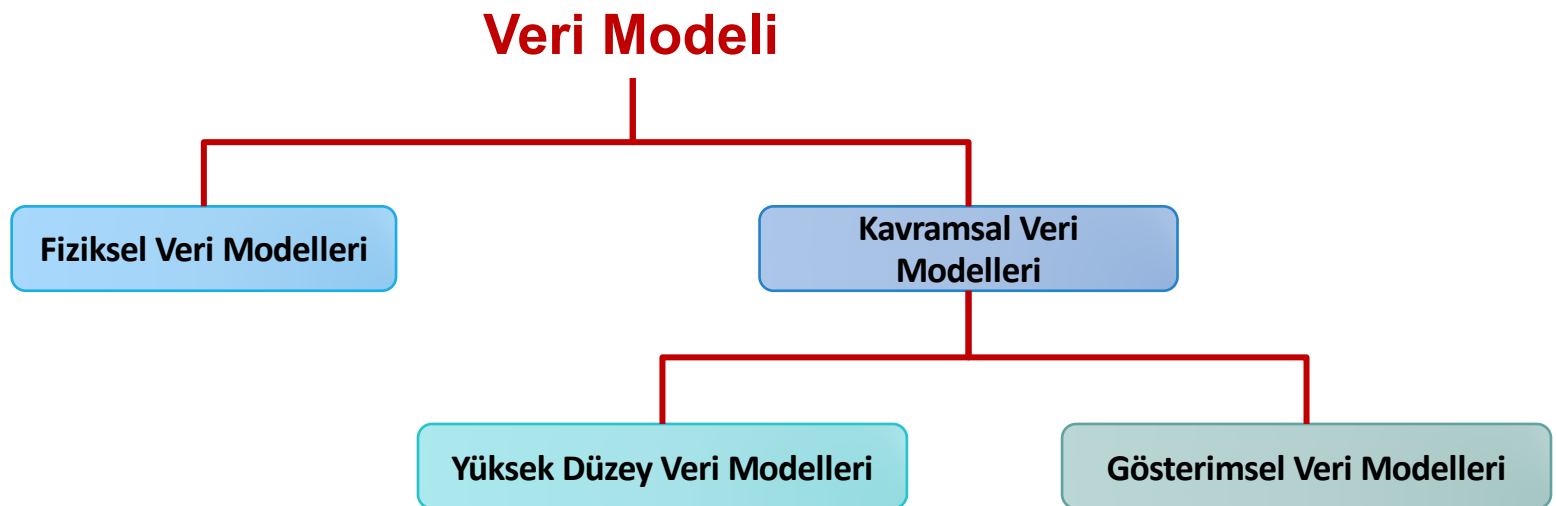
Verinin Birden Fazla İşlem Arasında Paylaşımı

- Çok kullanıcılı VTYS yazılımlarının temel rolü eş zamanlı işlemlerin karışıklık olmadan doğru bir şekilde yapılmasına olanak tanımak işlemlerin doğru olarak yapılmasını garanti etmektir.
- Bu özellik veri tabanı (VT) kavramını, dosya işleme kavramından ayıran en önemli özelliktir.
- Bu kontrol aynı anda birden çok kullanıcının aynı veriyi güncellemeye çalışmasını, güncellemenin doğru olması açısından garanti eder.

VTYS Kullanımının Ek Yararları

1. Genişleme Potansiyeli,
2. Esneklik
3. Uygun geliştirme zamanının azalması,
4. Güncel bilgilerin tüm kullanıcılara aynı zamanda ulaşması,
5. Ölçümde ekonomi,
6. İşletme ortamındaki ortak verilerin tekrarının önlenmesi verilerin merkezi denetimin ve tutarlılığın sağlanması,
7. Fiziksel yapı ve erişim yönetimi karmaşıklıklarının her kullanıcıya yalnız ilgilendiği verilerin kolay anlaşılır yapılarda sunulması,
8. Uygulama yazılımı geliştirmenin kolaylaşması,
9. Fiziksel yapı ve erişim yöntemi karmaşıklıklarının her kullanıcıya yalnız ilgilendiği verilerin kolay anlaşılır yapıda sunulması.

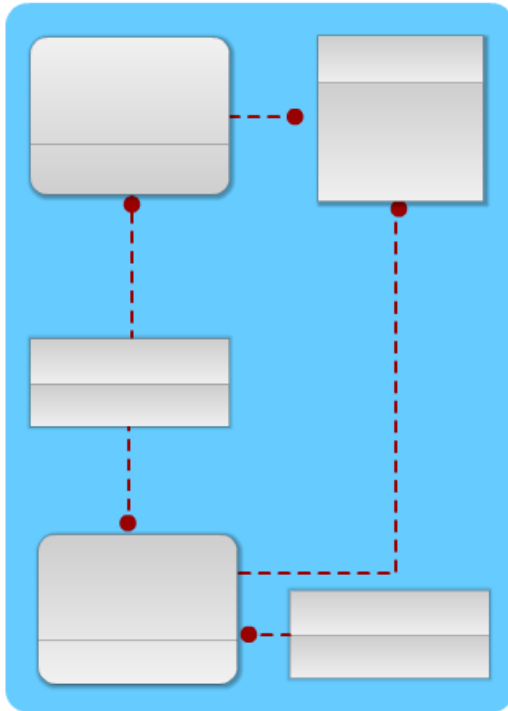
Veri Modelleri



Veri Modelleri

- **Fiziksel veri modelleri** verinin bilgisayarda nasıl saklandığının detayları ile ilgili kavramları sağlar.
- **Kavramsal veri modelleri** hem kullanıcı tarafından anlaşılan hem de verinin bilgisayar içerisindeki gösteriminden çok da fazla uzak olmayan kavramları sağlar.
 - **Yüksek Düzey Veri Modelleri**
 - **Varlık:** Veri tabanında saklanan, gerçek dünyadan bir nesne veya kavramdır (proje, işçi).
 - **Özellik:** Varlığı anlatan bir özelliği gösterir (işçinin adı, ücreti).
 - **İlişki:** Birden fazla varlık arasındaki ilişkidir (işçi ve proje arasındaki çalışma ilişkisi).
 - **Gösterimsel Veri Modelleri**, ticari veri tabanlarında sıklıkla kullanılır ve belli başlı veri üç modeli içerir. İlişkisel, ağ ve hiyerarşik veri modeli. Nesne yönelimli veri modelleri daha yüksek seviyeli gerçekleştirim veri modelleridir ve kavram veri modeline daha yakındır.

Şemalar



- Herhangi bir veri modelinde veri tabanının tanımlaması ile kendisini ayırmak önemlidir.
- Veri tabanının tanımlanması, veri tabanı şeması veya meta-veri olarak adlandırılır.
- Veri tabanı şeması tasarım sırasında belirlenir ve fazla değişmesi beklenmez.

VTYS Mimarisi

- Bu mimarinin amacı kullanıcı uygulamaları ile fiziksel veri tabanını birbirinden ayırmaktır. Bu mimaride şemalar 3 seviyede tanımlanabilir. Bu nedenle üç şema mimarisi olarak da anılır.

İçsel Düzey

- Veri tabanının fiziksel saklama yapısını açıklar.

Kavramsal Düzey

- Kavramsal şema içerir ve kullanıcılar için veri tabanının yapısını açıklar.

Dışsal Düzey

- Dış şemalar ve kullanıcı görüşlerini içerir.

Veritabanı Dilleri ve Arabirimleri

- Veri tabanı tasarımı tamamlandıktan sonra bir VTYS seçilir. Seçilen VTYS'de bulunan dil olanakları aşağıda verilmiştir.

Veri Tanımlama Dili (VTD)	Kavramsal şemaları tanımlamak üzere kullanılır.
Saklama Tanımlama Dili (STD)	İçsel şemayı tanımlamak üzere kullanılır.
Görüş Tanımlama Dili (GTD)	Dışsal şemayı tanımlamak için kullanılır.
Veri İşleme Dili (ViD)	Veri tabanı oluşturulduktan sonra veri eklemek, değiştirmek ve silmek için kullanılır.

Veritabanı Dilleri ve Arabirimleri

- İki tip VİD vardır; yüksek düzeyli ve düşük düzeyli.
 - **Yüksek düzeyli VİD**, bir bilgisayar terminalinden etkileşimli olarak kullanılabildiği gibi bir programlama dili içerisine de yerleştirilebilir.
 - **Düşük düzeyli VİD** de ise VİD bir programlama dili içerisine gömülü olarak çalışır.
- VTYS Arabirimleri olarak Menü-Tabanlı, grafiksel, form tabanlı ve doğal dil arabirimleri kullanılmaktadır.
- Menü tabanlı arabirimlerde kullanıcıya çeşitli seçenekler sunulurken, grafiksel arabirimde kullanıcıya veritabanı şeması diyagramı halinde sunulur.
- Kullanıcı bu diyagram yardımıyla sorgu belirtebilir.
- Form tabanlı arabirimler ise kullanıcıya doldurulmak üzere bir form sunarlar. Doğal dil arabirimleri İngilizce yazılan sorguları kabul eder ve anlamaya çalışırlar.

VTYS nin Sınıflandırılması

Sınıflandırmalar kullanılan Veri Modeline göre yapılır.

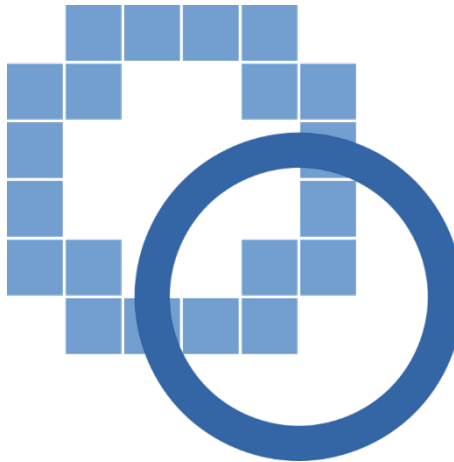
- **İlişkisel Model:** Veri tabanı tablo yığınınından oluşmuştur. Her bir tablo bir dosya olarak saklanabilir.
- **Ağ Modeli:** Veriyi kayıt ve küme tipleri olarak gösterir.
- **Hiyerarşik Model:** Veri ağaç yapısında gösterilir.
- **Nesne Yönelimli Model:** Veri tabanı nesneler, özellikleri ve işlemleri biçiminde gösterilir.

Hazır Program Kütüphaneleri

- Hemen hemen tüm programlama platformlarının kendilerine özgü hazır kütüphaneleri bulunmaktadır.
 - Pascal *.tpu
 - C *.h
 - Java *.jar
- Günümüzde bu kütüphanelerin temin edilmesi internet üzerinden oldukça kolaydır.

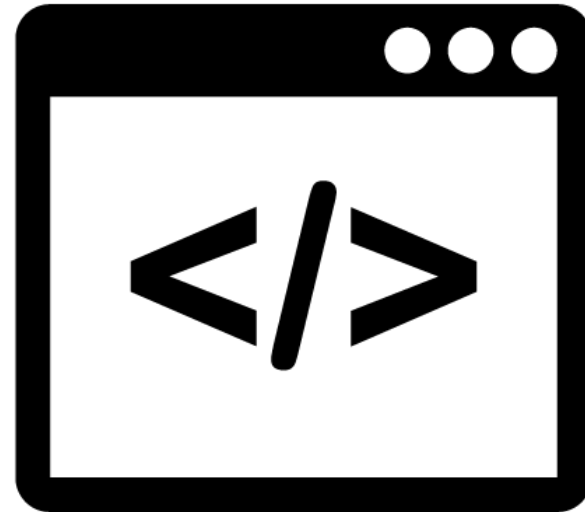
CASE Araç ve Ortamları

- Günümüzde bilgisayar destekli yazılım geliştirme ortamları oldukça gelişmiştir.
- CASE araçları yazılım geliştirme sürecinin her aşamasında üretilen bilgi ya da belgelerin bilgisayar ortamında saklanmasına ve bu yolla kolay erişilebilir ve yönetilebilir olmasına olanak sağlar.



Kodlama Stili

- Hangi platformda geliştirilirse geliştirilsin yazılımın belirli bir düzende kodlanması, yazılımın yaşam döngüsü açısından önem kazanmaktadır.
- Etkin kod yazılım stili için kullanılan yöntemler:
 - Açıklama Satırları
 - Kod Yazım Düzeni
 - Anlamlı İsimlendirme
 - Yapısal Programlama Yapıları



Açıklama Satırları

İYİ:

```
// Satış fiyatı hesaplaması 25.10.2011 İrfan Mevsim
//
// Fonksiyon birim fiyatı ve KDV oranı olarak gönderilen
// bilgilerin satış fiyatını geri döndürür.
//
// Örnek Kullanım
// Dim SatisFiyati as Double = SatisHesapla(1000,18)
//
// Dönüş Değeri --> 1018.0 (double)
//
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
    return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

KÖTÜ:

```
// Satış fiyatı hesaplaması (yetersiz açıklama)
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
    return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

KÖTÜ:

```
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
    // Dönüş Değeri --> 1018.0 (yanlış ve yetersiz açıklama)
    return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

- Modülün başlangıcına
 - genel amaç,
 - işlevi
 - desteklenen platformlar,
 - eksikler,
 - düzeltilen yanlışlıklar

gibi genel bilgilendirici açıklamaları yapılır.
- Aynı zamanda modülün kritik bölümleri vurgulanarak açıklanır.

Kod Biçimlemesi

İyi:

```
string[] test = { "bir", "iki", "üç" };
for ( int i = 0; i < test.length; i++ )
{
    MessageBox.Show(test[i]);
}
```

Kötü:

```
int i;
string[] test = { "bir", "iki", "üç" };
for(i;i<test.length;i++)
    MessageBox.Show(test[i]);
```

İyi:

```
string test = "bir,iki,uc";
foreach ( string item in test.Split(',') )
{
    MessageBox.Show(item);
}
```

Kötü:

```
string test = "bir,iki,uc";
string item;
foreach (item in test.split(','))
    MessageBox.Show(item);
```

- Programın okunabilirliğini artırmak ve anlaşılabilirliğini kolaylaştırmak amacıyla açıklama satırlarının kullanımının yanı sıra, belirli bir kod yazım düzeninin de kullanılması gerekmektedir.

Anlamlı İsimlendirme

iyi:

