

YAZILIM MÜHENDİSLİĞİNE GİRİŞ

HAFTA 10 YAZILIM MÜHENDİSLİĞİ

Dr. Öğr. Üyesi Ebru Aydoğan
eaydogan@mehmetakif.edu.tr

Ders İçeriği: Hafta 10

- Yazılım Mühendisliği Bilim Dalı
- Yazılım Yaşam Döngüsü
- Yazılım Mühendisliği Metodolojileri
- Modülerlik
- İş Araçları
- Kalite Güvencesi
- Dokümantasyon
- İnsan-Makine Arayüzü
- Yazılım Sahipliği ve Yükümlülük

➤ Yazılım Nedir?

- Tanımlanmış bir işlevi yerine getiren,
- Girdi ve çıktıları olan,
- Herhangi bir donanım üzerinde çalışan,
- Bilgisayar programı veya programlarını barındıran,
- Kullanım ve bakım kılavuzları gibi belgelerden oluşan bir üründür.

Yazılım Mühendisliği Bilim Dalı

➤ Yazılım Tasarımının Komplikasyonları

- Yazılım hataları büyük ölçekli felaketlere neden olmuştur.
- Yazılım mühendisliği → Güvenilir bir yazılım sistemi geliştirmenin ve sürdürmenin daha iyi bir yolunu bulmaya çalışır
- Yazılım Mühendisliği diğer mühendislik disiplinlerinden farklıdır:
 - Önceden üretilmiş bileşenler (hazır parçalar) eksiktir.
 - Gereken performans mükemmel olmalıdır.
 - Kalite metrikleri eksiktir ve tam tanımlı değildir.
 - Bilimsel temel net değildir.

Yazılım Mühendisliği Bilim Dalı

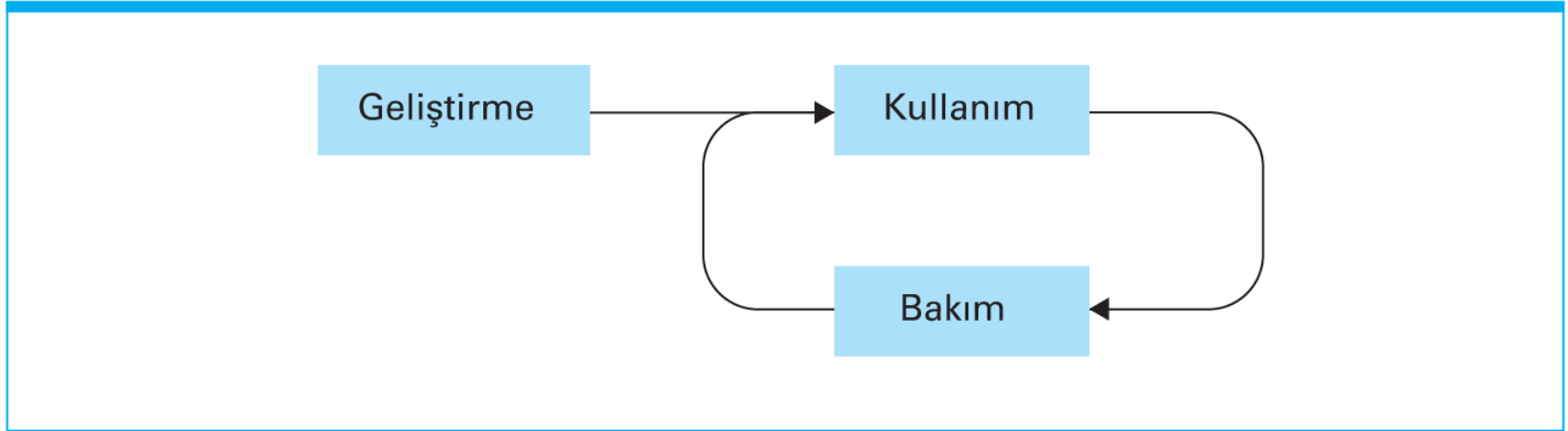
- Yazılım mühendisliğindeki araştırmalar iki seviyede ilerlemektedir:
 - Uygulayıcılar: Hızlı yazılım geliştirme teknikleri üzerine çalışmaktadırlar.
 - Kuramcılar: Gelecekte gerçekleştirilecek kararlı geliştirme tekniklerinin temel prensipleri ve teorileri üzerine araştırmalar yapmaktadırlar.
- Uygulayıcıların ve kuramcıların ilerlemesine çok ihtiyaç vardır.
 - Toplumlar bilgisayarlar ve onlarla ilişkili yazılımlara bağımlı hale gelmiştir.
- Meslek Kuruluşları: ISO, ACM, IEEE, vb. Mesleki etik kuralları ve Standartlar üzerinde çalışmaktadırlar.