```
KodlamaStandartlari.csproj
namespace KodlamaStandartlari

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Models

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Models
```

~~kötü:~~

```
KodlamaStandartlari.csproj
namespace Kodlama

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Modeller

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Modeller
```

“İsim Alanı” İsimlendirme

- Kullanılan tanımlayıcıların (değişken adları, dosya adları, veri tabanı tablo adları, fonksiyon adları, yordam adları gibi) anlamlı olarak isimlendirilmeleri anlaşılabilirliği büyük ölçüde etkilemektedir.

Yapısal Programlama Yapıları

- Program kodlarının, okunabilirlik, anlaşılabilirlik, bakım kolaylığı gibi kalite etmenlerinin sağlanması ve program karmaşıklığının azaltılması amacıyla "yapısal programlama yapıları" kullanılarak yazılması önemlidir.
- Yapısal Programlama Yapıları, temelde, içinde "go to" deyimi bulunmayan, "tek giriş ve tek çıkışlı" öbeklerden oluşan yapılardır.
- Teorik olarak herhangi bir bilgisayar programının, yalnızca Yapısal Programlama Yapıları kullanılarak yazılabileceği kanıtlanmıştır.
- Üç temel Yapısal Programlama Yapısı bulunmaktadır:
 - Ardışıl işlem yapıları
 - Koşullu işlem yapıları
 - Döngü yapıları

Yapısal Programlama Yapıları

Ardışıl İşlem Yapıları

- Ardışıl işlemler, herhangi bir koşula bağlı olmaksızın birbiri ardına uygulanması gereken işlemler olarak tanımlanır. Hemen her tür programlama dilinde bulunan, aritmetik işlem deyimleri, okuma/yazma deyimleri bu tür yapılara örnek olarak verilebilir.

Koşullu İşlem Yapıları

- Üç tür Koşullu işlem yapısı bulunmaktadır: tek koşullu işlem yapısı (if-then), iki koşullu işlem yapısı (if-then-else) ve çok koşullu işlem yapısı (case-when). 70'li yılların ortalarından sonra gelen programlama dillerinin hemen hepsinde, bu yapılar doğrudan desteklenmektedir.

Yapısal Programlama Yapıları

Döngü Yapıları

- Döngü yapıları, belirli bir koşula bağlı olarak ya da belirli sayıda, bir ya da dah çok kez yinelenen işlemler için kullanılan yapılardır. Temelde üç tür döngü yapısı bulunmaktadır. Bunlar:
 1. Belirli sayıda yinelenen işlemler için kullanılan yapılar (for yapısı),
 2. Bir koşula bağlı olarak, sıfır yada birden çok kez yinelenen işlemler için kullanılan yapılar (while-end yapısı),
 3. Bir koşula bağlı olarak, bir ya da daha çok kez yinelenen işlemler için kullanılan yapılar (repeat-until yapısı).

Bu yapıların her biri 'tek girişli ve tek çıkışlı' yapılardır.

Program Karmaşıklığı

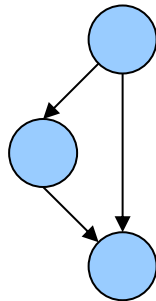
- Program karmaşıklığı konusunda çeşitli modeller geliştirilmiştir.
- Bunların en eskisi ve yol göstericisi McCabe tarafından 1976 yılında geliştirilen modeldir.
- Bunun için önce programın akış diyagramına dönüştürülmesi, sonra da **karmaşıklık ölçütünün** hesaplanması gerekmektedir.
- McCabe ölçütü, bir programda kullanılan "koşul" deyimlerinin program karmaşıklığını etkileyen en önemli unsur olduğu esasına dayanır ve iki aşamada uygulanır:
 1. Programın Çizge Biçimine Dönüştürülmesi
 2. Mc Cabe Karmaşıklık Ölçütünün Hesaplanması

Programın Çizge Biçimine Dönüştürülmesi-1

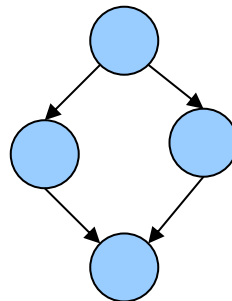
Sıradan İşlemler:



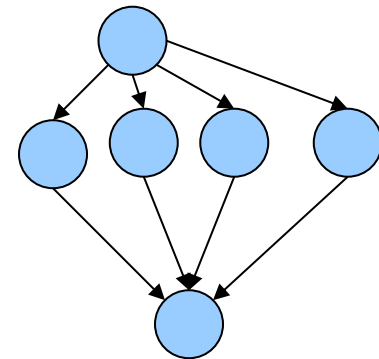
Koşullu İşlemler:



If-Then işlemi



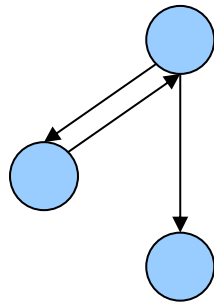
If-Then-Else işlemi



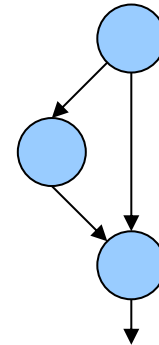
Case işlemi

Programın Çizge Biçimine Dönüştürülmesi-2

Döngü İşlemleri:



While Döngüsü



Repeat Döngüsü

McCabe Karmaşıklık Ölçütü

$$V(G) = k - d + 2p$$

K: Diyagramdaki kenar çizgi sayısı

D: Diyagramdaki düğüm sayısı

P: programdaki bileşen sayısı (ana program ve ilgili alt program sayısını göstermektedir. Alt program kullanılmadı ise $p=1$, 3 alt program kullanıldı ise $p=4$ tür)

Olağandışı Durum Çözümleme

- **Olağandışı durum:** Bir programın çalışmasının, geçersiz ya da yanlış veri oluşumu ya da başka nedenlerle istenmeyen bir biçimde sonlanmasına neden olan durumdur.
- Genelde kabul edilen; program işletiminin sonlandırılmasının bütünüyle program denetiminde olmasıdır.



Kod Gözden Geçirme

- Bir gazetede hiç bir yazı editörün onayı alınmadan basılamayacağı gibi, kod gözden geçirme olmadan da yazılım sistemi geliştirilemez.
- Kod gözden geçirme ile program sınaama işlemleri birbirlerinden farklıdır.
- Kod gözden geçirme, programın kaynak kodu üzerinde yapılan bir işlemdir ve bu işlemlerde program hatalarının %3-5'lik bir kısmı yakalanabilmektedir.

Gözden Geçirme Sürecinin Düzenlenmesi

- Gözden geçirme sürecinin temel özellikleri;
 - Hataların bulunması, ancak düzeltilmemesi hedeflenir,
 - Olabildiğince küçük bir grup tarafından yapılmalıdır. En iyi durum deneyimli bir inceleyci kullanılmasıdır. Birden fazla kişi gerektiğinde, bu kişilerin, ileride program bakımı yapacak ekipten seçilmesinde yarar vardır.
 - Kalite çalışmalarının bir parçası olarak ele alınmalı ve sonuçlar düzenli ve belirlenen bir biçimde saklanmalıdır.

biçiminde özetlenebilir. Burada yanıtı aranan temel soru, programın yazıldığı gibi çalışıp çalışmayacağının belirlenmesidir. Gözden Geçirme çalışmasının olası çıktıları:

- ? Programı olduğu gibi kabul etmek
- ? Programı bazı değişikliklerle kabul etmek
- ? Programı, önerilen değişikliklerin yapılmasından sonra tekrar gözden geçirmek üzere geri çevirmek.

Gözden Geçirme Sırasında Kullanılacak Sorular

- Bir program incelenirken, programın her bir öbeği (yordam ya da işlev) aşağıdaki soruların yanıtları aranır.
- Bu sorulara ek sorular eklenebilir.
- Bazı soruların yanıtlarının "hayır" olması programın reddedileceği anlamına gelmemelidir.



Öbek Arayüzü

1. Her öbek tek bir işlevsel amacı yerine getiriyor mu?
2. Öbek adı, işlevini açıklayacak biçimde anlamlı olarak verilmiş mi?
3. Öbek tek giriş ve tek çıkışlı mı?
4. Öbek eğer bir işlev ise, parametrelerinin değerini değiştiriyor mu?

Giriş Açıklamaları

1. Öbek, doğru biçimde giriş açıklama satırları içeriyor mu?
2. Giriş açıklama satırları, öbeğin amacını açıklıyor mu?
3. Giriş açıklama satırları, parametreleri, küresel değişkenleri içeren girdileri ve kütükleri tanıtıyor mu?
4. Giriş açıklama satırları, çıktıları (parametre, kütük vb) ve hata iletilerini tanımlıyor mu?
5. 5. Giriş açıklama satırları, öbeğin algoritma tanımını içeriyor mu?
6. 6. Giriş açıklama satırları, öbekte yapılan değişikliklere ilişkin tanımlamaları içeriyor mu?
7. 7. Giriş açıklama satırları, öbekteki olağan dışı durumları tanımlıyor mu?
8. 8. Giriş açıklama satırları, Öbeği yazan kişi ve yazıldığı tarih ile ilgili bilgileri içeriyor mu?
9. 9. Her paragrafı açıklayan kısa açıklamalar var mı?

Veri Kullanımı

1. İşlevsel olarak ilintili bulunan veri elemanları uygun bir mantıksal veri yapısı içinde gruplanmış mı?
2. Değişken adları,işlevlerini yansıtacak biçimde anlamlı mı?
3. Değişkenlerin kullanımları arasındaki uzaklık anlamlı mı?
4. Her değişken tek bir amaçla mı kullanılıyor?
5. Dizin değişkenleri kullanıldıkları dizinin sınırları içerisinde mi tanımlanmış?
6. Tanımlanan her gösterge değişkeni için bellek ataması yapılmış mı?

Sunuş

1. Her satır, en fazla bir deyim içeriyor mu?
2. Bir deyim birden fazla satıra taşması durumunda, bölünme anlaşılabilirliği kolaylaştıracak biçimde anlamlı mı?
3. Koşullu deyimlerde kullanılan mantıksal işlemler yalın mı?
4. Bütün deyimlerde, karmaşıklığı azaltacak şekilde parantezler kullanılmış mı?
5. Bütün deyimler, belirlenen program stiline uygun olarak yazılmış mı?
6. Öbek yapısı içerisinde akıllı "programlama hileleri" kullanılmış mı?

Sorular

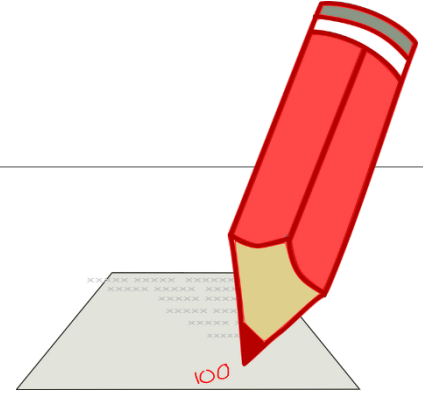
1. Kendi yazılım geliştirme ortamınızı açıklayınız.
2. Veri tabanı ile veri tabanı yönetim sistemi arasındaki farkı belirtiniz.
3. Veri tabanı Yönetim Sistemi kullanarak, uygulama geliştirme zamanının nasıl kısaldığını açıklayınız.
4. Veri tabanı Yönetim Sistemi kullanımının yararlarını ve eksik yönlerini belirtiniz?
5. Kullandığınız bir veri tabanı yönetim sistemini inceleyiniz. VTD, STD, GİD, GTD dillerinin özelliklerini araştırınız.
6. Kendi kullanımınız için kodlama stili geliştiriniz.
7. Kodlama stilleri ile programlama dilleri arasındaki ilişkiyi belirtiniz.
8. Bildiğiniz bir programlama dilinin, yapısal programlama yapılarından hangilerini doğrudan desteklediğini, hangilerini desteklemediğini araştırınız. Desteklenmeyen yapıların, bu programlama dilinde nasıl gerçekleştirilebileceğini belirtiniz.
9. Verilen bir N doğal sayısının asal olup olmadığını belirleyen bir programı dört değişik biçimde yazınız. Programlar arasındaki zaman farklılıklarını ölçünüz.
10. Program geliştirirken kullandığınız olağan dışı durum çözümleme yöntemlerini açıklayınız?
11. Geliştirdiğiniz bir program için, bölüm içerisinde verilen gözden geçirme sorularını yanıtlayınız. Programınızın niteliği hakkında ne söyleyebilirsiniz?

Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- ” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- ”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Ödev

?



Sorularınız





YMT 312-Yazılım Tasarım Ve Mimarisi

Test - Doğrulama ve Geçerleme

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Doğrulama Ve Geçerleme.....5

Sinama Kavramları.....21

Doğrulama ve Geçerleme Yaşam Döngüsü.....27

Sinama Yöntemleri.....29

Sinama ve Bütünleştirme Stratejileri.....34

Sinama Planlaması.....47

Sinama Belirtileri.....48

Yaşam Döngüsü Boyunca Sinama Etkinlikleri.....50

Amaçlar

- Doğrulama ve Geçerleme Kavramlarını Öğrenmek
- Doğrulama Tekniklerini Öğrenmek
- Sınama Kavramları Hakkında Bilgi Edinmek
- Doğrulama ve Geçerleme Yaşam Döngüsünü Kavramak
- Bütünleştirme Sınamasında "koçan" Kullanımı
- Yazılım Yaşam Döngüsü Boyunca Sınama Etkinliklerini Öğrenmek



Giriş

- Geliştirilecek bilgi sistemi yazılımının doğrulanması ve geçerlenmesi, üretim süreci boyunca süren etkinliklerden oluşur. Söz konusu etkinlikler:
 - Yazılım belirtilmelerinin ve proje yaşam sürecindeki her bir etkinlik sonunda alınan çıktıların, tamam, doğru, açık ve önceki belirtilmeleri tutarlı olarak betimler durumunda olduğunun doğrulanması.
 - Proje süresince her bir etkinlik ürününün teknik yeterliliğinin değerlendirilmesi ve uygun çözüm elde edilene kadar aktivitenin tekrarına sebep olması.
 - Projenin bir aşaması süresince geliştirilen anahtar belirtilmelerin önceki belirtilmelerle karşılaştırılması.
 - Yazılım ürünlerinin tüm uygulanabilir gerekleri sağladığının gerçekleşmesi için sınamaların hazırlanıp yürütülmesi.

biçiminde özetlenebilir.

Doğrulama ve Geçerleme

Doğrulama

Doğru ürünü mü üretiyoruz ?

Ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmadığına dair etkinliklerden oluşur.

Geçerleme

Ürünü doğru olarak mı üretiyoruz?

Ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.

Doğrulama ve geçerleme işlemleri temel olarak çeşitli düzeylerde sına, gözden geçirme, denetim ve hata giderme süreçlerinden oluşur.

Bağımsız Doğrulama ve Geçerleme

Tüm D&G işlemleri bir başka şirket tarafından yapılır. Kurum içi başka bir bölüm/grup tarafından da yapılabilir.

Avantajları

- Konusunun uzmanları tarafından yapılır
- Kurum içi siyasetten etkilenmez

Dezavantajları

- Daha pahalıdır
- Daha sıkı koordinasyon gerektirir

Doğrulama Teknikleri

➤ Gözden Geçirme

- Yönetim
- Teknik
- Arkadaş
- Masaüstü

➤ Üstünden Geçme

➤ Denetleme

➤ İnceleme

Gözden Geçirme - Yönetim

Amaç: [IEEE1028-97]

- Durum izlemek
- Planların ve takvimin durumlarını belirlemek
- Gereksinimleri ve sistem kaynaklarını onaylamak
- Yönetim şeklinin hedefe uygunluğunu değerlendirmek

Yöntem:

- Proje Yöneticisi veya Lideri tarafından yöneticilere sunum

Katılımcılar:

- Karar verme yetkisi olan yöneticiler

Neler:

Raporlar (Denetleme, Durum, Sonuçlar, ...)

Planlar (Risk, Proje, Konfigürasyon, ...)

Beklenen Sonuçlar:

Yapılması planlanan değişikliklere ve iyileştirme kararlarına destek ve onay almak

Plan, takvim, ve gereksinimlerin yeterliliğini belirlemek

Projenin yolunda gidip gitmediğini belirlemek

Gözden Geçirme - Teknik

Amaç: [IEEE1028-97]

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek

Yöntem:

- Teknik Lider tarafından sunum

Katılımcılar:

- Karar verme yetkisi olan yöneticiler,
- Gözden geçirme sorumlusu,
- Kaydedici,
- Teknik Uzmanlar

Neler:

Ürüne ait:

- Amaç ve hedefler
- Proje yönetim planı
- Problem listesi

Beklenen Sonuçlar:

- Ürünün beklentilere ve standartlara uygun olup olmadığını yöneticilere göstermek
- Değişiklikleri kontrol etmek,
- Devam edip etmeme kararını almak

Gözden Geçirme - Arkadaş

Amaç:

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek.

Yöntem:

- Ürün konuya hakim bir takım arkadaşına verilir ve bulunan hatalar/düzeltilmeler ürün sahibine açıklanır

Katılımcılar:

- Konusunda uzman takım elemanı
- Ürün sahibi

Neler:

- Gereksinimler, tasarım dokümantasyonu, kaynak kodu
- Planlar (Proje, Yazılım Geliştirme, Test, ...)

Beklenen Sonuçlar:

- Bulunan hatalar/düzeltilmeler/tavsiyeler belgenin üzerine yazılır ve yazara açıklanır

Gözden Geçirme – Masa Üstü

Amaç:

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek.

Yöntem:

- Ürün konuya hakim uzmanlara dağıtılır ve bulunan hatalar/düzeltilmeler ürünün bir kopyası üzerine kaydedilir

Katılımcılar:

- Uzmanlar
- Belgenin müşterileri
- Belgenin sahibi

Neler:

- Gereksinimler, tasarım dokümantasyonu, kaynak kodu
- Planlar (Proje, Yazılım Geliştirme, Test, ...)

Beklenen Sonuçlar:

- Bulunan hatalar/düzeltilmeler/tavsiyeler belgenin üzerine yazılır ve yazara açıklanır

Üstünden Geçme

Amaç: [IEEE1028-97]

- Ara veya son ürünü değerlendirmek,
- Katılımcıları eğitmek

Yöntem:

- Konu ile ilgili proje elemanlarına sunum

Katılımcılar:

- Yazar (Sunucu)
- Konusunda uzmanlar
- Ürünün müşterileri

Neler:

Sistem Gereksinim Belirtileri (SRS)
Tasarım
Proje Planı

Beklenen Sonuçlar:

Bulgu listesi
İyileştirme ve alternatif tavsiyeleri

Denetleme

Amaç: [IEEE1028-97]

- Ürünün ve süreçlerin bağımsız uzmanlar tarafından değerlendirilip regülasyonlara, standartlara, kılavuzlara, planlara ve protokollere uygunluğunun belirlenmesi.

Yöntem:

- Bağımsız bir uzman takımı tarafından değerlendirme

Katılımcılar:

- Bağımsız uzmanlar

Neler:

- Gereksinimler, Sistem Mimarisi, Tasarımlar
- Proje Planı

Beklenen Sonuçlar:

- Bulgu listesi
- İyileştirme ve alternatif tavsiyeleri

İnceleme

Amaç: [IEEE1028-97]

- Üründeki hata ve anormallikleri belirlemek.

Yöntem:

- İnceleme toplantısı düzenlenir
- Katılımcılar toplantıdan önce ürünü detaylı olarak değerlendirir
- Toplantı sırasında ürün satır satır okunarak bulgular tartışılır

Katılımcılar : (Yöneticiler katılmaz)

- İnceleme sorumlusu (Uzlaştırıcı)
- Yazar,
- Okuyucu,
- Kayıtcı,
- İnceleyiciler
 - Tekrar İncelenmesi gerekir

Neler:

Gereksinimler, tasarım ve kaynak kod

Beklenen Sonuçlar:

Hata listesi

Kabul Durumu:

Olduğu gibi veya önemi düşük düzeltmelerle kabul
Önemli değişikliklerle kabul

İncelemenin Önemi

- İnceleme maliyeti artırır. Faydası nedir?
- Sonraki aşamalarda, özellikle Test ve Müşteri tarafından bulunan hataların onarılması maliyeti daha fazla etkiler
- Bu hataları giderirken, başka hataların sisteme girme olasılığı artar
- Test ve Müşteri tarafından hataların düzeltilmesi proje süresini artırır
- Müşterinin ürün ve firma hakkındaki fikirleri zedelenir
- Proje elemanlarının işlerinden tatmini azalır

İnceleme Yapmayan Bir Proje

Proje büyüklüğü aşağıdaki rakamlarla verilmiştir:

- Gereksinimler : 300 sayfa
- Tasarım : 150 sayfa
- Kaynak Kodu : 10 000 satır

Varsayımlar:

- Gereksinim, Tasarım ve Kodlama aşamaları 100'er hata yapıyor
- Tüm hatalar Test ve Müşteri tarafından bulunuyor
- Testin Hata Bulma Etkinliği (HBE) %75 ve geriye kalan hataların hepsi müşteri tarafından bulunuyor
- Test tarafından bulunan hataların onarılması Gereksinim, Tasarım ve Kodlama hataları için sırasıyla 3, 2.5, ve 2 gündür
- Müşteri tarafından bulunan hataların onarılması Gereksinim, Tasarım ve Kodlama hataları için sırasıyla 4, 3.5, ve 3 gündür

İnceleme Yapmayan Bir Proje (Devam)

Aşama	Hata	Tst	Mlyt	Mışt	Mlyt	Top Mlyt
Gerek	100	75	225	25	100	325
Tasar	100	75	188	25	88	275
Kod	100	75	150	25	75	225
Toplam	300	225	563	75	263	825

Tüm Hataların Onarılma Maliyeti

825 adam-gün

Aynı Proje – Gereksinimler İncelenirse

İnceleme için varsayımlar:

- Gereksinim İncelemesi hızı 15 sayfa/saat
- İnceleme Takımı 5 kişi
- İncelemenin HBE %75
- İncelemede bulunan hataların onarılma süresi ½ adam-gün
- İş Günü 8 saat

İncelemenin maliyeti

$$(300/15) \times 5 = 100 \text{ saat} = 12.5 \text{ gün}$$

İncelemede bulunan hataları onarma süresi

$$75 \times 0.5 = 37.5 \text{ gün}$$

Toplam

$$12.5 + 37.5 = \underline{\underline{50 \text{ gün}}}$$

Aynı Proje – Gereksinimler İncelenirse (Devam)

Test ve Müşteri Tarafından Bulunan Hataların Onarılma Maliyeti

Aşama	Hata	Tst	Mlyt	Mşt	Mlyt	Top Mlyt
Gerek	25	19	57	6	24	81
Tasar	100	75	188	25	88	275
Kod	100	75	150	25	75	225
Toplam	225	169	395	56	187	581

Tüm Hataların Toplam Maliyeti

581+50 = 631 adam-gün

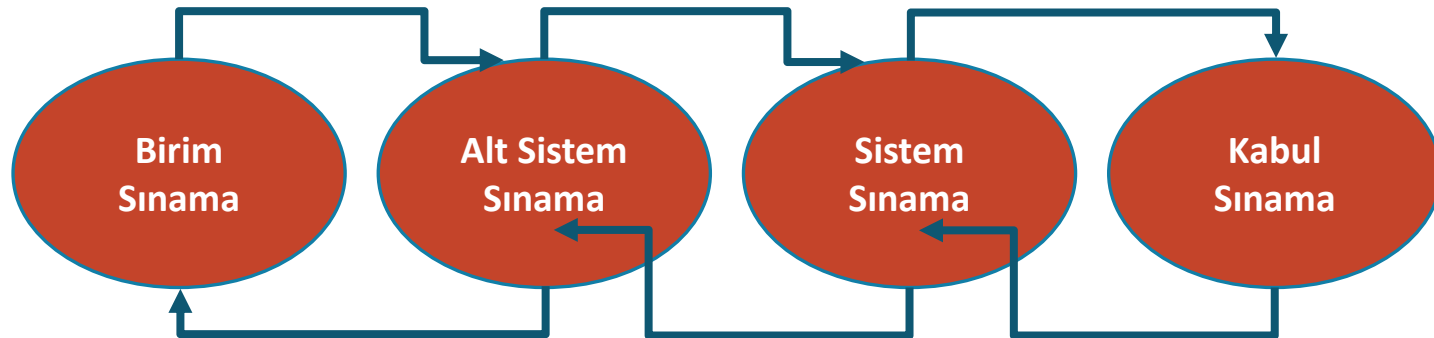
İncelemenin faydaları

- Hataların Onarılması:
 - İncelemesiz : 825 adam-gün
 - İnceleme ile : 631 adam-gün
 - Kazanılan zaman : 194 adam-gün
 - 39 adam-hafta
 - 9 adam-ay
- Maliyet azaldı
- Takvim kısaldı
- Kalite arttı
- Müşteri memnun (Neden?)
- Proje elemanları memnun (Neden?)
- Yöneticiler memnun (Neden?)

Sinama Kavramları

- Sinama ve bütünleştirme işlemlerinin bir strateji içinde gerçekleştirilmesi, planlanması ve tekniklerin seçimi gerekmektedir. Bütünleştirme işlemi, en küçük birimlerden başlanarak sistem düzeyine çıkılmaktadır.
- Bu değişik düzeylere hitap edecek sinama yöntemleri olmalıdır.
- Sınamalar, hatalardan kurtulmanın bir güvencesi değildir.
- Hatalardan bütünüyle arınıldığı gibi bir kanı edinilmemelidir.
- Yalnızca, sınamalar uzadıkça hata bulma sıklığı azalır, daha zor bulunacak hatalar gizli kalmağa devam eder.
- Ne kadar hata sıklığına erişildiğinde sinama işleminin durdurulacağı, maliyet ve kalite arasında bir en iyileştirme yapma gereğini öne çıkarır.
- Aynı zamanda vakit de önemli bir unsurdur.
- Daha uzunca süreler vakitler harcanarak daha az hatalar bulunmaya devam eder.
- Yazılımın kritiklik düzeyine göre, sinamaya ayrılan süre ve çaba artar.

Sinama Kavramları



Birim Sınama

- Bağlı oldukları diğer sistem unsurlarından bütünüyle soyutlanmış olarak birimlerin doğru çalışmalarının belirlenmesi amacıyla yapılır.
- Birimler, ilişkili yapıtaşlarının bütünleştirilmesinden oluşurlar.

Alt Sistem Sınama

- Alt sistemler, modüllerin bütünleşmesi ile ortaya çıkar. Yine bağımsız olarak sınamaları yapılmalıdır.
- Bu düzeyde en çok hata arayüzlerde bulunmaktadır, arayüz hatalarına yönelik sınamalara yoğunlaşılmalıdır.

Sistem Sınama

- Üst düzey bileşenlerin sistem ile olan etkileşimlerinde çıkacak hatalar aranmaktadır.
- Ayrıca belirtilen ihtiyaçların doğru yorumlandıkları da sınanmalıdır.

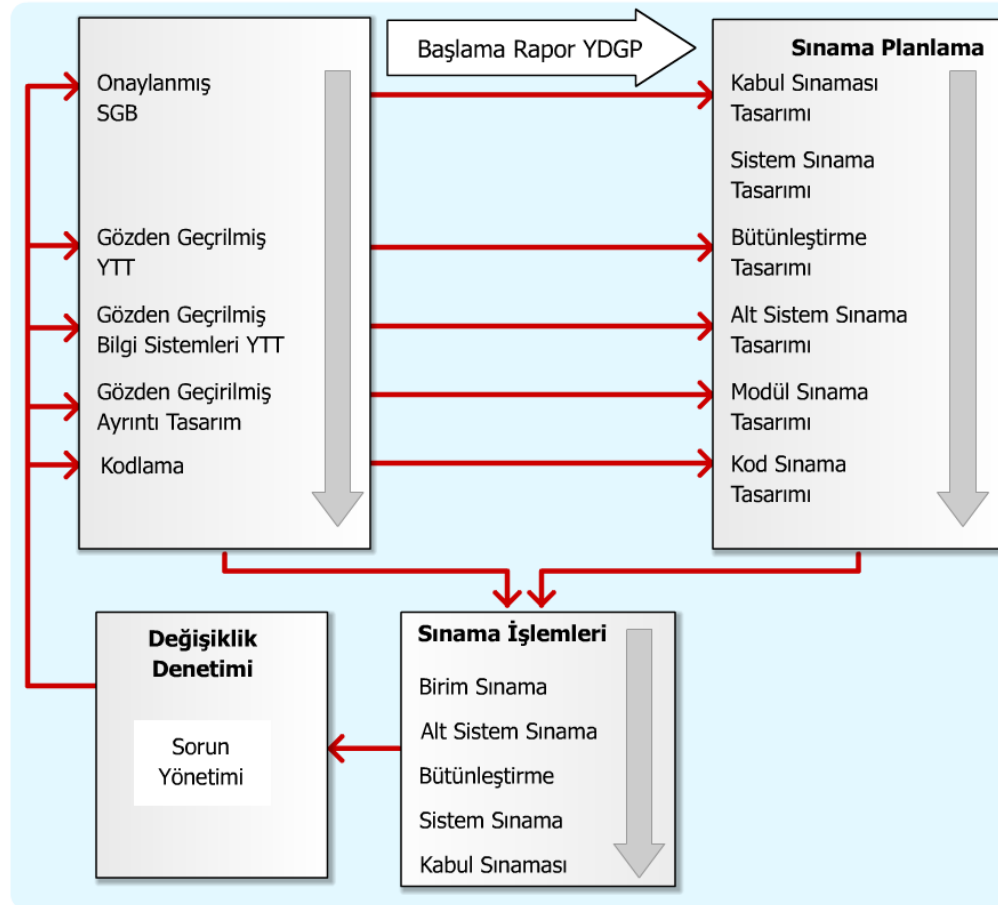
Kabul Sınama

- Çalıştırılmadan önce sistemin son sınavasıdır. Artık yapay veri yerine gerçek veriler kullanılır.
- Bu sinama türü alfa sinaması ve beta sinaması olarak ta bilinir.
- Alfa sinamada, tanımında, sinamanın geliştirici organizasyonun yerleşkesinde, kullanıcıların da gelerek katkıda bulunması içerilir.
- Daha sonra ürünün pazarlama işlemi sırasında beta sinama denilen, sinama kullanıcının kendi yerleşkesinde geliştirici gözetiminde yapılır.

Doğrulama ve Geçerleme Yaşam Döngüsü

- Doğrulama ve geçerleme işlemleri yazılım üretim yaşam döngüsünün tüm süreçlerinde ve bu süreçlere koşul olarak sürer.
- Gerçekleştirim aşamasına kadar olan süreçlerde doğrulama ve geçerleme işlemlerinin planlaması yapılır.
- Planlama, genel olarak, birim, alt sistem, bütünleştirme, sistem ve kabul sınamalarının tasarımı içerir. Gerçekleştirim aşamasının sonunda ise söz konusu planlar uygulanır.
- Uygulama sonucu elde edilen bulgular, Yazılım Doğrulama ve Geçerleme raporları biçiminde sürekli olarak raporlanır.
- Bu bilgiler, değişiklik denetim sistemi ve sorun yönetim sistemlerinde girdi olarak kullanılır.
- Değişiklik Denetim sistemi, sınama süresince elde edilen bulguların izlenmesi amacıyla oluşturulan bir sistemdir.
- Bu sistemde, sınama sonucu elde edilen bulgular ve bunlara karşı sorun yönetimi tarafından alınan önlemler izlenir.

Doğrulama ve Geçerleme Yaşam Döngüsü



Beyaz Kutu Sınaması

Beyaz kutu sınaması tasarlanırken, birimin süreç belirtiminden yararlanır.

Yapılabilecek denetimler arasında:

- Bütün bağımsız yolların en azından bir kere sınanması,
- Bütün mantıksal karar noktalarında iki değişik karar için sınamaların yapılması,
- Bütün döngülerin sınır değerlerinde sınanması,
- İç veri yapılarının denenmesi

bulunur.

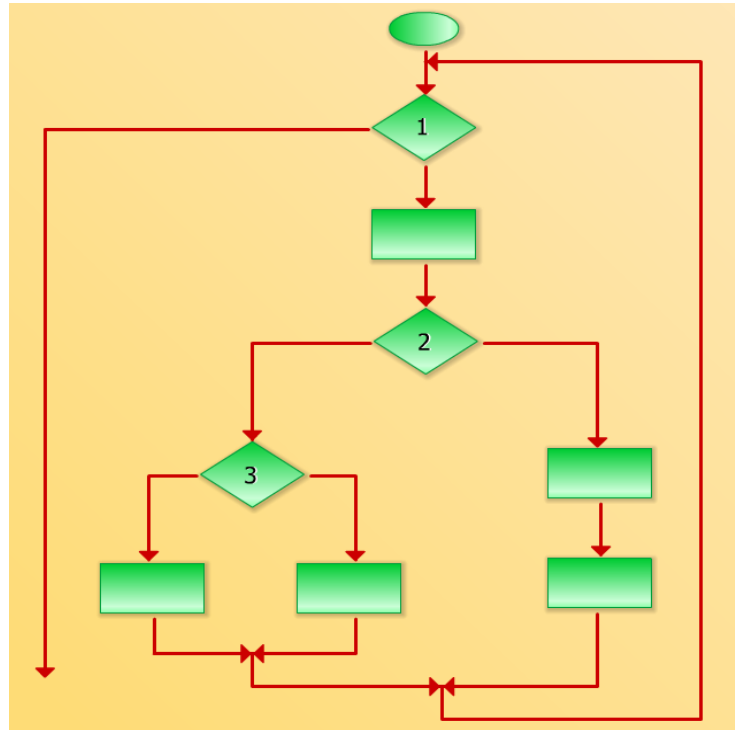
Bunun için hataların bazı özelliklerinin bilinmesinde yarar vardır:

- Bir program kesiminin uygulamada çalıştırılma olasılığı az ise o kesimde hata olması ve bu hatanın önemli olması olasılığı fazladır.
- Çoğu zaman, kullanılma olasılığı çok az olarak kestirilen program yolları, aslında çok sıkça çalıştırılıyor olacaktır.
- Yazım hataları rasgele olarak dağılır. Bunlardan bazılarını derleyiciler bulur, bazıları da bulunmadan kalır.

Temel Yollar Sınaması

- Daha önce çevrimsellik karmaşıklığı konusunda gördüğümüz hesap yöntemi ile bir programdaki bağımsız yollar bulunduktan sonra, bu kadar sayıda sınama yaparak programın her birimini bir şekilde sınamalara dahil etmiş oluruz.
- Bağımsız yolların saptanması için önce, program çizgesel bir biçime çevrilir.
- Bunu yapmak için ise, program iş akış şemaları diyagramları iyi bir başlangıç noktasıdır.

Temel Yollar Sınaması



Bir Programın İş Akış Şeması

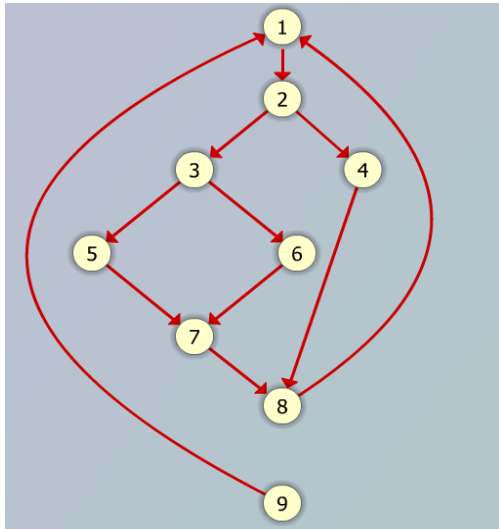
Temel Yollar Sınaması

- Program akış diyagramları matematiksel titizlik ile tanımlanmayan daha serbestçe program yapılarını alt düzeyde modelleyen çizimlerdir. Akış diyagramları ise Çizge Teorisi dalında kabul görecektir bir "Çizge"dir.
- Her çizgede olduğu gibi burada da düğümler ve dallar (veya kırımlar) bulunur.
- Program akış diyagramından akış diyagramına geçmek için süreç kutuları ortadan kaldırılır, koşul baklavaları yerine düğümler çizilir ve her koşul düğümlerine karşı düşecek birleştirme düğümleri eklenir.
- Birleştirme düğümleri, koşul kollarının kapandığı noktaya konur.
- Artık bu çizge üzerinden temel yol sayısını da verecek olan çevrimsellik karmaşıklığı sayısını hesaplayabiliriz:

$E - N + 2$
E: toplam dal sayısı
N: toplam düğüm sayısı

- Bağımsız temel yol sayısı kadar temel yolları çizge üzerinde (sonunda programa yansıtılmak üzere) veya program üzerinde işaretlenir. Sonra bu yolların hepsinin koşturulacağı sınav programları tasarlanır.

Temel Yollar Sınaması



Programın Grafik Diyagramı

Formüle göre bağımsız yol sayısı: $11 - 9 + 2 = 4$

Bunun anlamı da şöyle açıklanabilir: Çizgedeki her dal sınaama işleminde kapsanmalıdır. Bu, sınaama sırasında her işlemin çalıştırılması demektir. Her dalın en az bir kere kapsanacağı ve en az sayıda yollar bulunmalıdır. Programa ortadan giremeyeceğimize göre de bu yollar başlangıç noktasından bitiş noktasına kadar uzanmalıdır. Son olarak da minimum sayıda yol ile bu şartları sağlamalıyız. Bu sayının 4 olduğu, daha önce yukarıdaki formülde hesaplanmıştı.

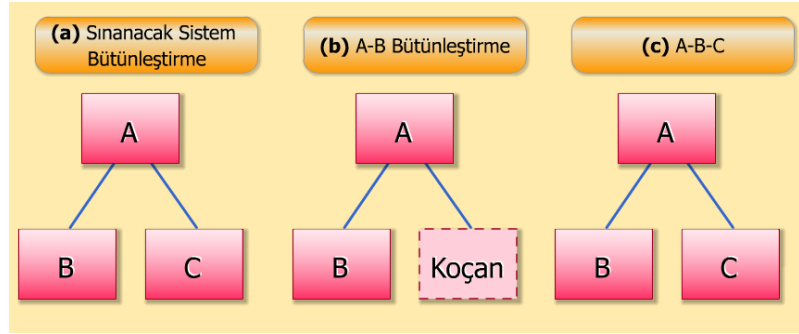
Sinama ve Bütünleştirme Stratejileri

- Genellikle sinama stratejisi, bütünleştirme stratejisi ile birlikte değerlendirilir. Ancak bazı sinama stratejileri bütünleştirme dışındaki tasaları hedefleyebilir.
- Örneğin, yukarıdan aşağı ve aşağıdan yukarı stratejileri bütünleştirme yöntemine bağlıdır.
- Ancak işlem yolu ve gerilim sinamaları, sistemin olaylar karşısında değişik işlem sıralandırmaları sonucunda ulaşacağı sonuçların doğruluğunu ve normal şartların üstünde zorlandığında dayanıklılık sınırını ortaya çıkarır.

Yukarıdan Aşağı Sınama ve Bütünleştirme

- Yukarıdan aşağı bütünleştirmede, önce sistemin en üst düzeylerinin sınanması ve sonra aşağıya doğru olan düzeyleri, ilgili modüllerin takılarak sınanmaları söz konusudur.
- En üst noktadaki bileşen, bir birim/modül/alt sistem olarak sınıandıktan sonra alt düzeye geçilmelidir.
- Ancak bu en üstteki bileşenin tam olarak sınanması için alttaki bileşenlerle olan bağlantılarının da çalışması gerekir.
- Alt bileşenler ise bu stratejiye göre henüz hazırlanmış olamazlar.
- Bunların yerine üst bileşenin sınaması için kullanılmak üzere 'koçan' programları yazılır.
- Koçanlar, bir alt bileşenin, üst bileşen ile ara yüzünü temin eden, fakat işlevsel olarak hiç bir şey yapmayan, boş çerçeve programlarıdır.
- Üst bileşenin sınanması bittikten sonra bu koçanlar, içleri doldurularak kendi kodlama ve birim sınama işlemlerini tamamladıktan sonra üst bileşen ile yeniden sınanırlar.

Bütünleştirme Sınamasında "koçan" Kullanımı



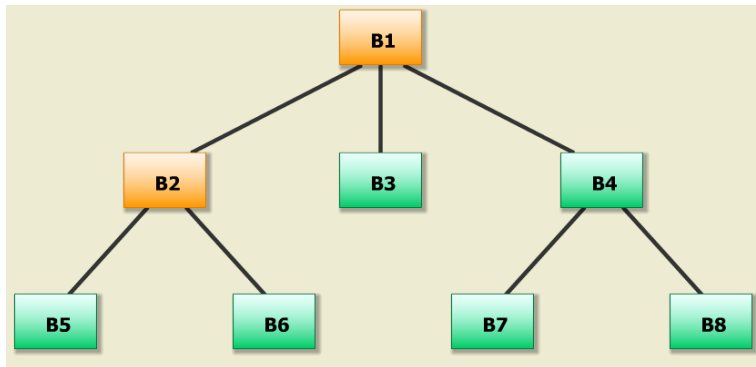
A, B, C birimlerinden oluşan ve birim şeması (a) şıkında belirtilen bir sistemin bu tür koçan kullanılarak sınanması (b) şıkı ve şekil (c) şıkında belirtilmektedir.

İlk adımda A ve B birimleri bütünleştirilir; C için bir "koçan" yazılır. İkinci adımda ise "koçan" kaldırılır ve C ile yer değiştirilerek A-B-C bütünleştirilir.

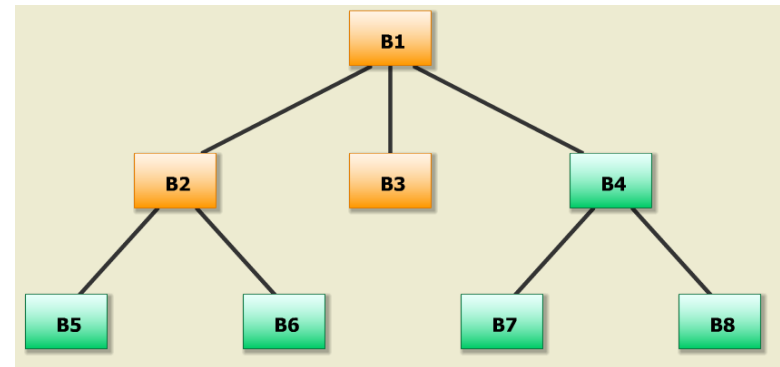
Yukarıdan aşağıya doğru bütünleştirme işleminde iki yaklaşım izlenebilir:

- 1. Yaklaşım:** Düzey Öncelikli Bütünleştirme (En üst düzeyden başlanır, öncelikle aynı düzeylerdeki birimler bütünleştirilir.)
- 2. Yaklaşım:** Derinlik Öncelikli Bütünleştirme (En üst düzeyden başlanır. Birim şemasında bulunan her dal soldan sağa olma üzere ele alınır. Bir dala ilişkin bütünleştirme bitirildiğinde diğer dalın bütünleştirilmesi başlar.)

1. Yaklaşım: Düzey Öncelikli Bütünleştirme

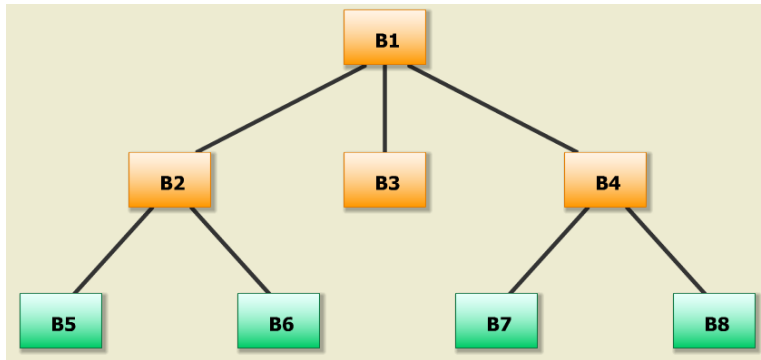


1. Adım: B1-B2 (KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)

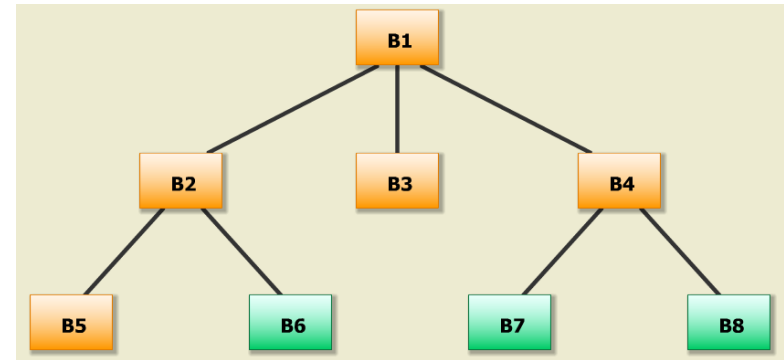


2. Adım: B1-B2-B3 (KoçanB4- KoçanB5 - KoçanB6)

1. Yaklaşım: Düzey Öncelikli Bütünleştirme

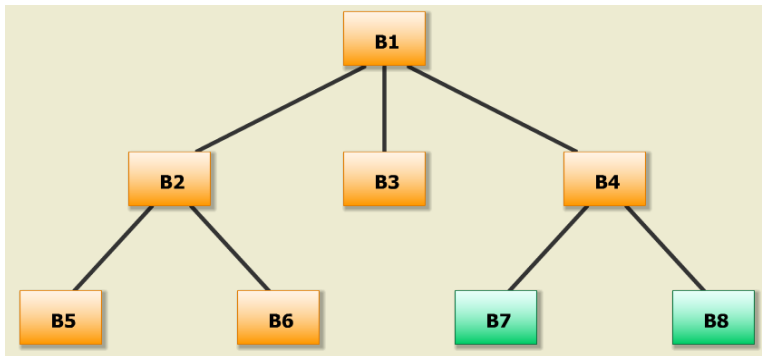


3. Adım: B1-B2-B3-B4 (KoçanB7 - KoçanB8- KoçanB5 - KoçanB6)

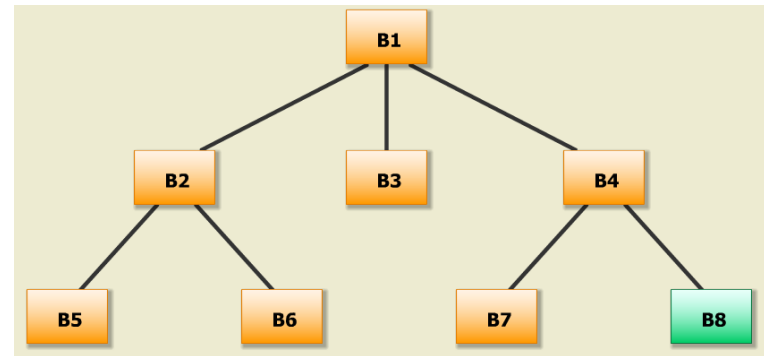


4. Adım: B1-B2-B3-B4-B5 (KoçanB7 - KoçanB8- KoçanB6)

1. Yaklaşım: Düzey Öncelikli Bütünleştirme

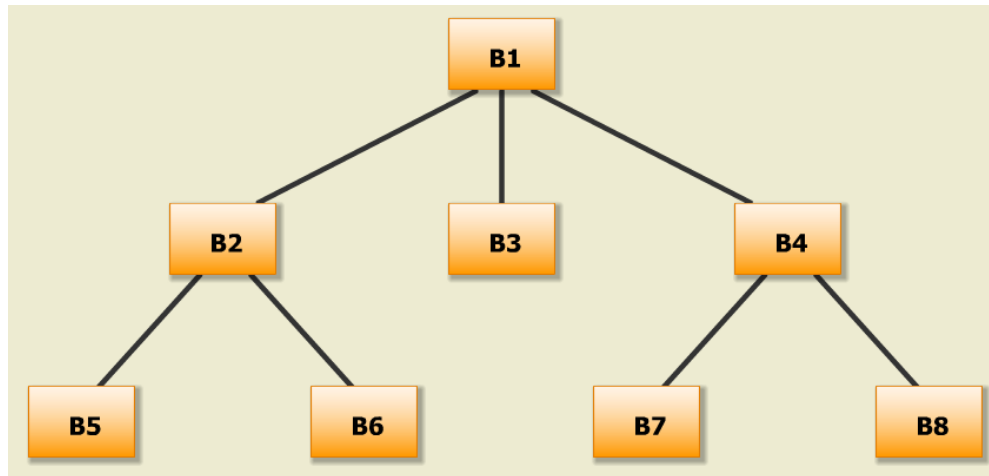


5. Adım: B1-B2-B3-B4-B5-B6 (KoçanB7 - KoçanB8)



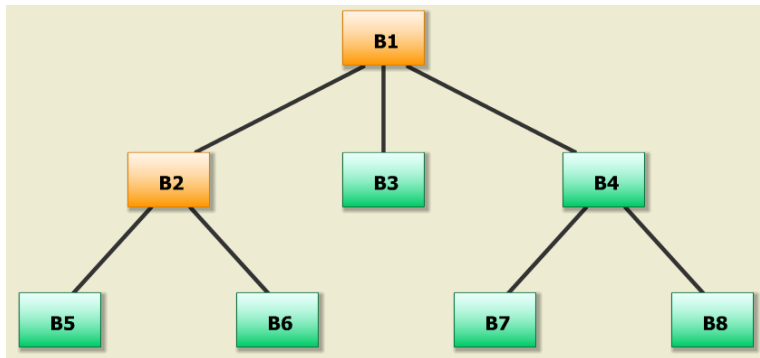
6. Adım: B1-B2-B3-B4-B5-B6-B7 (KoçanB8)

1. Yaklaşım: Düzey Öncelikli Bütünleştirme

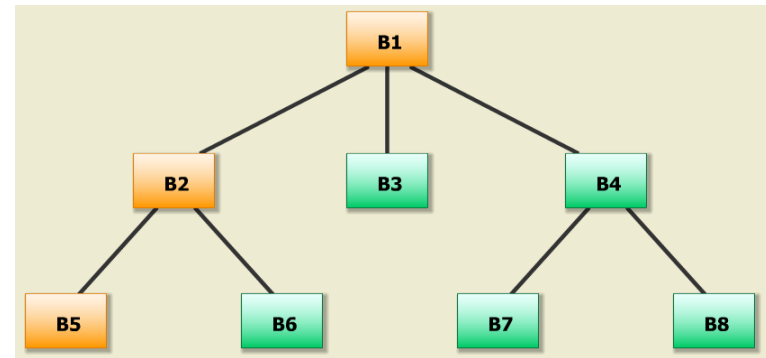


7. Adım: B1-B2-B3-B4-B5-B6-B7-B8

2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

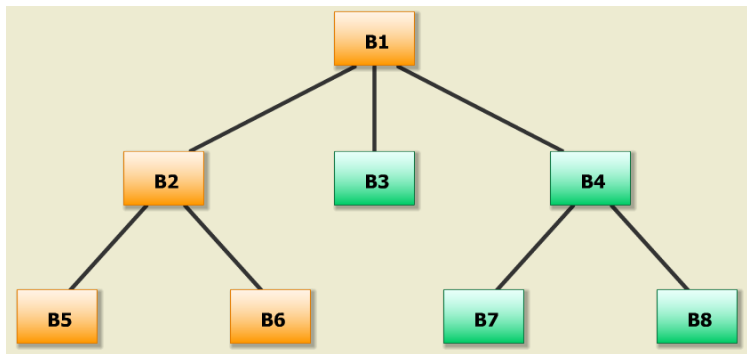


1. Adım: B1-B2 (KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)

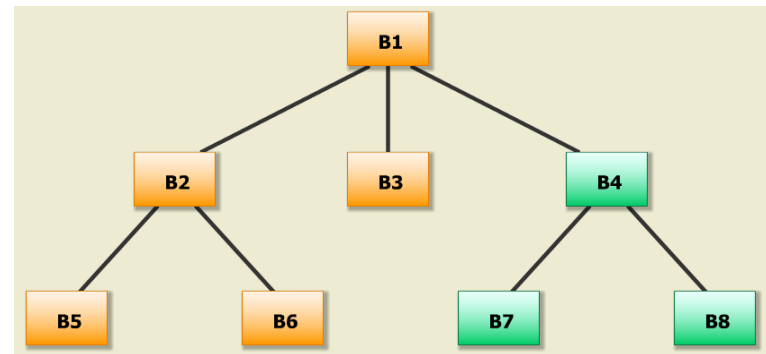


2. Adım: B1-B2 - B5 (KoçanB3- KoçanB4 - KoçanB6)

2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

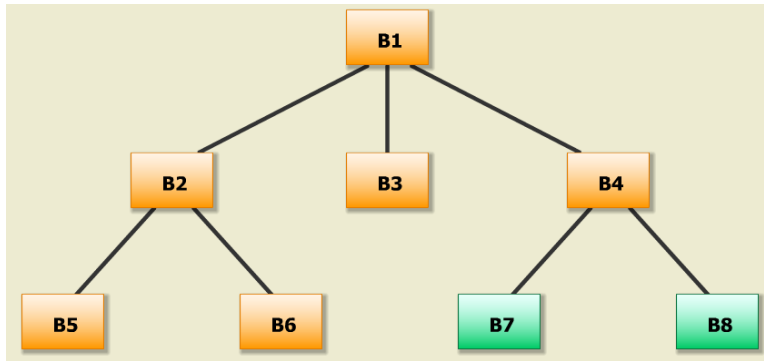


3. Adım: B1-B2-B5-B6 (KoçanB3 - KoçanB4)

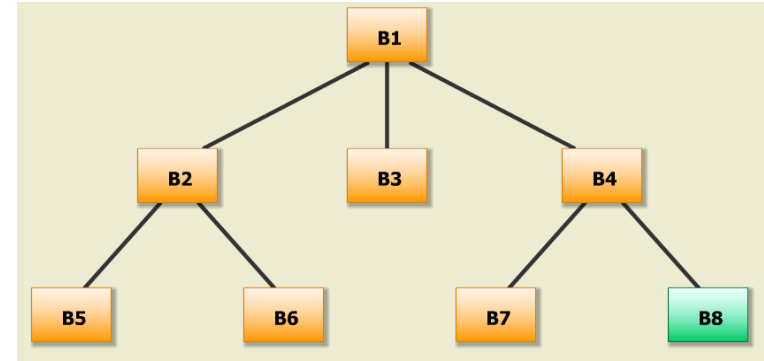


4. Adım: B1-B2-B5-B6-B3 (KoçanB4)

2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

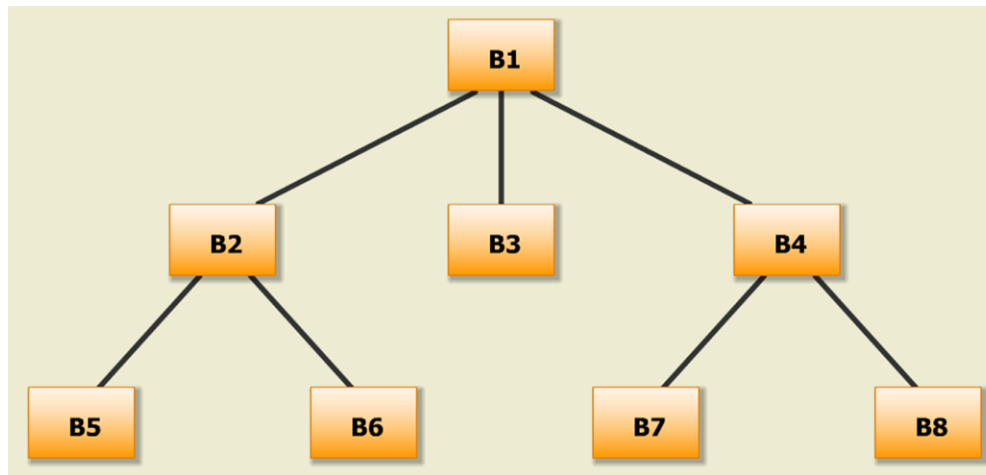


5. Adım: B1-B2-B5-B6-B3-B4 (KoçanB7 - KoçanB8)



6. Adım: B1-B2-B5-B6-B3-B4-B7 (KoçanB8)

2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

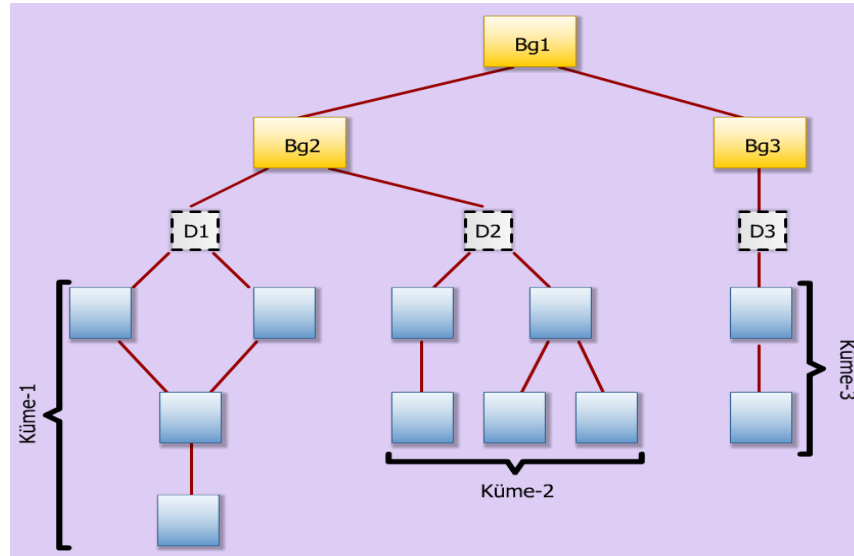


7. Adım: B1-B2-B5-B6-B3-B4-B7-B8

Aşağıdan Yukarıya Sınama ve Bütünleştirme

- Aşağıdan yukarı bütünleştirmede ise, önceki yöntemin tersine uygulama yapılır.
- Önce en alt düzeydeki işçi birimleri sınanır ve bir üstteki birimle sınama edilmesi gerektiğinde bu üst bileşen, bir 'sürücü' ile temsil edilir.
- Yine amaç, çalışmasa bile arayüz oluşturacak ve alt bileşenin sınanmasını sağlayacak bir birim edinmektir.
- Bu kez kodlama, bütünleştirme ve sınama aşağı düzeylerden yukarı düzeylere doğru gelişir ve yukarı düzeylerde önce sürücü olarak yazılan birimler sonra gerçekleriyle yer değiştirerek o düzeyin birimleri/alt sistemleri olurlar.
- Bütünleştirme yukarı doğru yapıldıkça daha az sürücü gereği duyulur.
- Uygulamada, hem aşağıdan yukarıya, hem de yukarıdan aşağıya sınama stratejilerinin iki stratejinin birleştirildiği de olur. 'Sandviç' adı verilen bu karma yaklaşımda hem üstten hem alttan sınama etkinliği sürer.

Aşağıdan Yukarıya Sınama ve Bütünleştirme



Şekilde;

1. Belirli bir yazılım alt işlevini gören alt düzey birimler **kümeler** biçiminde oluşturulurlar,
2. Denetim amaçlı bir **sürücü** programı sınama işlemi için girdi ve çıktı oluşturmak amacıyla yazılır,
3. Sürücüler aşağıdan yukarı kaldırılır ve gerçek birim ya da birim kümeleriyle değiştirilerek sınama işlemi sürdürülür.

Sinama Planlaması

- Sinama işlemi çok kapsamlıdır.
- Bir plan güdümünde gerçekleştirilmelidir.
- Böyle bir planın temel bileşenleri önceki sayfalarda belirtilmiştir.
- Yazılım yaşam döngüsünün süreçlerine koşut olarak, farklı ayrıntı düzeylerinde birden fazla sinama planı hazırlanır. Sinama planları;
 - Birim (Modül) Sinama Planı,
 - Alt Sistem Sinama Planları,
 - Bütünleştirme Sinama Planları,
 - Kabul Sinama Planları,
 - Sistem Sinama Planları biçimindedir.

Her sinama planı, sinama etkinliklerinin sınırlarını, yaklaşımını, kaynaklarını ve zamanlamasını tanımlar. Plan neyin sinanacağını, neyin sinanmayacağını, sorumlu kişileri ve riskleri göstermektedir. Sinama planları, sinama belirtilmelerini içerir.

Sinama Belirtilimleri

➤ Sinama belirtilimleri, bir sinama işleminin nasıl yapılacağına ilişkin ayrıntıları içerir. Bu ayrıntılar temel olarak:

- sinanan program modülü ya da modüllerinin adları,
- sinama türü, stratejisi (beyaz kutu, temel yollar vb.),
- sinama verileri,
- sinama senaryoları

türündeki bilgileri içerir.

➤ Sinama verilerinin elle hazırlanması çoğu zaman kolay olmayabilir ve zaman alıcı olabilir. Bu durumda, otomatik sinama verisi üreten programlardan yararlanılabilir.

➤ Sinama senaryoları, yeni sinama senaryosu üretebilmeye yardımcı olacak biçimde hazırlanmalıdır.

➤ Zira sinama belirtilimlerinin hazırlanmasındaki temel amaç, etkin sinama yapılması için bir rehber oluşturmaktır. Sinama işlemi sonrasında bu belirtilimlere,

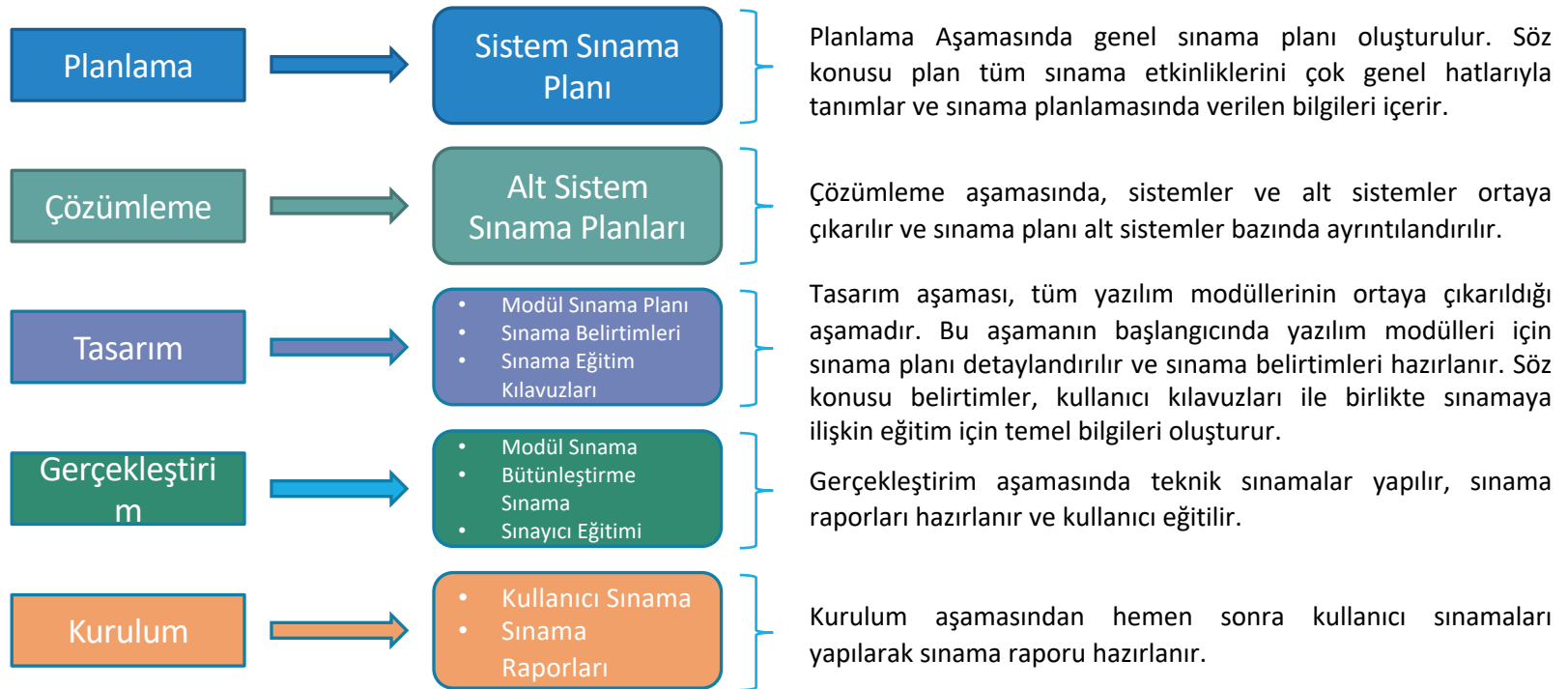
- sinamayı yapan,
- sinama tarihi,
- bulunan hatalar ve açıklamaları

türündeki bilgiler eklenerek sinama raporları oluşturulur.

Sinama Belirtilimleri

- Sinama raporları, sinama bitiminde imzalanır ve yüklenici ile iş sahibi arasında resmi belge niteliği oluşturur.
- Sinama planları; Birim (Modül) Sinama Planı, Alt Sistem Sinama Planları, Bütünleştirme Sinama Planları, Kabul Sinama Planları, Sistem Sinama Planları biçimindedir.
- Her sinama planı, sinama etkinliklerinin sınırlarını, yaklaşımını, kaynaklarını ve zamanlamasını tanımlar.
- Plan neyin sinanacağını, neyin sinanmayacağını, sorumlu kişileri ve riskleri göstermektedir. Sinama planları, sinama belirtilmelerini içerir.

Yazılım Yaşam Döngüsü Boyunca Sınama Etkinlikleri



Yazılım Yaşam Döngüsü Boyunca Sınama Etkinlikleri

- Hazırlanan sınama raporları, doğrulama ve geçerleme yaşam döngüsü işlemleri gereği "sorun yönetimi "ne iletilir. Bu bölümde hatalar kaydedilir ve bulunan hatalara karşı yapılacak işlemler planlanır.
- Sınamalar sırasında bulunan her bulgu ya da hata olarak belirtilen her durum gerçekte hata olmayabilir.
- Farklı sınavıcılardan biri, bir durumu hata olarak nitelerken diğeri aynı durumu doğru olarak değerlendirebilir.
- Bu nedenle sınama raporlarında hata olarak bildirilen her durum hemen düzeltilmek üzere ele alınmaz.
- Önce çözümlenir, kullanıcı çelişkileri giderilir ve gerçekten hata olduğuna karar verilirse düzeltilir. Söz konusu karar kullanıcı temsilcileri ile birlikte alınır.
- Sınama sırasında bulunan her hata için, değişiklik kontrol sistemine (DKS), "Yazılım Değişiklik İsteği" türünde bir kayıt girilir.

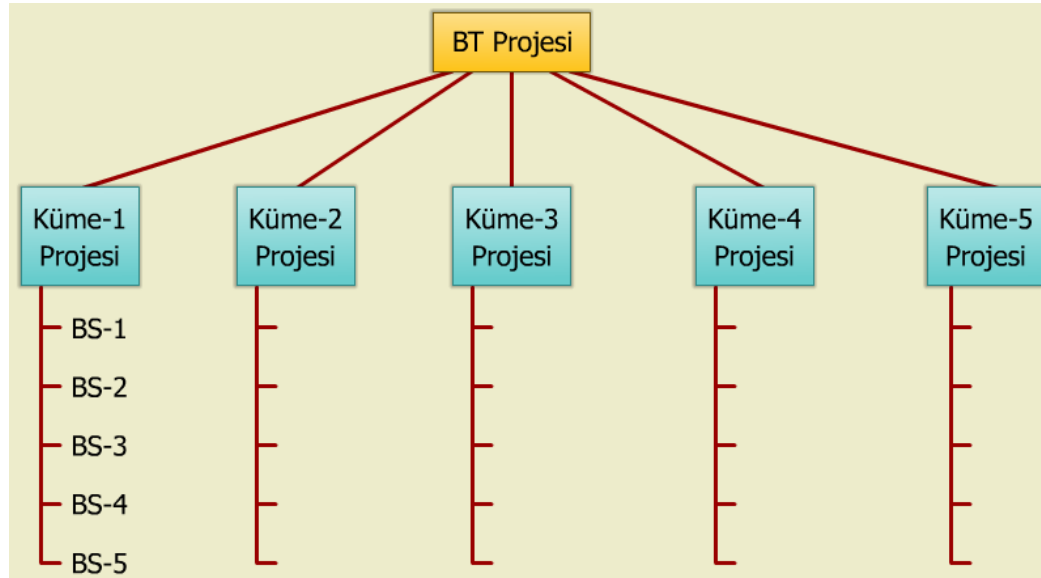
Yazılım Yaşam Döngüsü Boyunca Sınama Etkinlikleri

- Hatalar, DKS kayıtlarında aşağıdaki gibi gruplara ayrılabilir:
 - **Onulmaz Hatalar:** BT projesinin gidişini bir ya da birden fazla aşama gerileten ya da düzeltilmesi mümkün olmayan hatalardır.
 - **Büyük Hatalar:** Projenin kritik yolunu etkileyen ve önemli düzeltme gerektiren hatalardır.
 - **Küçük Hatalar:** Projeyi engellemeyen, ve giderilmesi az çaba gerektiren hatalardır.
 - **Şekilsel Hatalar:** Heceleme hatası gibi önemsiz hatalardır.

Uygulama : Görsel Yazılım Geliştirme Ortamında Sınama

- Bu kısımda, gerçek yaşam ortamında, **Oracle Designer CASE** aracı ve **Developer** görsel yazılım geliştirme platformu kullanılarak geliştirilen yazılım modüllerinin sınanması işleminin nasıl yapılacağı ve buraya kadar açıklanan sınama yöntemlerinin nasıl uygulandıkları bir örnek üzerinde anlatılmaktadır.
- Oracle Developer kullanılarak geliştirilen her yazılım formlardan oluşur. Bir form, bir ekran ve bu ekranda yapılan işlemlere karşılık gelen PL/SQL kodları biçiminde tanımlanır.
- Bu örnekte elimizde, sınama işlemine koşulacak ve uygulamanın çeşitli işlevlerine ilişkin bir dizi form olduğunu düşünebiliriz.
- Bu örnekte söz edilen uygulama, 2000'den fazla kullanıcısı olan, ülkenin çeşitli yörelerine dağılmış birimlerde çalışacak biçimde tasarlanmış ve 1000'den fazla Developer formundan oluşmaktadır.
- Uygulamanın sınama aşamasına gelmesi, 2 yıllık bir süre ve yaklaşık 100 kişi-yıl'lık bir iş gücü gerektirmiştir. Uygulama beş ana kümeye bölünmüş ve her küme belirli sayıda bilgi sistemini içermektedir.
- Toplam olarak 30 bilgi sistemi bulunmaktadır.
- Uygulama sıra düzeni bir sonraki slaytta gösterilmektedir.

Uygulama : Görsel Yazılım Geliştirme Ortamında Sınama



- Şimdi böyle kapsamlı bir uygulamanın sınama aşamasında kullanılan yöntemleri ve metodolojiyi bir sonraki slaytta inceleyelim.

Sinama Ortamı Oluşturulması

- Üretimin etkilenmemesi amacıyla, yalnızca sinayıcıların kullanacakları ve ayrı bilgisayarlardan oluşan bir sinama ortamı oluşturuldu.
- Oluşturulan sinama ortamı ile üretim ortamının bir bir aynı olması sağlandı.
- Üretimi biten yazılım parçaları, bir kayıt düzeni içerisinde sinama ortamına alındı.

Sinama Yöntemlerine Karar Verilmesi

- Uygulamada kullanılacak sinama yöntemleri aşağıdaki gibidir:
 - Teknik Sinama
 - Biçimsel Sinama
 - İşletimsel Sinama
 - Senaryo Sinama
 - Kullanıcı Sinama

Teknik Sınama

- Üretim ortamında yapılacak sınama olarak karar verildi. Bu sınama, modül sınaması ve bütünleştirme sınaması olarak üretim ekipleri tarafından gerçekleştirildi. Modül sınama yöntemi olarak ‘Beyaz kutu’ sınama yöntemi uygulandı. Tüm program deyimleri en az bir kez, tüm döngüler en az 10 kez yinelenecek biçimde sınama yapıldı.
- Bütünleştirme sınama yöntemi olarak ‘yukarıdan aşağıya sınama yöntemi’ ve ‘derinlik öncelikli bütünleştirme’ stratejisi uygulandı.

Biçimsel Sınama

- Üretim ekiplerinden bağımsız olarak, sınama ekipleri tarafından yapılan sınamadır. Bu sınama, Developer formları üzerinde görsel olarak yapıldı. Amaç, formların, önceden kararlaştırılan standartlara uygunluğunun saptanmasıydı.
- Örneğin,
 - form alanları, kararlaştırılan uzunlukta mı?
 - Başlıklar istenilen gibi koyu mu?
 - Yardım düğmesi hep aynı yerde mi vb.
- Sınama, formlar işletilmeden yapılır. Tüm formlar tek tek incelenir ve standartlara uygun olmayanlar belirlenip, düzeltilmek üzere üretim ekibine geri iletilir.
- Biçimsel sınamaların yapılması amacıyla denetim listeleri hazırlanır ve sınama sırasında bu listeler kullanılır. Listelere kaydedilen her sonuç DKS'ye aktarılır.
- Bu yolla üretim ekiplerinin performansı izlenebilir.

İşletimsel Sınama

- Üretim ekiplerinden bağımsız olarak, sınama ekipleri tarafından yapılan sınamadır.
- Biçimsel sınama işlemi bittikten sonra yapılır.
- Bu sınamada her form ayrı ayrı çalıştırılarak işlem yapılır.
- Amaç, formun çalışıp çalışmadığının belirlenmesidir.
- Form alanlarının sınır değerlerle çalışıp çalışmadığı, aykırı değer verildiğinde uygun hata iletisi alınıp alınmadığı vb. belirlenmeye çalışılır.

Senaryo Sınama

- Sınama ekipleri tarafından yapılan sınamadır.
- Ancak, senaryoların hazırlanması sırasında üretim ekipleri ile birlikte çalışılır.
- Amaç, birden fazla formun bir arada sınanmasıdır. Bu amaçla, 'senaryo'lar hazırlanır.
- Her senaryo, çözümleme aşamasında belirlenen bir iş fonksiyonuna karşılık gelecek biçimde hazırlanır.

Kullanıcı Sinaması

- Kullanıcılar tarafından yapılması öngörülen sinamadır.
- Senaryo sinamasının kullanıcı tarafından yapılan biçimi olarak düşünülebilir.

Kullanıcı Sinama Eğitimi

- Sinama yapılacak kullanıcı sinayıcılarına, sinamaların nasıl yapılacağına ilişkin eğitim verilmesi gerekmektedir.
- Eğitim kitapçıklarının hazırlanması amacıyla, senaryo sinamalarında kullanılan "senaryo"lar ve kullanıcı kitapçıkları kullanılır.

Sınamaların Yapılması

- Sınamalar sırasında yapılan her işlem, DKS'de izlenir. Özellikle kullanıcı sınamalarının izlenmesi ve ortaya çıkabilecek tartışmaların önlenmesi bu yolla sağlandı.
- Yaklaşık 100 kullanıcı sınavıcısının, birbirinden farklı yerlerde yaptıkları sınamalar için "haftalık sinama sonuçları" formları toplandı.
- Yerinde destek elemanları, sürekli olarak kullanıcı sınavıcılarını ziyaret ederek sinama sonuçlarının düzenli olarak toplanmasını sağladı.
- Bu formlar DKS'ye aktarılarak, daha sonra Yazılım Doğrulama ve Geçerleme Raporları için önemli girdiler oluşturdu.

Özet

Doğrulama: Doğru ürünü mü üretiyoruz ?

- Ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmadığına dair etkinliklerden oluşur.

Geçerleme: Ürünü doğru olarak mı üretiyoruz?

- Ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.

Doğrulama ve geçerleme işlemleri temel olarak çeşitli düzeylerde sınaama, gözden geçirme, denetim ve hata giderme süreçlerinden oluşur.

Doğrulama Teknikleri: Gözden Geçirme, Üstünden Geçme, Denetleme ve İncelemedir.

Sınaama planları;

- Birim (Modül) Sınaama Planı,
- Alt Sistem Sınaama Planları,
- Bütünleştirme Sınaama Planları,
- Kabul Sınaama Planları,
- Sistem Sınaama Planları biçimindedir.

Özet

Hatalar, DKS kayıtlarında aşağıdaki gibi gruplara ayrılabilir:

- **Onulmaz Hatalar:** BT projesinin gidişini bir ya da birden fazla aşama gerileten ya da düzeltilmesi mümkün olmayan hatalardır.
- **Büyük Hatalar:** Projenin kritik yolunu etkileyen ve önemli düzeltme gerektiren hatalardır.
- **Küçük Hatalar:** Projeyi engellemeyen, ve giderilmesi az çaba gerektiren hatalardır.
- **Şekilsel Hatalar:** Heceleme hatası gibi önemsiz hatalardır.

Uygulamada kullanılacak sınaama yöntemleri aşağıdaki gibidir:

- Teknik Sınama
- Biçimsel Sınama
- İşletimsel Sınama
- Senaryo Sınama
- Kullanıcı Sınama

Sorular

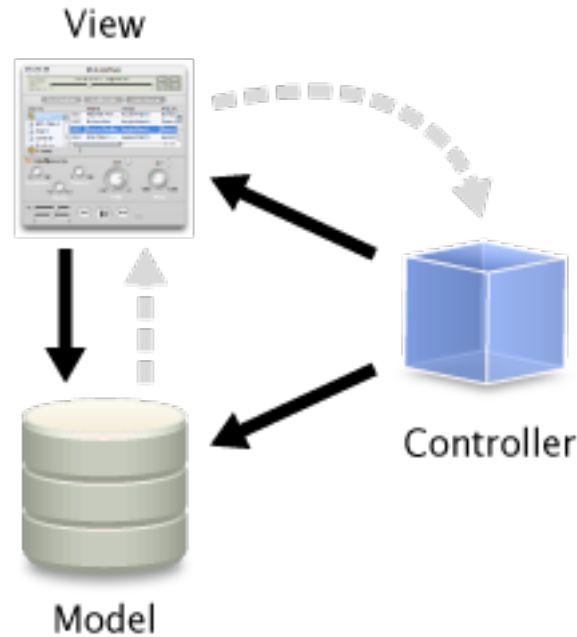
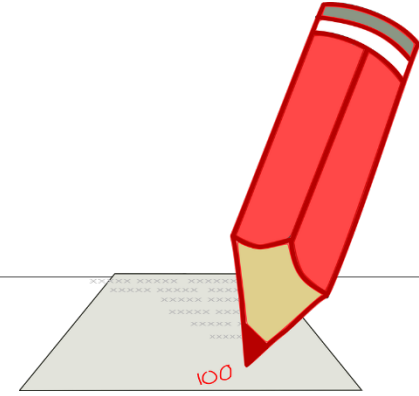
1. Doğrulama ile Geçerleme arasındaki farklılıkları belirtiniz. Birer örnekle açıklayınız.
2. Sınama Yöntemlerini açıklayınız.
3. "Beyaz Kutu" sınama ile "Temel Yollar Sınama" yöntemleri arasındaki farklılıkları belirtiniz.
4. Sınama Yöntemleri ile sınama belirtilimleri arasındaki farkı belirtiniz.
5. Yukarıdan aşağıya doğru bütünleştirme ve aşağıdan yukarıya bütünleştirme yöntemlerinin zorluklarını ve kolaylıklarını belirtiniz.
6. Sınama belirtilimlerinin önemi nedir.
7. Kullanıcı sinaması sırasında yaşanabilecek sorunları belirtiniz.

Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- ” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- ”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Ödev

- Yazılım Mimarileri Hakkında Araştırma Yapınız.
- İki Katmanlı ve Üç Katmanlı Mimarilerle yapılmış örnek sistemleri araştırınız.



Sorularınız





YMH311-Yazılım Tasarım Ve Mimarisi

Yazılım Mimari Stilleri ve Desenler

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

Tasarım kalıpları (design patterns).....7

Yazılım Mimarisi.....19

Katmanların Oluşturulması.....25

Model-View-Controller (MVC) Stili.....42

GENEL BAKIŞ

KISIM 1 – Yazılım Tasarımında Kalıplar

KISIM 2 – Mimari Stilleri



KISIM 1

Yazılım Tasarımında Kalıplar (Patterns in Software Design)



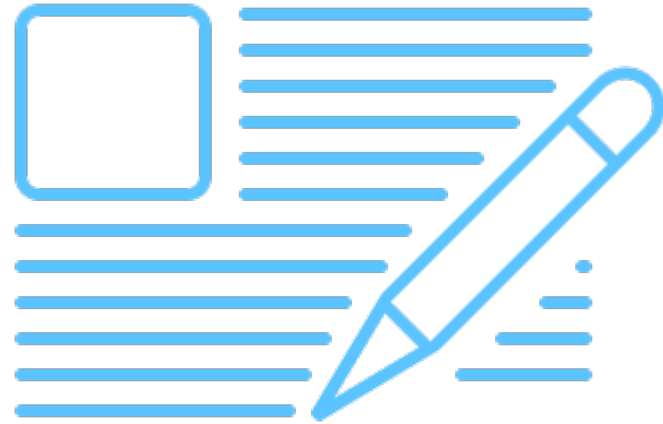
Amaçlar

- Tasarım kalıplarının neden önemli olduğunu açıklamak
- Yazılım tasarım kalıplarını tanımlamak
- Ayrıntılarına bağlı olarak tasarım kalıplarının bir sınıflandırmasını sunmak
- Tasarım kalıbı kataloglarını tartışmak



İçerik

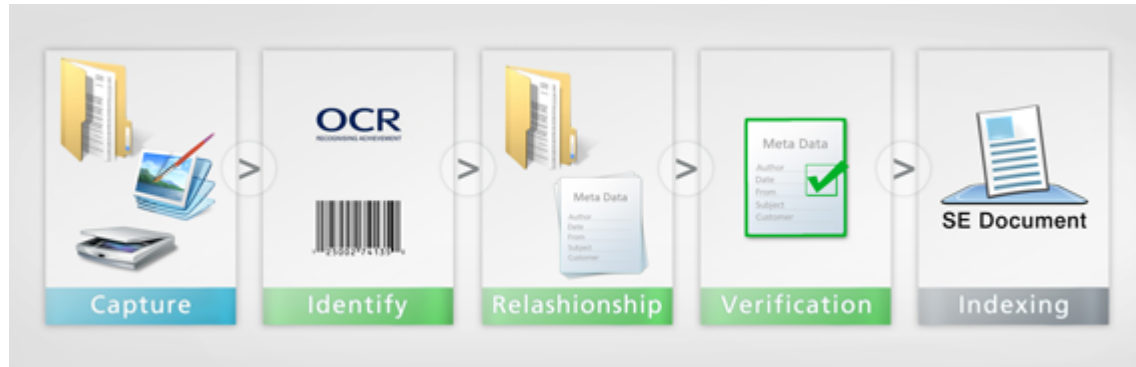
- Tasarım kalıpları (design patterns) neden önemli?
- Christopher Alexander
- Yazılım tasarım kalıpları
- Kalıpların ayrıntıları
- Kalıp katalogları



Neden Tasarım Kalıpları?

Uzman tasarımcılar acemilerden farklı davranır — **uzmanlar, acemilerin bilmediği ne biliyor olabilir?**

Diğer şeylerin dışında, uzmanlar yeni problemlere uyguladıkları, geçmiş deneyimlerden gelen başarılı tasarım kalıpları bilgisine sahiptir.



Kalıpların Avantajları

- **İletişimi arttırmak** (Promoting communication)—Kalıp isimleri ve avantaj/dezavantajlarının bilinmesi tasarımcılar arasındaki iletişimi hızlandırır
- **Dokümantasyonu kolaylaştırmak** (Streamlining documentation)—Kalıpların yapılarını ve davranışlarını ayrıntılandırmaya gerek yoktur
- **Verimliliği arttırmak** (Increasing efficiency)—Kalıplara yönelik araç desteği yazılım geliştirmeyi hızlandırır
- **Yeniden kullanımı desteklemek** (Supporting reuse)—Kalıplar ve onların implementasyonları geniş biçimde yeniden kullanılabilir
- **Yeni fikirler sağlamak** (Providing ideas)—Kalıplar bir tasarımın ya da iyileştirmenin başlangıç noktası olabilir.

Bina Mimarisinde Tasarım Kalıpları

- 1970'lerde mimar Christopher Alexander bina tasarımında tasarım kalıplarına bağlı yeni bir yaklaşım öne sürdü.
- Alexander diyordu ki:
 - Kalıplar insan anatomisi, psikolojisi, fizyolojisi, sosyolojisi, ve politikasının doğal sonuçlarıdır;
 - Büyük mimariler daima kalıplara dayalı gerçekleştirilmiştir, fakat bunlar üzerinde sistematik olarak çalışılmamıştır; ve
 - Kalıplar bilindiği ve anlaşıldığı taktirde herkes büyük yapılar oluşturulabilir.



Alexander'ın Kalıplarından Örnekler

Yayılmış İşyerleri (Scattered Work)—Yasalar ve teşvikler kullanılarak iş yerlerinin şehrin her yerine dağılmasının sağlanması.

4-Kat Sınırı (Four-Story Limit)—Şehirdeki binaların çoğunun en fazla 4 kat yüksekliğinde olması.

Güneye Bakan Dış Mekanlar (South Facing Outdoors)—Binaların daima dış mekanların kuzeyine inşa edilmesi.

Sıcak Renkler (Warm Colors)—Odadaki ışık ve duvar renklerinin sarı-yeşil ton elde edilecek şekilde seçilmesi.

Alexander'ın Çalışmasının Sonuçları

- Alexander'ın çalışması, kendisinin de itiraf ettiği gibi, bina mimarları arasında yaygın kabul görmedi, ve profesyonel olmayanların bile büyük mimariler tasarlamasına yönelik çabaları da genellikle başarısız oldu.
- Fakat Alexander, tasarım kalıpları kullanılarak üretim yapılması konusunda yazılım geliştirme topluluklarına esin kaynağı oldu.



Tasarım Kalıpları

Bir **kalıp/desen** (pattern) taklit edilmek için önerilen bir modeldir. Bir **yazılım tasarım kalıbı** (software design pattern) bir yazılım tasarım probleminin çözümünde kullanılacak, taklit edilmek için önerilen bir modeldir.

Tasarım Kalıplarının Ayrıntılanışı

- **Mimari stilleri veya kalıpları** (Architectural styles or patterns) bütün bir sistem ve alt-sistemler içindir.
- **Tasarım kalıpları** (Design patterns) etkileşen çeşitli fonksiyonlar veya sınıflar içerirler.
- **Veri yapıları & algoritmalar** (Data structures & algorithms) alt-düzey kalıplardır.
- **Programlama deyimleri** (Idioms) spesifik programlama dilleri ile bir takım işleri yapmanın yollarıdır.

Kalıp Katalogları

- Tasarım kalıplarının öneminin anlaşılması, kalıp kataloglarının oluşturulmasına yol açtı.
- Bunlar bina mimarisinde veya iç mimaride kullanılan kalıp kitaplarına ya da mühendislikte kullanılan el kitaplarına benzer kalıp kataloglarıdır.
- Bu ders kapsamında yazılım tasarım kalıplarının küçük bir bölümüne değineceğiz.

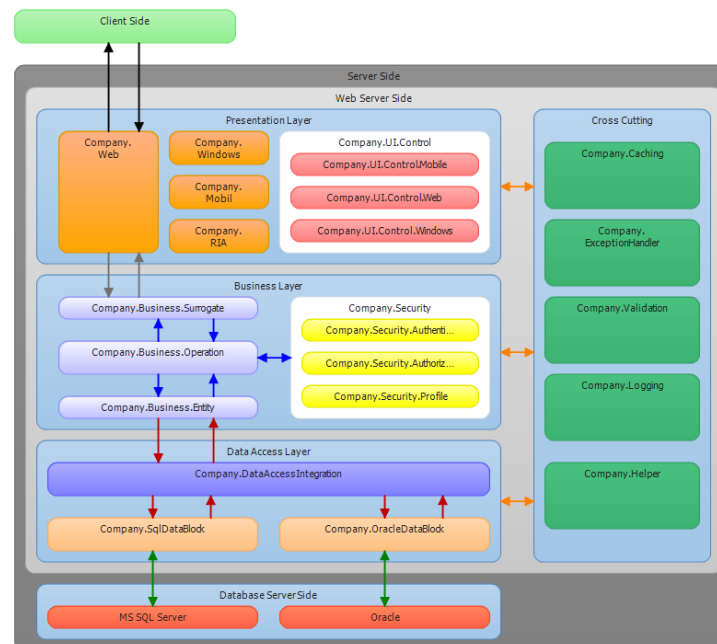


Özet

- Tasarım kalıpları, bir alandaki uzmanlığı ve deneyimi barındıran, hem acemilere hem de uzmanlara büyük kolaylıklar sağlayan şablonlardır.
- Kalıplar çeşitli ayrıntı düzeylerinde olabilir, örneğin mimari stilleri, orta-düzey tasarım kalıpları, veri yapıları ve algoritmalar, ve programlama deyimleri.

KISIM 2

Mimari Stilleri (Architectural Styles)



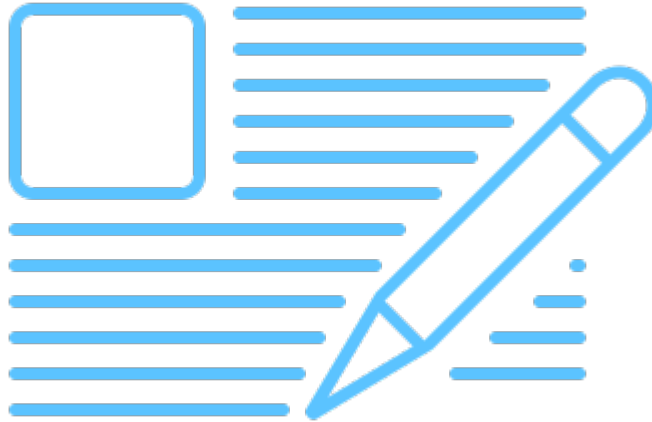
Amaçlar

- Mimari tasarım ve mimari stilinin tanımlarını gözden geçirmek
- Çeşitli önemli mimari stillerini sunmak:
 - Katmanlı (Layered)
 - Kanal-ve-Filtre (Pipe-and-Filter)
 - Paylaşımlı-Veri (Shared-Data)
 - Olay-Güdümlü (Event-Driven)
 - Model-View-Controller
 - Hybrid



İçerik

- Yazılım mimarisi ve mimari stilleri
- Katmanlı (Layered) stil
- Kanal-ve-Filtre (Pipe-and-Filter) stili
- Paylaşılan-Veri (Shared-Data) stili
- Olay-Güdümlü (Event-Driven) stili
- Model-View-Controller stili
- Hibrit mimariler



Yazılım Mimarisi

Bir **yazılım mimarisi** (software architecture) bir programın büyük bileşenlerinden, onların sorumluluk ve özelliklerinden, ve aralarındaki ilişki ve etkileşimlerden oluşan yapısıdır (structure).

Yazılım Mimarisi Konuları

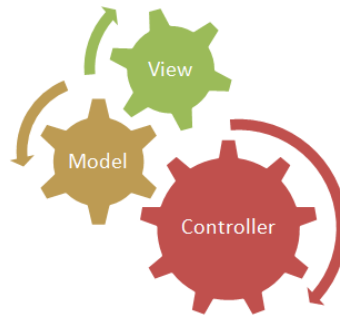
- Sistem fonksiyonlarının kabaca ayrıştırılması (decomposition)
- İşlevlerin bileşenlere tahsis edilmesi
- Bileşen arabirimleri (interfaces)
- Bileşenler arasındaki iletişim ve etkileşim
- Bileşen ve sistem özellikleri: kapasite, verim, ve kısıtlar (capacity, throughput, constraints)
- Yaygın tasarım stillerinin yeniden kullanımı

Mimari Stilleri (Architectural Styles)

Bir **mimari stili** (architectural style) bir program veya sistemi oluşturan tipler ve bunların etkileşimlerinin bir paradigmasıdır.

Katmanlı Mimariler (Layered Architecture)

- Program bir dizi katmana veya gruba ayrıştırılır.
- Bir katman altındaki katman(lar)dan hizmet alır ve üzerindeki katman(lar)a hizmet verir.
- Tüm mimari stilleri içerisinde katmanlı stil en çok kullanılan stildir.

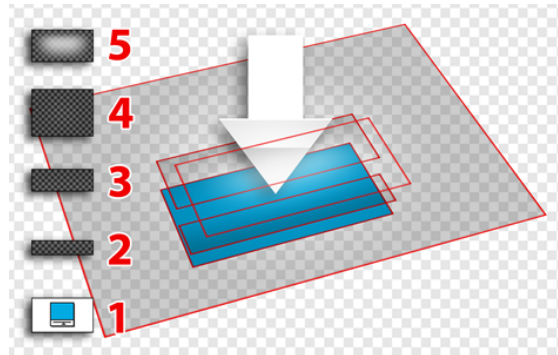


Katman Kısıtları

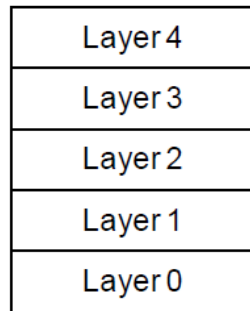


Statik yapı (Static structure)—Yazılım, her bir katman iyi tanımlanmış bir arabirim üzerinden yüksek uyuma (cohesion) sahip bir dizi servisi sağlayacak şekilde bölümlenir.

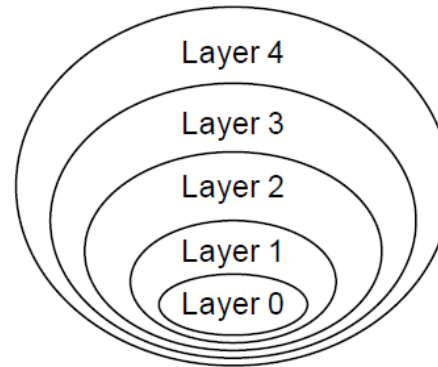
Dinamik yapı (Dynamic structure)—Her bir katman yalnızca hemen altındaki katmanı kullanabilir (**Strict Layered**) veya altındaki tüm katmanları kullanabilir (**Relaxed Layered**).



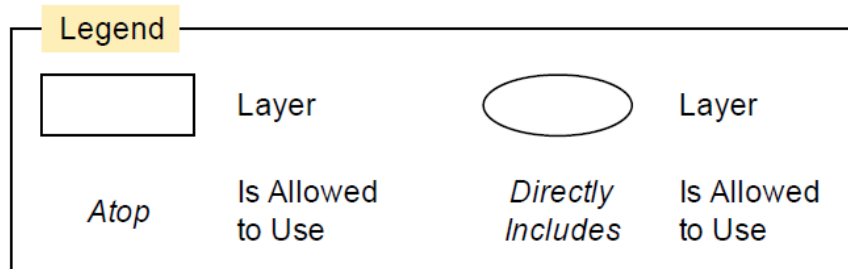
Katmanların Gösterimi



Wedding Cake
Diagram



Onion Diagram



Katmanların Oluşturulması

Soyutlama düzeyleri

- Örneğin: Ağ iletişim katmanları

Sanal makinalar

- Örneğin: İşletim sistemleri, yorumlayıcılar (interpreters)

Bilgi saklama, decoupling, vb.

- Örneğin: Kullanıcı arayüzü katmanı, sanal cihaz (cihaz sürücüsü) katmanı

Katmanlı Stilin Avantajları

- Katmanlar son derece yüksek uyumludur (cohesive) ve bilgi saklamayı arttırır.
- Katmanlar, üzerlerindeki katmanlarla yüksek derecede bağlı değildir (loose coupling), bu sayede toplam bağıllık azaltılmış olur.
- Katmanlar programı parçalara ayırmayı kolaylaştırır, karmaşıklığı azaltır.
- Katmanları değiştirmek veya onarmak, veya tüm katmanı başkasıyla değiştirmek oldukça kolaydır.
- Bir katmana işlevsellik ekleyerek katmanı genişletmek kolaydır.
- Katmanlar genellikle yeniden kullanım için çok uygundur.

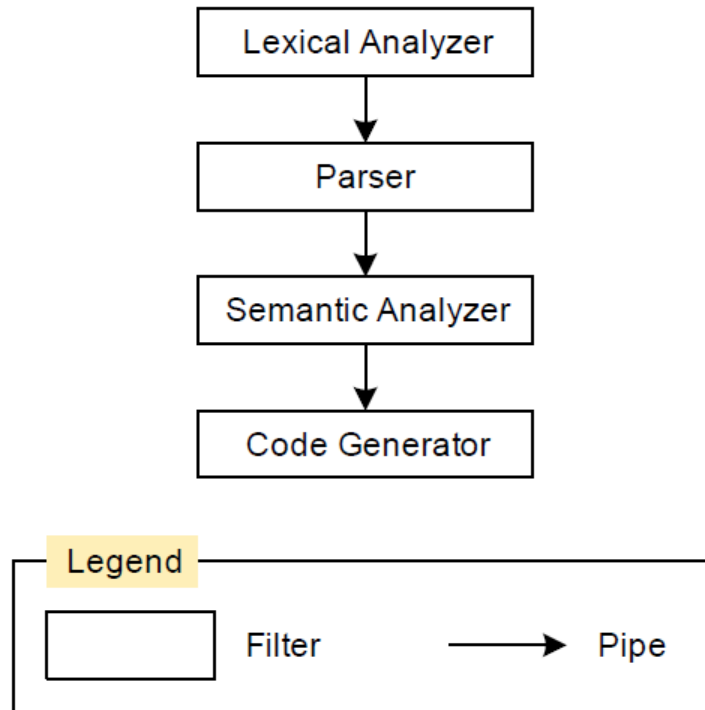
Katmanlı Stilin Dezavantajları

- Herşeyi bir çok katman içinden geçirmek sistemi karmaşık hale getirebilir ve performansı düşürebilir.
- Bir çok katman içinden hata ayıklamak genellikle zordur.
- Doğru katmanları oluşturmak zor olabilir.
- Önceden öngörüleemeyen bazı işlevsellikleri sağlayabilmek için katmanlar arası kısıtların ihlal edilmesi zorunlu olabilir.

Kanal-ve-Filtre (Pipe-and-Filter) Stili

- Bir **filtre** (filter) bir girdi akışını (input stream) bir çıktı akışına (output stream) dönüştüren bir program bileşenidir.
- Bir kanal (pipe) bir akışı taşıyan aracı bir mecradır.
- **Pipe-and-Filter** stili, program bileşenlerinin kanallar ile birleştirilmiş filtrelerden oluştuğu dinamik bir modeldir.

Kanal-ve-Filtre Örneği



Kanal-ve-Filtre Karakteristikleri

- Kanallar (Pipes) genellikle izole edilmiştir ve yalnızca veri akışları üzerinden iletişim kurarlar. Bu sayede yazılmaları, test edilmeleri, yeniden kullanılmaları, ve başkasıyla değiştirilmeleri kolaydır.
- Filtreler eşzamanlı (concurrent) olarak çalışabilir.
- Kanalların filtreleri senkronize etmesi gerekir
- Pipe-and-filter topolojileri acyclic graph olmalıdır.
- Zamanlama ve dead-lock sorunları yaşanmaması için
- Basit ve doğrusal bir düzenlemesine örnek olarak **pipeline** verilebilir.

Kanal-ve-Filtre Avantajları

- Filtreler kolayca modifiye edilebilir ve bir başkasıyla değiştirilebilir.
- Filtreler çok az bir çaba ile yeniden düzenlenebilir, bu sayede benzer programlar geliştirmek kolaydır.
- Filtreler son derece yüksek yeniden kullanılabilirlik sunarlar.
- Eşzamanlılık desteklenir ve implemente etmesi nispeten kolaydır.

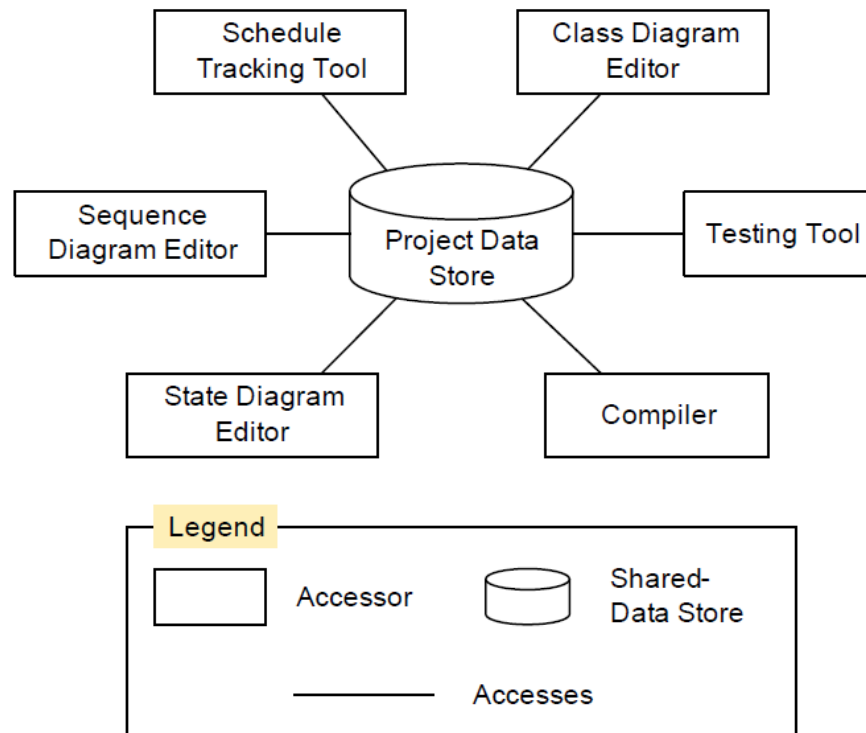
Kanal-ve-Filtre Dezavantajları

- Filtreler yalnızca kanallar üzerinden haberleşir, dolayısıyla onları koordine etmek zor olabilir.
- Filtreler genellikle basit veri akışları (data stream) üzerinde çalışırlar, dolayısıyla boşa giden veri dönüşüm eforuna neden olabilirler.
- Hata yakalaması/yönetimi zordur.
- Eşzamanlılıktan elde edilen kazanç yanıltıcı olabilir.

Paylaşılan-Veri (Shared-Data) Stili

- Bir veya daha fazla **paylaşılan-veri deposu** (shared-data stores) yalnızca bu veri depoları üzerinden iletişim kuran bir veya daha fazla **paylaşılan-veri erişimcisi** (shared-data accessors) tarafından kullanılır.
- İki türü vardır:
 - **Karatahta stili** (Blackboard style)—Paylaşılan-veri depoları depolarda değişim olduğunda erişimcileri bilgilendirir.
 - **Depo stili** (Repository style)—Paylaşılan-veri depoları pasiftir ve erişimciler tarafından manipüle edilir.
- Bu yalnızca dinamik bir modeldir.

Paylaşılan-Veri Stili Örneği

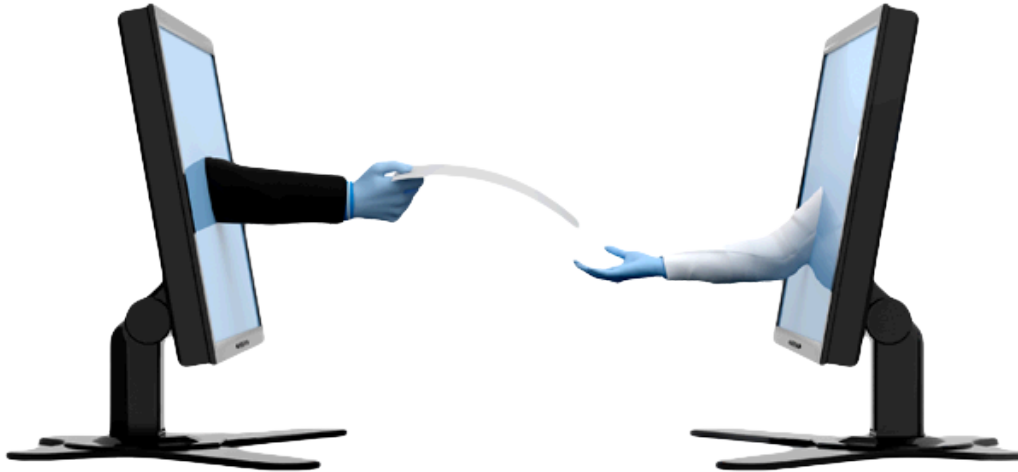


Paylaşılan-Veri Stili Avantajları

- Paylaşılan-veri erişimcileri yalnızca paylaşılan-veri depoları üzerinden haberleşir, dolayısıyla bunları değiştirmek, çıkartmak, ya da eklemek kolaydır.
- Erişimci bağımsızlığı, sistemin dayanıklılığını ve hata toleransını artırır.
- Tüm veriyi paylaşılan-veri deposuna yerleştirmek güvenliği ve denetimi kolaylaştırır.

Paylaşılan-Veri Stili Dezavantajları

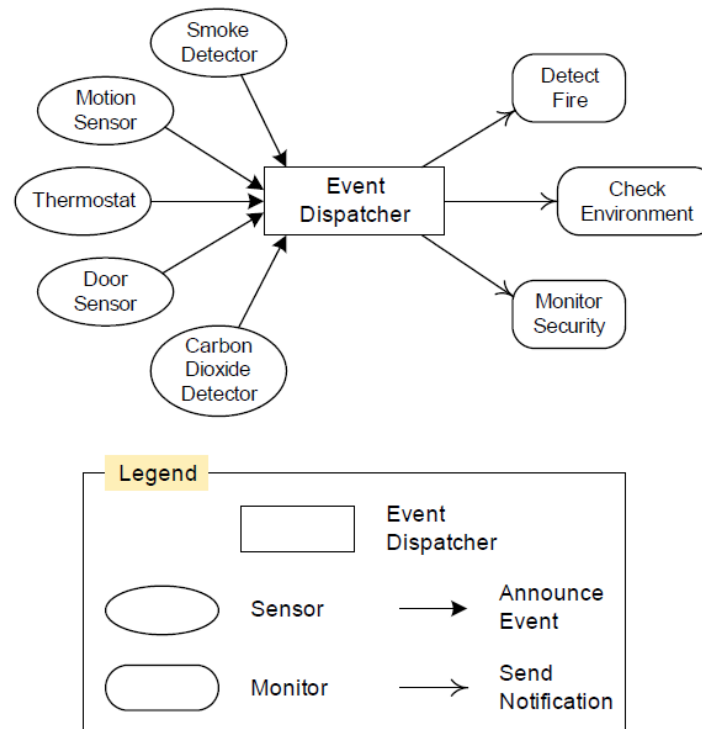
- Tüm verinin paylaşımlı-veri deposu üzerinden kullanılmasını zorlamak performansı düşürebilir.
- Paylaşılan-veri deposunda sorun oluşursa, bundan tüm program olumsuz etkilenir.



Olay-Güdümlü (Event-Driven) Stili

- Ayrıca **üstü kapalı çağırma** (Implicit Invocation) stili olarak da bilinir.
- Bir **olay** (event) dikkate değer herhangi bir oluşum/etkinliktir.
- Bir **olay dağıtım memuru** (event dispatcher) olayı duyuran ve olaydan haberdar edilen bileşenler arasında aracılık eder.
- Bu yalnızca dinamik bir modeldir.

Olay-Güdümlü Stili Örneği



Çeşitli Varyasyonlar

- Olaylar yalnızca bildirim şeklinde olabilir ya da veri taşıyabilir.
- Olay hareket memuru tarafından bazı olaylar manipüle edilebilir ya da gözardı edilebilir.
- Olaylar senkron ya da asenkron biçimde yönetilebilir.



Olay-Güdümlü Stili Avantajları

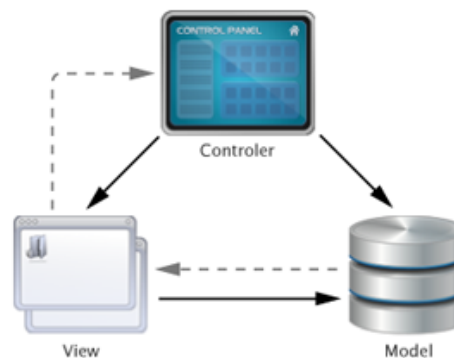
- Bileşenleri eklemek veya çıkartmak kolaydır.
- Bileşenler bağlı değildir (decoupled), bu sayede son derece yüksek yeniden kullanılabilirliğe ve değiştirilebilirliğe sahiptirler.
- Bu stil kullanılarak oluşturulan sistemler genelde sağlamdırlar ve hata toleransları yüksektir.

Olay-Güdümlü Stili Dezavantajları

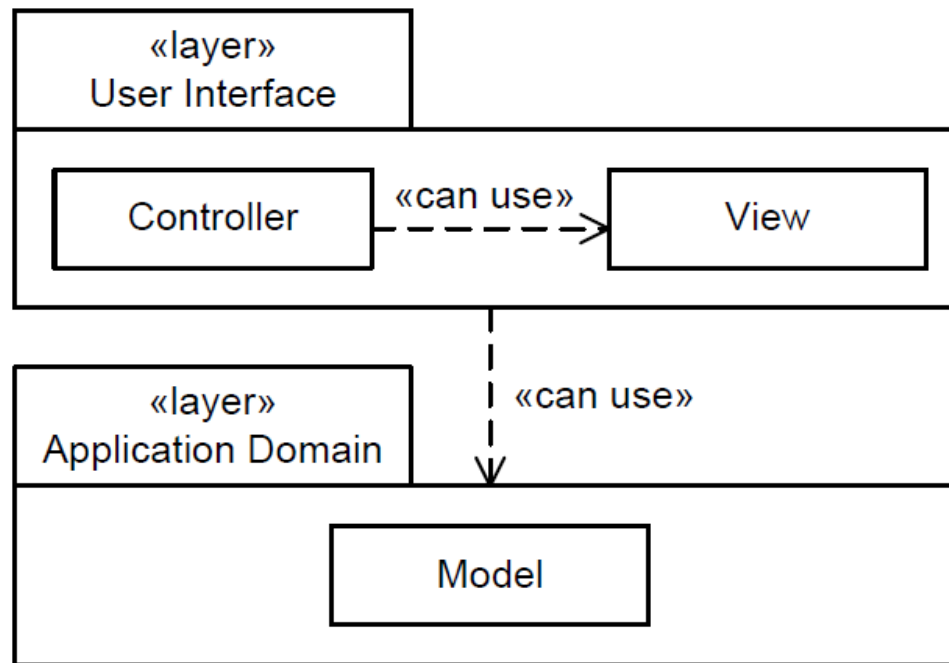
- Olay hareket memuru tarafından yönetildiğinde bileşenlerin etkileşiminde tuhaflıklar olabilir.
- Olayların sıralamasında ve zamanlamasında herhangi bir garanti verilemez, bu da doğru programlar yazmayı zorlaştırabilir.
- Olay trafiği çok değişken olabilir, bundan dolayı performans hedeflerine ulaşmak zor olabilir.

Model-View-Controller (MVC) Stili

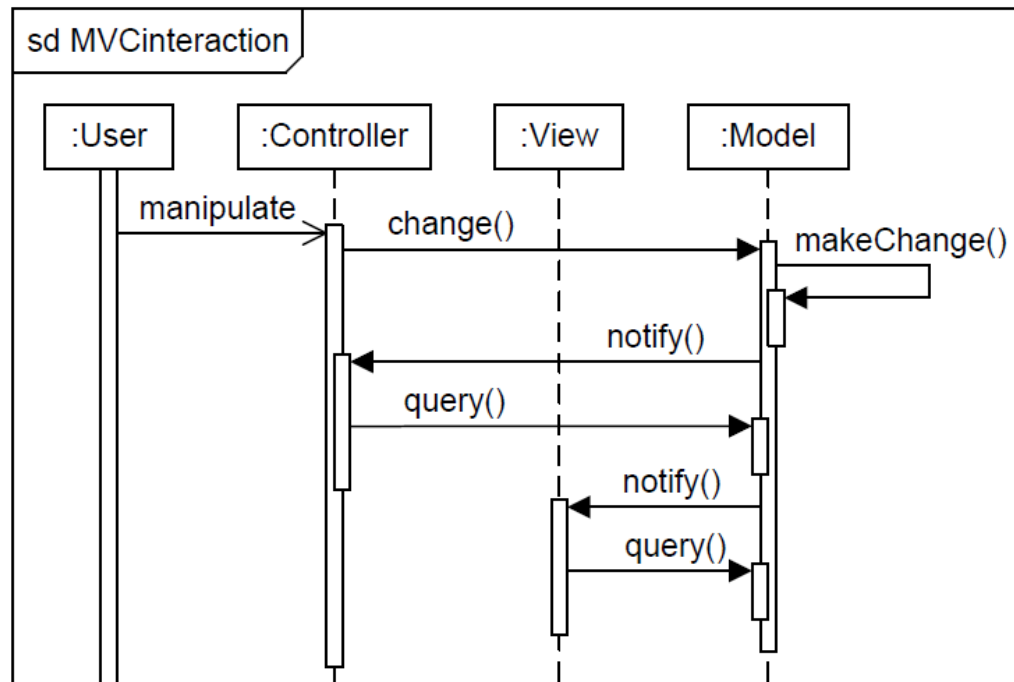
- Bu stil kullanıcı arayüzü ile problem-alanı bileşenleri arasındaki ilişkilerin nasıl kurulacağını modeller.
- **Model**—Kullanıcı arayüzünden tamamen bağımsız bir şekilde programın amaçlarına ulaşmak amacıyla veri ve operasyonlara sahip bir problem-alanı bileşenidir.
- **View**—Bir veri gösterim bileşenidir.
- **Controller**—Kullanıcı girdilerini alan ve bunlara karşı aksiyonda bulunan bir bileşendir



MVC Statik Yapısı



MVC Davranışı



MVC Avantajları

- Model bileşenini hiç etkilemeden View ve Controller bileşenleri eklenebilir, çıkartılabilir, veya değiştirilebilir.
- Programın çalışması sırasında View bileşenleri eklenebilir veya değiştirilebilir.
- Çalışma zamanında (runtime) bile kullanıcı arayüzü bileşenleri değiştirilebilir.

MVC Dezavantajları

- View ve Controller bileşenlerini ayırmak genellikle zordur.
- Sık güncellemeler (yani bileşenler arasındaki etkileşimler) veri gösterimini yavaşlatabilir ve kullanıcı arayüzünde performans sıkıntısı oluşturulabilir.
- MVC stili kullanıcı arayüzü bileşenlerini model bileşenlerine çok fazla bağımlı hale getirebilir.

Hibrit Mimariler

- Çoğu sistem çeşitli mimari stillerini, genellikle farklı soyutlama seviyelerinde, bir arada içerir.
 - Sistemin genelinde Katmanlı stil kullanılırken katmanların birinde Olay-Güdümlü, diğerinde ise Paylaşımlı-Veri stili kullanılabilir.
 - Sistemin genelinde Pipe-and-Filter stili kullanılırken, filtre bileşenleri Katmanlı stilde oluşturulmuş olabilir.

Özet

- Katmanlı stilinde, program bileşenleri katmanlara ayrılmıştır ve her bir katman yalnızca altındaki katman(lar)ı kullanacak şekilde sınırlandırılmıştır.
- Pipe-and-Filter stilinde bileşenler kanallarla (pipe) bağlanmış filtreler (filter) şeklindedir.
- Paylaşılan-Veri stilinde bileşenler bir veya daha fazla paylaşılan-veri erişimcisi tarafından manipüle edilen bir veya daha fazla paylaşılan-veri deposu olarak modellenir.
- Olay-Güdümlü stilinde program bileşenleri bir olay hareket memuruna kaydolurlar, ve olay bildirimleri ve olaydan haberdar olmalar bu hareket memuru aracılığıyla gerçekleşir.
- Model-View-Controller stilinde kullanıcı arayüzü View ve Controller bileşenleri problem-alanı Model bileşenleriyle haberleşirler.
- Mimariler genellikle farklı soyutlama seviyelerinde birden çok stilden oluşurlar. Bu tür mimarilere hibrit mimariler adı verilir.

Kaynaklar

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme

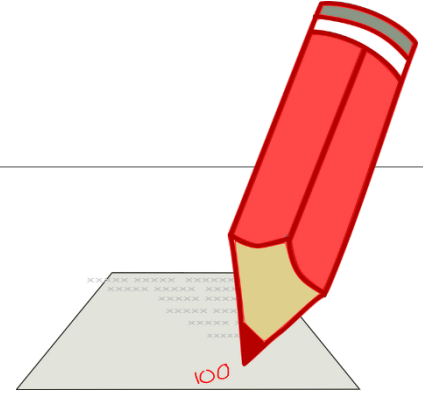
http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk

<http://dsdmofagilemethodology.wikidot.com/>

<http://caglarkaya.piqestion.com/2014/07/01/244/>

Ödev

?



Sorularınız





YMH311-Yazılım Tasarım ve Mimarisi

Sunucu Mimarileri

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

İstemci Sunucu Mimarileri.....5

Sunucu Türleri.....11

WWW Standartları.....20

Katmanlı Mimariler.....21

Amaçlar

- İstemci ve Sunucu Arasındaki Farkları Kavramak
- Sunucu Türlerini Öğrenmek
- Web Sunucusu Yapısını Kavramak
- WWW Standartlarını Öğrenmek
- Yazılım Geliştirmede Katmanlı Mimari Yapılarının Avantajlarını Görmek
- Katmanlı Mimarilerin Neler Olduğunu Öğrenmek
- İki Katmanlı ve Üç Katmanlı Mimari Arasındaki Farkları Tartışmak
- WAP Mimarisi Hakkında Bilgilenmek



GİRİŞ

- 1980'li yılların ortalarına kadar ana bilgisayarların yoğun olarak kullanıldığı dönemde, yazılımların hemen hemen tümüyle ana bilgisayarda olduğu, kullanıcıya ilişkin tüm denetim ve paylaşım işlemlerinin yine ana bilgisayarlar üzerinde yapıldığı gözlenmektedir.
- Grafikselle kullanıcı arabirimi desteklenmemekte ve "akılsız" uçlar kanalıyla bilgisayara bağlantı yapılabilmektedir.
- Programlamanın günümüze göre daha kolay olduğu bu yapılarda, coğrafik olarak birbirinden ayrı bulunan veri tabanlarına erişim ve işlem oldukça sınırlı olmuştur.
- "Ana çatı" ya da "ana bilgisayar" mimarileri olarak adlandırılan bu mimarileri, 80'li yılların ortalarında "kütük paylaşım mimarileri" izlemiştir.
- Bu mimarilerde ise bilgiler bir sunucu üzerinde bulunmakta ve kullanıcılar kendi mikrobilgisayarları üzerinde iş yaparak bu bilgileri paylaşmaktadır.
- Gerek "ana bilgisayar" gerekse "kütük paylaşımı" mimarileri günümüzde artık kullanılmamakta ve "geleneksel" mimariler olarak adlandırılmaktadır. Bu nedenle bu bölümde bu mimarilere yer verilmemiştir.
- Günümüzde yaygın olarak kullanılan istemci/sunucu mimarileri ve yeni teknolojiler olan web mimarisi ve bileşen tabanlı mimariler bu bölümde daha geniş olarak incelenmektedir.

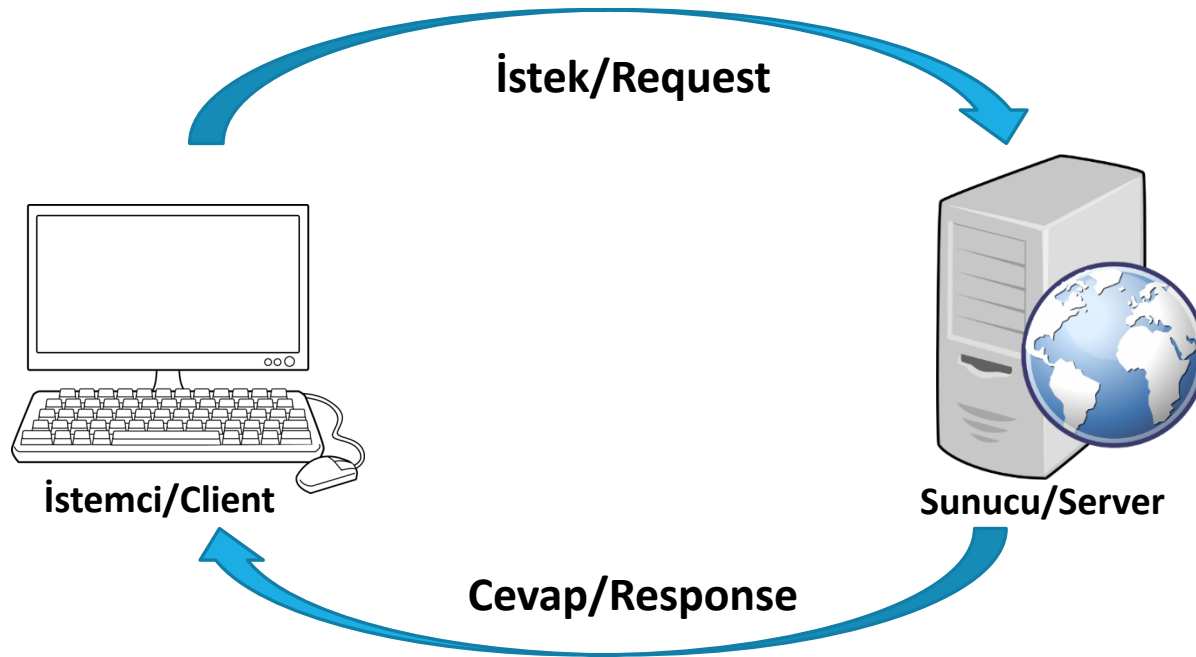
İstemci Sunucu Mimarileri

- Dağıtık bilgi işleme ortamlarının gelişmesinin etkisiyle, istemci sunucu (İ/S) mimarileri yaygınlaşmıştır.
- İstemci sunucu mimarisinde, yerel alan ağları, geniş alan ağı dağıtık veri işleme ve yönetim ve açık sistem ortamları bulunmaktadır.
- Yalın tanımıyla İ/S mimarisinde, sunucu yazılımı, istemci yazılımdan gelen istekleri yerine getirerek sonucu istemciye iletir.
- İstemci sunucudan gelen sonuçları değiştirerek ya da olduğu gibi kullanıcıya sunar.
- İ/S mimarilerinde temel etmen, donanım değil yazılımdır.
- Geleneksel mimarilerin tersine, uygulama yazılımı, istemci ve sunucu bilgisayarlarda çalıştırılmak üzere her iki platform için geliştirilir.
- İstemci ve sunucu arasındaki veri iletişimi ve denetim ilişkisi yazılım aracılığı ile yönetilir.
- İ/S mimarileri değişik yapılarda ve ölçeklerde oluşturulabilir.

İstemci Sunucu Mimarileri

- Küçük bir yerel alan ağı üzerinde tek sunucu ile çalışabilecek İ/S mimarileri oluşturulabileceği gibi, binlerce kullanıcıyı yüzlerce sunucuya bağlayan ve geniş alan ağları üzerinde çalışan e-posta servislerini içeren mimariler oluşturulabilir.
- Ölçeği ne olursa olsun tüm istemci sunucu mimarisindeki yazılımların ortak özellikleri aşağıda belirtilmektedir.
 - Kullanıcılar genel olarak bir yerel ağ üzerinde bulunan sunucularla çalışırlar. Dolayısıyla tipik bir İ/S mimarisi varsa geniş alan ağına iletişim sunucusu ile bağlı bir yerel ağ üzerinde kurulur.
 - Standart bir İ/S mimarisi yoktur, ancak standart İ/S bileşenleri vardır.
 - Sunucular genellikle belli işlere atanmış güçlü iş istasyonlarından (örneğin kütük sunucuları, yazıcı sunucuları vb) oluşturulur.
 - İstemciler ise kişisel bilgisayar sınıfındaki bilgisayarlardır. El bilgisayarları, akıllı telefon gibi istemciler giderek yaygınlaşmaktadır.
 - Tipik yerel ağ bileşenleri ise, köprüler, yönlendiriciler, alıcılar, öbekler ve geçitlerden oluşur.
 - Genellikle her sunucu için gerekli kullanıcı sayısı 10 ile 50 arasında değişir.
 - Daha geniş ağları oluşturmak amacıyla alt ağlar oluşturma biçiminde tabakalama yapılır.

İstemci Sunucu Örnek



İstemci

- İstemci, sunucuya bir iş yapmak ya da servis sağlamak üzere ileti gönderen bir süreç ya da yazılım olarak tanımlanabilir.
- İstemci programları genellikle uygulamanın kullanıcı arayüzünü yönetirler.
- Kullanıcı tarafından girilen verinin doğrulanması ve kullanıcı isteğinin sunuculara ayrıştırılması gibi işlemler İstemci yazılımları aracılığı ile yapılır.
- Özetle, bir uygulama yazılımının kullanıcı arayüzü ile ilgili yazılım parçaları istemciler üzerinde çalışır.
- İ/S mimarisinin gelişmesi, kullanıcıya, kişisel bilgisayar tabanlı istemciler kullanmasından dolayı grafiksel arayüz olanakları sağlamıştır.

Sunucu

- İstemci tarafından gerekenen isteği yerine getiren süreç ya da yazılımlar sunucu yazılımları olarak adlandırılır.
- Sunucu, istemci isteğini alır, gerekli işlemleri yapmak amacıyla veri tabanlarında erişim ve güncleme yapar.
- Sonucu istemciye iletir. Sunucu yazılımları ağ üzerinde bir başka bilgisayarda çalışır.
- Sunucu, bir ana bilgisayar işletim sistemi, kütük sunucusu ya da uygulama sunucusu olabilir.
- Sunucular, uygulamanın arka yüzdeki işlemlerini yapar.

Sunucu

- Tipik bir sunucu yazılımının özellikleri aşağıda belirtilmektedir.
 - Sunucu yazılımı zamanının büyük bir kısmını istemciden gelecek istekleri beklemekle geçirir. Söz konusu istekler, iletişim oturumu aracılığı ile ve ileti biçiminde ulaştırılır. Bazı sunucular her istemci için ayrı oturum atarken, diğerleri de tekrar kullanılabilir oturumlar için dinamik atama yaparlar. Sunucu için başarılı olmanın ön koşulu, istemcilere her zaman yanıt verebilmesidir.
 - Sunucu, istemcinin isteğini yerine getirmek durumundadır. Ancak istemci yalnızca bir sunucuya bağlı kalmayabilir. Öte yandan bir sunucu da eş zamanlı olarak birden fazla istemciye servis verebilir.
 - Sunucular, istemcilere değişik düzeylerde servis öncelikleri verebilir.
 - Sunucu, uygulama yazılımının içeriğinden farklı olarak, arka planda başka işlemler yapar.
 - Sunucu yazılımı tipik bir görev-kritik yazılımdır. Sunucu durduğunda, bağlı tüm istemciler etkilenir. Bu nedenle sunucu yazılımının çalışacağı ortam çok güvenli olmalıdır.
 - Sunucu ortamı, ölçeklenebilir, yükseltilebilir ve modüler olmalıdır.

Sunucu Türleri

➤ Sunucular için farklı sınıflandırmalar yapılabilir.

➤ Bu bölümde:

1. Kütük Sunucuları
2. Veri Tabanı Sunucuları
3. Ara İşlem Sunucuları
4. Nesne Sunucuları
5. Web sunucuları

biçiminde bir ayırım verilmektedir.



Kütük Sunucuları

- Bilgisayar ağı üzerindeki tüm kütüklerin paylaşımı, kütük sunucuları aracılığı ile yapılır.
- Veri servisinin en temel biçimi, istemcinin isteğini kütük sunucu aracılığı ile ağa iletmesi ve sunucunun isteği inceleyerek ağ üzerinde ileti değişimi yoluyla istenen veriye ulaşmasının sağlanmasıdır.
- Yerel ağ üzerindeki her türlü belge, resim, veri kütüğü vb. nesnelerin paylaşımı kütük sunucuları aracılığı ile yapılır.

Veri Tabanı Sunucuları



- İstemci bir SQL komutu ya da komut kümesini veri tabanı sunucusuna iletir.
- Her SQL komutunun sonucu, ağ üzerinden istemciye iletilir. SQL isteğini işleyen kod aynı bilgisayarda bulunur.
- Sunucu, ileti aktarma işlemleri ile uğraşma yerine, işlem gücünü veri tabanından istenen verilerin bulunması ve işlenmesi işlemleri ile uğraşır.
- SQL sunucuları İ/S uygulamaları oluşturmak için iyi bir örnektir.
- Oracle, Sybase gibi bir çok veri tabanı yönetim sistemi geliştiricileri için SQL standart veri tabanı dili haline gelmiştir.

Ara İşlem Sunucuları

- İstemcinin uzakta bulunan yordamları çağırması durumunda bu yordamların çalıştırıldığı sunucular ara işlem sunucu olarak adlandırılır.
- Uzakta bulunan yordamlar bir grup SQL deyimi çalıştırırlar. Bu SQL deyimleri, ara işlem olarak tanımlanır.
- Ara işlem sunucu ile, hem istemci hem de sunucu tarafında yazılımlar hazırlanarak uygulama geliştirilebilir.
- İstemci tarafındaki yazılım genellikle grafiksel kullanıcı arabirimi içerir.
- Sunucu yazılımı ise bir veritabanı üzerinde çalışacak SQL deyimlerinden oluşur.
- Bu uygulamalar **çevrim içi ara işlem işleme (ÇİAİ-OLTP)** olarak adlandırılır.
- ÇİAİ uygulamaları, veri tabanı üzerinde sıkı denetim, güvenlik ve tamlık gerektirir. İki tür ÇİAİ uygulaması vardır:

Ara İşlem Sunucuları

- Kritik ÇİAİ uygulamalarını etkin ve iyi yönetebilmek için ara işlem yöneticileri kullanılır.
 - Ara işlem yöneticisi, ara işlemlerin yönetimi ve işlenmesi için bir "işletim sistemi" olarak tanımlanabilir.
 - Temel olarak;
 - Süreç yönetimi (sunucu süreçlerinin başlatılması, yönetimi ve iş yükünün dengelenmesi) ve
 - Ara işlem yönetimi
- işlevlerini yerine getirirler.

Nesne Sunucuları

- İ/S uygulaması, birbiri ile iletişimde bulunan bir nesne kümesi olarak geliştirilir.
- İstemci nesneleri, sunucu nesneleri ile bir Nesne İstek Yöneticisi (NİY - ORB) aracılığı ile haberleşir.
- İstemci, uzakta bulunan bir nesneye ilişkin bir metodu çalıştırmak istediğinde NİY, ilgili nesne sınıfına ilişkin varlık oluşturur, istenen metodu çağırır ve sonucu istemci nesnesine iletir.
- Sunucu nesneleri, paylaşım ve eşzamanlılığı desteklemelidirler.

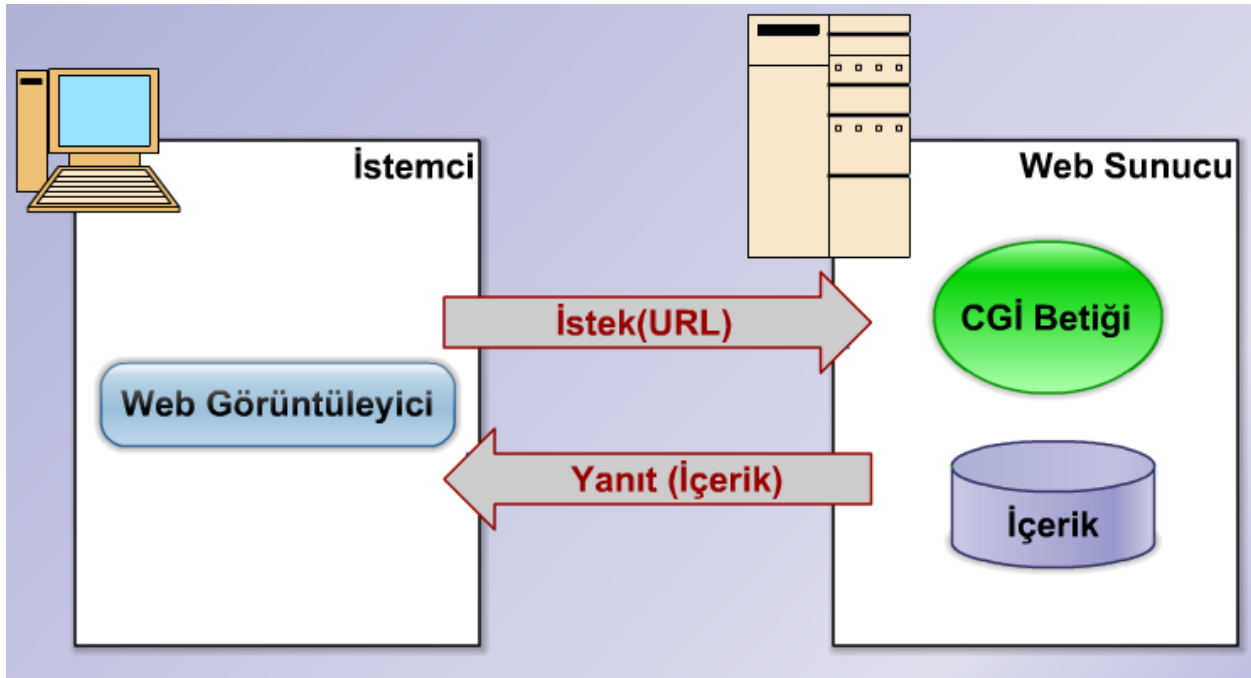
Ticari NİY'lere örnek olarak Digital ObjectBroker, IBM SOM 3.0, Sun's NEO, HP ORB Plus, Expersoft's PowerBroker, ve Iona's Orbix verilebilir.

Web Sunucuları



- Dünya çapındaki Web, ilk galaksiler arası İ/S uygulaması olarak tanımlanabilir.
- Bu yeni model zayıf, taşınabilir ve "**evrensel**" istemcilerin çok şişman sunucularla iletişimini sağlar.
- En yalın biçimiyle bir web sunucu, istemcinin ismiyle aradığı bir belgeyi kendisine sunar.
- Web, küresel bir hipertekst sistemidir.
- Hipertekst, aynı bilgisayar yada bir bilgisayar ağı üzerinde bulunan ilgili belgeleri birbirine bağlayan bir yazılım sistemidir.
- İstemci ve sunucular, adına Hypertext Transfer Protokolü (HTTP) denilen Uzak Yordam Çağırma (UYÇ -RPC Remote Procedure Call) benzeri bir protokol aracılığı ile iletişim kurar.

Web Sunucuları



Web Sunucuları

HTTP:

1. İ/S bağlantısı kurar,
 2. Parametreleri gönderir ve alır,
 3. İ/S bağlantısını kapatır.
- HTTP istemci ve sunucuları ileti içeriklerini göstermek amacıyla internetin MIME veri gösterim yöntemini kullanır.



WWW Standartları



- Genel amaçlı uygulamaların geliştirilmesi amacıyla oluşturulan www standartları geliştirilmiştir.
- Bu standartlar şu şekildedir;
 - **Standart isimlendirme Modeli:** Bütün sunucuların isimlendirilmesi ve içeriği için internet Tek Kaynak Yerleştirici (URL-Uniform Resource Locator) standardı (RFC1738,RFC1808)
 - **İçerik Türü:** WWW üzerindeki her tür içerik, web görüntüleyicilerin işleyebileceği özel bir türe sahiptir (RFC2045, RFC2048)
 - **Standart İçerik Biçimi:** Bütün web sunucuları bir dizi standart içerik biçimini destekler. Bunlar arasında Hiper Metin Biçimlendirme Dili (HTML-Hypertext Markup Language), JavaScript betik dili(ECMA Script, Javascript) en yaygın olanlarıdır.
 - **Standart Protokol:** Web görüntüleyicilerin herhangi bir web sunucu ile iletişimini sağlayan ağ protokol standartları arasında en yaygın olan, HiperText Transfer Protokolüdür (HTTP).

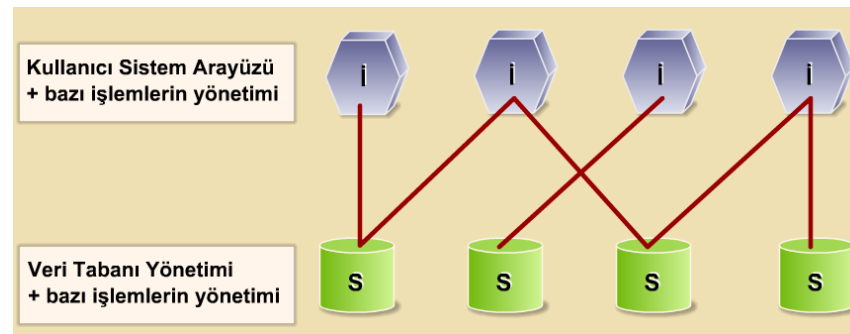
İki Katmanlı Mimariler

➤ İki katmanlı mimarilerde, uygulamanın üç temel bileşeni iki katmanda bulunur.

➤ Bu bileşenler:

- Kullanıcı Arayüzü (Grafiksel Kullanıcı Arabirimi, görüntü yönetimi),
- İşlem yönetimi (ya da uygulama mantığı) ve
- Veritabanı yönetimi (veri ve kütük servisleri)

biçimindedir.



İ: İstemci

S: Sunucu

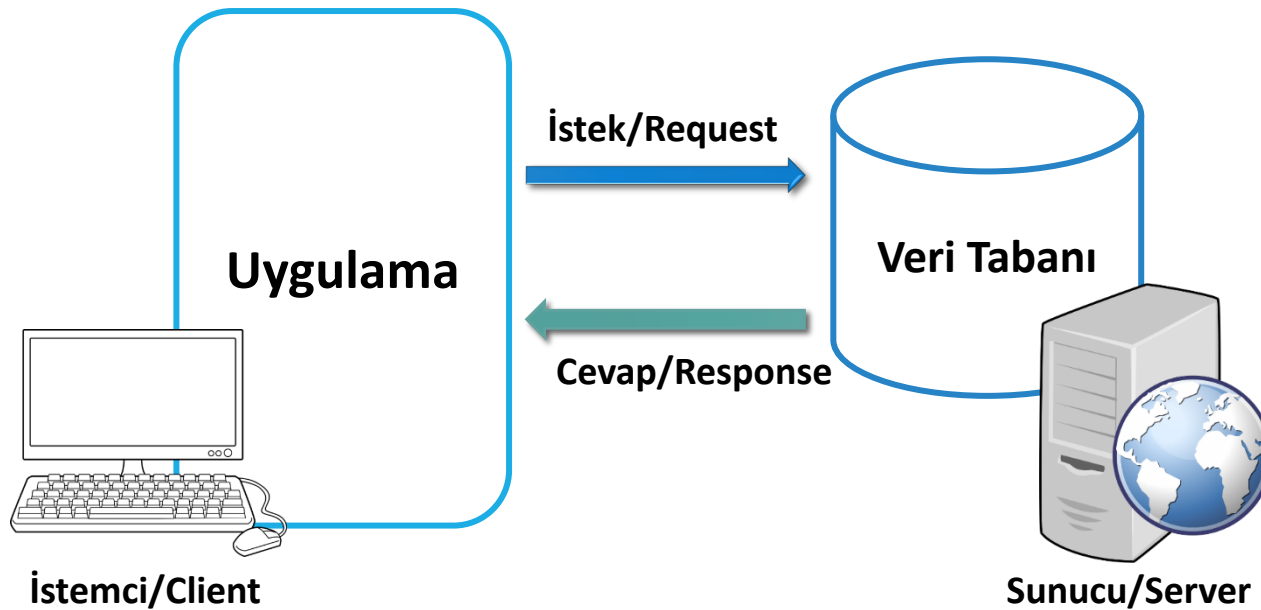
İki Katmanlı Mimariler

- Kullanıcı arayüzü istemci üzerinde bulunur. Veri tabanı yönetimi ise sunucu üzerine yerleştirilir. İşlem yönetimi ya da uygulama mantığı ise istemci ve sunucu arasında paylaştırılır.
- Birçok iki katmanlı tasarımda, uygulamanın büyük bir kesimi sunucu ortamında bulunur. Bu yolla esneklik ve ölçeklenebilirlik arttırılır.
- Bu mimari, homojen ortamlarda başarılı sonuçlar verir.
- İki katmanlı mimariler, 4. kuşak programlama platformlarının da desteği ile, günümüzde, endüstride oldukça yaygınlaşmış ve başarılı bir biçimde uygulanmaktadır.

İki Katmanlı Mimariler

- İki katmanlı mimari, bilgisayar ağı ve sunucunun yeteneklerine göre belirli bir sınıra kadar ölçeklenebilir.
- 100 kullanıcıya kadar gözlemlenmiş uygulamalar bulunmaktadır. Bunun nedeni, bilgisayar ağının belirli bir yoğunluğa ulaşması ve sunucu üzerindeki işlem yoğunluğudur.
- Hiçbir veri değişimi olmazsa bile yalnızca ağ üzerinde dolaşan "istemcileri canlı tutma" iletileri belirli bir yoğunluğa neden olur.
- Sunucu üzerinde işlem yapan her istemci bir darboğaz oluşturmaya neden olabilir.
- Veri tabanı sunucusu üzerinde geliştirilen uygulama mantığı, birlikte çalışılabilirliği azaltır. Çünkü kod, ilgili VTYS dili ile yazılmıştır ve ticari veri tabanları için tanımlanmış standart bir VTYS dili yoktur. Bu nedenle kod bakımı zorlaşır.
- Öte yandan bu diller, standart programlama dilleri kadar yetenekli değildir. İstemci tarafında, sunucuya erişim amacıyla, SQL ya da UYÇ kullanılır.
- Yapılacak her değişikliğin tüm istemcilere yansıtılması gerekliliğinden dolayı, istemci tarafının yönetimi ve bakımı kolay değildir. Ayrıca, bu mimari toplu iş düzeni için uygun değildir.

İki Katmanlı Mimari Örnek

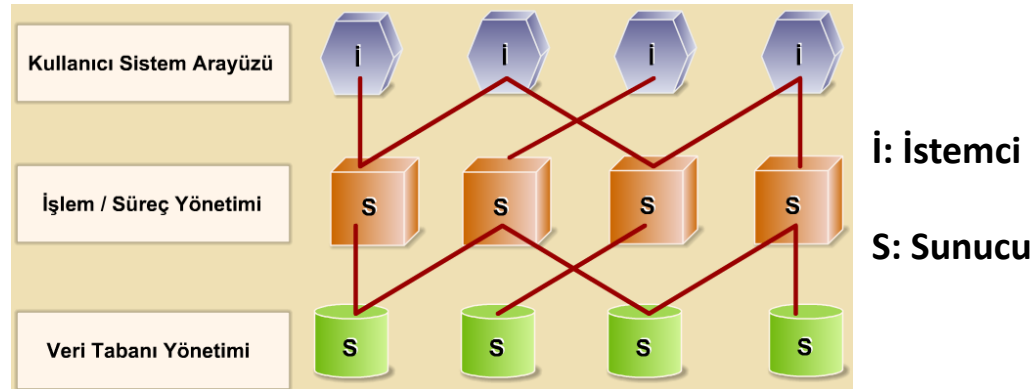


Üç Katmanlı Mimariler -1

Üç katmanlı mimarilerde, uygulamanın üç temel bileşeni, üç bağımsız katmanda bulunur.

Bu bileşenler:

- Kullanıcı (sistem) Arayüzü,
- İşlem Yönetimi (Uygulama Mantığı) ve
- Veri tabanı servisleri



Üç Katmanlı Mimariler -2

- Üç katmanlı mimarilerde veri sunucusu yalnızca veritabanı bilgilerini içerir ve uygulama mantığı ile ilişkisi yoktur.
- Bu nedenle uygulama mantığına ilişkin herhangi bir VTYS dili kodlaması bu sunucu için gerekmez.
- Bu mimarilerde ara katman sunucusu olan uygulama sunucusu, uygulama mantığını tümüyle içerdiğinden, bakım kolaylığı, esneklik ve ölçeklenebilirlik sağlanır.
- Değişiklik yönetimi de kolaylaşır. Uygulama sunucusu üzerinde yapılan değişiklikler istemcileri etkilemez.
- Buna ek olarak, üç katmanlı mimariler güvenilir işlem yapma olanağı sağlar.
- Üç katmanlı mimarilerde, iki katmanlı mimarilerin tersine yüzlerce kullanıcının desteklenmesi olasıdır. Heterojen yapılar için uygundur. Bu mimari 90'lı yılların başından bu yana kullanılmaktadır.

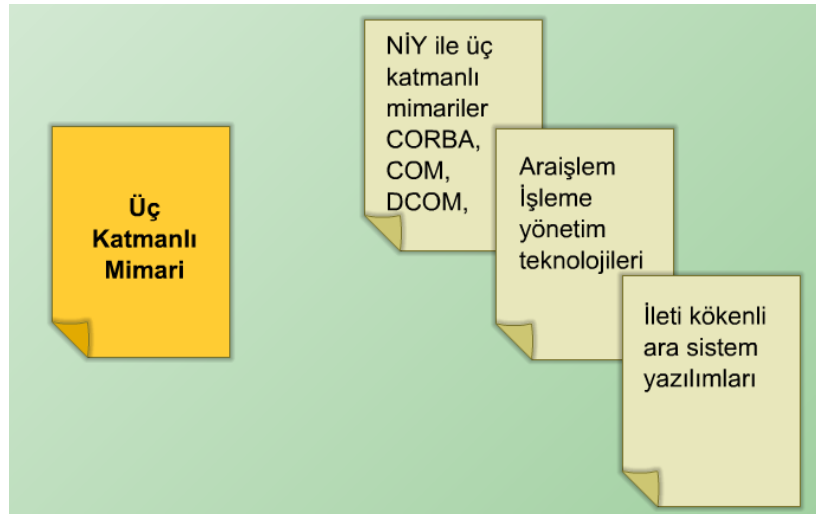
Üç Katmanlı Mimariler -2

- En büyük zorluklardan biri, yeterince destekleyen yazılım aracının bulunmayışı ve karmaşıklığıdır.
- Günümüzde geliştirilmiş olan yazılım geliştirme yöntem ve metodoloji çoğu iki katmanlı İ/S sistemlerine yöneliktir.
- Diğer bir zorluk ise, bu üç katmanın ayrıştırılmasında oluşmaktadır. Bu konuda kullanılabilecek ayrıştırma kriterlerinin bazıları:
 - Geliştirme ve sınama kolaylığı,
 - Yönetim kolaylığı,
 - Sunucuların ölçeklenebilirliği ve
 - Performans

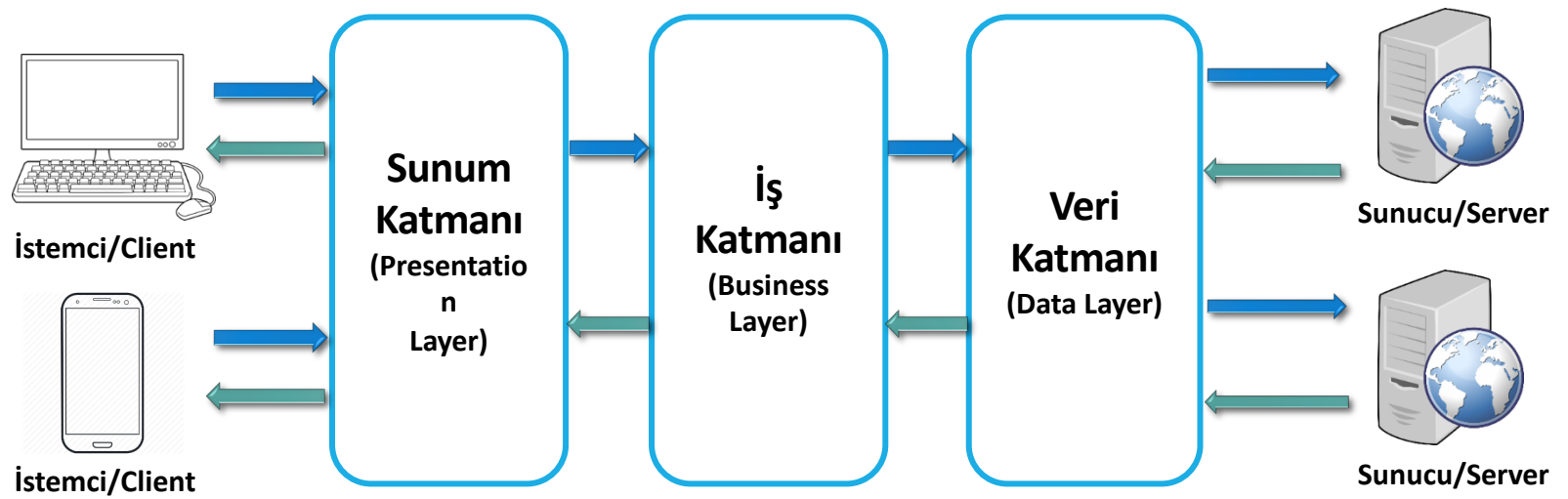
olarak verilebilir.

Üç Katmanlı Mimariler -2

- Üç katmanlı mimari üzerinde geliştirilmiş alt yapı yazılımları uygulamalarına örnek olarak aşağıdakiler gösterebilir.



Üç Katmanlı Mimari Örnek



İki ve Üç Katmanlı Mimarilerin Karşılaştırması

- İki katmanlı mimarilerde, uygulama mantığı ve veri tabanının aynı bilgisayarda yer alması, sunucu tarafında geliştirilen uygulamanın VTYS'nin kendi dilinde olması nedenleriyle birlikte çalışılabilirlik azalmaktadır.
- Üç katmanlı mimarilerde ise, ara bir katmanın oluşu bu sınırlamayı kaldırmaktadır.
- Uygulama mantığının bu ara katmanda oluşu, istemciyi veri tabanı sunucusundan bağımsız duruma getirmektedir.

İki ve Üç Katmanlı Mimarilerin Karşılaştırması

Kriter	İki Katmanlı Mimari	Üç Katmanlı Mimari
Birlikte Çalışabilirlik	Sınırlı	Yüksek
Taşınabilirlik	Az taşınabilir	Daha fazla taşınabilir
Yeniden Kullanılabilirlik	Sınırlı	Yeniden Kullanılabilir
Karmaşıklık	Yalın	Karmaşık
Bakım	Zor	Kolay
Geliştirme Süresi	Daha az başlangıç geliştirme süresi	Daha az ardışık geliştirme süresi
Ölçeklenebilirlik	100 kullanıcıya kadar	1000 kullanıcıya kadar

İki ve Üç Katmanlı Mimarilerin Karşılaştırma Kriterleri

- İki katmanlı mimarilerde, **taşınabilirlik** bir bakıma sınırlıdır.
- Mevcut iki katmanlı mimari uygulamaları, yordamlara ilişkin kodları tekrar üretmeden bir başka ortama taşınamamaktadır.
- Üç katmanlı mimarilerde ise, ara katman aracılığı ile bu sorun çözülmektedir.
- Ara katman yazılımının modüler biçimde geliştirilmesi, herhangi bir sorunla karşılaşmaksızın sunucu kesiminde ya da istemci kesiminde herhangi bir zorlukla karşılaşılmaz.
- İki katmanlı uygulamalarda **yeniden kullanılabilirlik** ana hedef değildir ve oldukça sınırlıdır.
- Üç katmanlı mimariler ise değişik açılardan yeniden kullanılabilirliği sağlarlar.
- İlk olarak, ara katmanda geliştirilmiş olan uygulama mantığı yazılımı fazla değiştirilmeden başka uygulamalarda yeniden kullanılabilir. Diğer taraftan NİY modeli, nesne kökenli yaklaşımla geliştirilmiş uzak nesnelerin yeniden kullanılmasını olanaklı kılar.

İki ve Üç Katmanlı Mimarilerin Karşılaştırma Kriterleri

- **Karmaşıklık** açısından incelendiğinde, iki katmanlı mimarilerin, üç katmanlı mimarilere oranla daha yalın olduğu söylenilebilir.
- En azından bu yalınlığın nedeni üç katman yerine iki katman bulunmasıdır.
- Buna ek olarak, uygulama geliştirme iki katmanlı mimarilerde daha kolaydır.
- Üç katmanlı mimari yazılımlarının geliştirilmesine yönelik araç, yöntem ve metodolojilerin azlığı da bu mimariler için bir başka karmaşıklık nedenidir.
- **Bakım** açısından bakıldığında ise üç katmanlı mimarilerin iki katmanlı mimarilere oranla daha kolay bakılabilir olduğu açıktır.
- Bu kolaylık ara katman sayesinde sağlanır.

i/S Mimarisinin Yararları ve Aksak Yönleri

İstemci sunucu mimarisi yazılımlarının yararları aşağıda belirtilmektedir.

- Gelişen masaüstü teknolojisinin etkin olarak kullanılmasını olanaklı kılar. Bu yolla maliyetleri azaltır.
- Varolan altyapı ve aygıtları kullanarak kurumların yapılarını korur.
- İşlem ve işlenecek verinin birbirine yakın olmasını sağlayarak ağ trafiğini azaltır ve yanıt süresini kısaltır. Ağ için gerekli bant genişliği düşer.
- Anabilgisayar mimarileri yazılımlarına oranla yazılım geliştirme süresi kısalır. Önyüz ve arka yüz işlemleri ayrı olduğundan bakım kolaylaşır.
- Grafikselle Kullanıcı Arayüzünü destekler. Böylelikle eğitim için gerekli yatırım maliyetlerini azaltır.
- Açık sistem mimarisini destekler. Bu yolla kullanıcılara marka esnekliği sağlar.
- Yeni yazılım bileşenlerinin eklenmesi oldukça kolaydır.

i/S Mimarisinin Yararları ve Aksak Yönleri

i/S mimarisinin aksak yönleri ise:

- Uygulama mantığının büyük ölçüde sunucu üzerinde gerçekleştirilmesi durumunda darboğaz oluşabilir.
- Özellikle, birlikte çalışabilecek biçimde tasarlanmış dağıtık uygulamalar, dağıtık olmayan uygulamalara oranla çok daha fazla karmaşıktır.
- Varolan geliştirme metodolojileri ve yöntem ve araçlar sınırlıdır.

biçiminde özetlenebilir.

Sonuç olarak i/S mimarileri yazılım bakım maliyetlerini düşürür, yazılım taşınabilirliğini arttırır ve ağ performansını etkinleştirir.

Bileşen-Tabanlı Mimariler

- Günümüzde, internet kullanımının giderek yaygınlaşması ve elektronik ticaret, e-iş kavramlarının uygulamaya geçirilmesi, uygulama ortamlarının dağıtık ve heterojen yapıya dönüşmesi sonucunda "**yeniden kullanılabilirlik**" olgusu giderek önem kazanmış ve bileşen tabanlı mimari kavramı gündeme gelmiştir.
- Donanım üreticilerinin yıllardır standart bileşenler üretilip bu bileşenleri birleştirilerek ürün oluşturdukları bilinmektedir.
 - Örneğin, bir otomobil, aynı ya da farklı üreticilerden edinilen parçaların bir araya getirilmesi ile oluşturulabilir.
- Günümüzde otomobil üretimi büyük ölçüde bu biçimde yapılmaktadır. Donanım üreticilerinin bu üretim tarzını yazılım üretimine uygulamaya çalıştığımızda "**Bileşen Tabanlı**" yazılım mimarileri ortaya çıkar.
- Yazılım bileşeni, "**çalışan yazılım yapıtaşları**" ya da işletim zamanında birlikte çalışmak üzere bütünleştirilebilen yazılım parçası olarak tanımlanmaktadır.

Bileşen-Tabanlı Mimariler

- Küçük boyutlu yazılım teknolojilerine örnek olarak: java Beans, OLE/COM ve DCOM verilebilir.
- Bir yazılım bileşeni, üç temel parçadan oluşur: **Kullanıcı Arayüzü, Yöntemler ve Amaç Kod.**
- Arayüz, bileşenin nasıl kullanacağını; yöntemler, bileşenin işlevselliğini; amaç kodu ise bileşenin doğrudan çalışabilir kodunu içerir.
- Bir bileşen, **açıklayıcı, içerilebilir, değiştirilebilir ve genişletilebilir** özellikleri içermelidir. Bu dört özellik bir bileşen oluşturmak için sağlanması zorunlu koşullardır.
- Öte yandan, bir bileşen uygulama, dil ya da platform ile sınırlı değildir.
- Bileşen tabanlı mimarilerde, ayrıntı tasarım daha az önem kazanmakta, çalışma daha çok genel tasarım ve arayüz tasarımı üzerinde yoğunlaşmaktadır.

Bileşen Tabanlı Yazılım Geliştirmede Kullanılan Metodolojiler

Bileşen tabanlı yazılım geliştirmede kullanılan metodolojiler:

- Bileşenin destekleyeceği iş fonksiyonlarına dayalı sistem gereklerinin açık tanımlamaları,
- Sistemin her süreç için davranış çözümlemesi,
- Arayüz tanımları ve aralarındaki ilişkiler,
- Birleştirme, yeniden kullanım ve sistem iş akış kodlarının ayrıntılı tanımları

üzerinde yoğunlaşırlar.



Özet

İstemci: Sunucuya bir iş yapmak ya da servis sağlamak üzere ileti gönderen bir süreç ya da yazılım olarak tanımlanabilir.

Sunucu: İstemci tarafından gerekenen isteği yerine getiren süreç ya da yazılımlar sunucu yazılımları olarak adlandırılır.

Sunucu Türleri:

1. Kütük Sunucuları
2. Veri Tabanı Sunucuları
3. Ara İşlem Sunucuları
4. Nesne Sunucuları
5. Web sunucuları

şeklindedir.

HTTP:

1. İ/S bağlantısı kurar,
2. Parametreleri gönderir ve alır,
3. İ/S bağlantısını kapatır.



Özet

WWW standartları şu şekildedir.

- **Standart isimlendirme Modeli:** Bütün sunucuların isimlendirilmesi ve içeriği için internet Tek Kaynak Yerleştirici (URL-Uniform Resource Locator) standardı (RFC1738,RFC1808)
- **İçerik Türü:** WWW üzerindeki her tür içerik, web görüntüleyicilerin işleyebileceği özel bir türe sahiptir (RFC2045, RFC2048)
- **Standart İçerik Biçimi:** Bütün web sunucuları bir dizi standart içerik biçimini destekler. Bunlar arasında Hiper Metin Biçimlendirme Dili (HTML-Hypertext Markup Language), JavaScript betik dili(ECMA Script, Javascript) en yaygın olanlarıdır.
- **Standart Protokol:** Web görüntüleyicilerin herhangi bir web sunucu ile iletişimini sağlayan ağ protokol standartları arasında en yaygın olan, HiperText Transfer Protokolüdür (HTTP).

İki katmanlı mimarilerde, uygulamanın üç temel bileşeni iki katmanda bulunur. Bu bileşenler: Kullanıcı Arayüzü (Grafiksel Kullanıcı Arabirimi, görüntü yönetimi), İşlem yönetimi (ya da uygulama mantığı) ve Veritabanı yönetimi (veri ve kütük servisleri)'dir. Üç katmanlı mimarilerde, uygulamanın üç temel bileşeni, üç bağımsız katmanda bulunur. Bu bileşenler: Kullanıcı (sistem) Arayüzü,İşlem Yönetimi (Uygulama Mantığı) ve Veri tabanı servisleridir.

Çalışma Soruları

1. Yazılım Mimarisi terimini açıklayınız.
2. Yazılım mimarilerini sınıflandırınız.
3. Geleneksel yazılım mimarileri ile istemci sunucu mimarileri arasındaki farklılıkları belirtiniz.
4. Çevrim içi araışlem işlemeyi tanımlayınız.
5. 2-katmanlı mimari ve 2-katmanlı mimarileri karşılaştırınız.
6. 2 katmanlı ve 3 katmanlı mimarilere uygun olmayan uygulama örnekleri veriniz.
7. Katmanlı mimarilerin uygulama alanlarını belirtiniz.
8. Ağ trafiği üzerindeki etkileri açısından iki ve üç katmanlı İ/S mimarilerini karşılaştırınız.
9. Web mimarisinde kullanılan standartları belirtiniz. Standart kullanımının önemini açıklayınız.
10. Bileşen tabanlı uygulama mimarisini açıklayınız.
11. Bileşen tabanlı uygulama mimarisi ile istemci sunucu mimarisini kıyaslayınız.
12. Bileşen ile nesne arasındaki benzerlik ve farklılıkları belirtiniz.

Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- ” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- ”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Sorularınız