Yazılım Mühendisliği Bilim Dalı

- Yazılım Mühendisliğinde önemli bir konu Computer Aided Software Engineering (CASE - Bilgisayar Destekli Yazılım Mühendisliği) araçlarının tasarımıdır:
- Proje Planlama Sistemleri
 - Proje Yönetim Sistemleri
 - Belgeleme Araçları
 - Prototipleme ve Simülasyon
 - Arayüz Tasarım Sistemleri
 - Programlama Sistemleri

Yazılım Yaşam Döngüsü

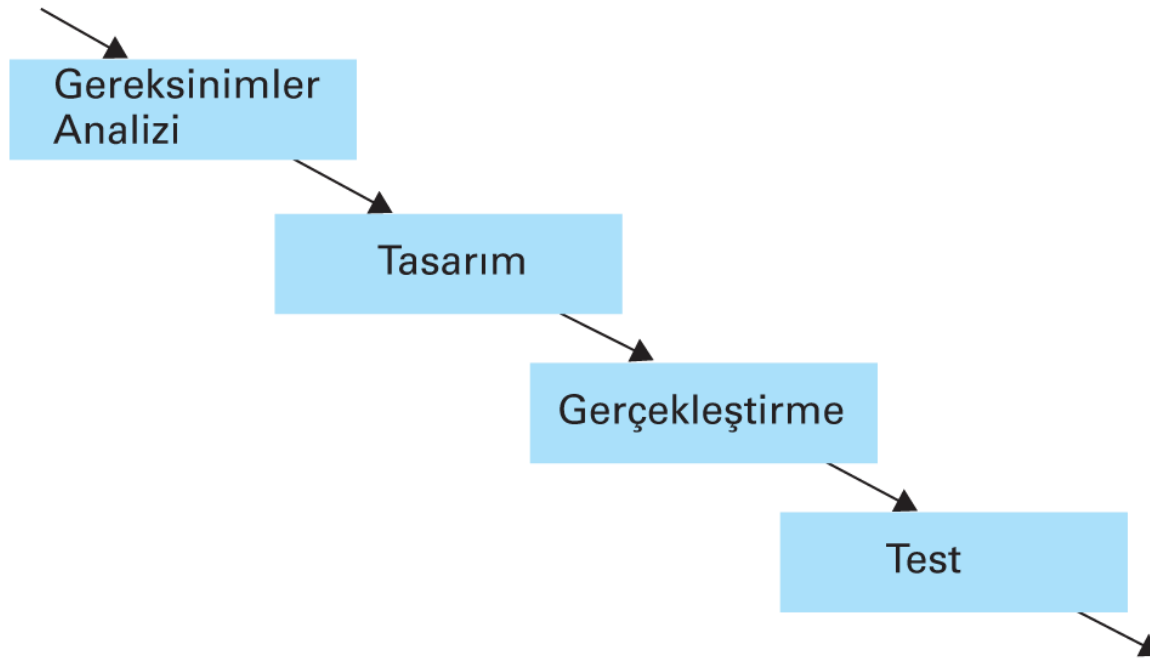
- Basit bir yazılım yaşam döngüsü üç aşamadan oluşur:



- Yazılımın geliştirilme sürecinde gösterilecek ufak bir gayret, değişiklik gerektiği zaman çok büyük bir fark yaratır.
- Üretilen ürünlerden farklı olarak bakım aşaması, yükseltme ve güncelleme aşamalarından oluşur.
- Devam eden bakım süreci, programı ve belgeleri anlayan birini gerektirir.

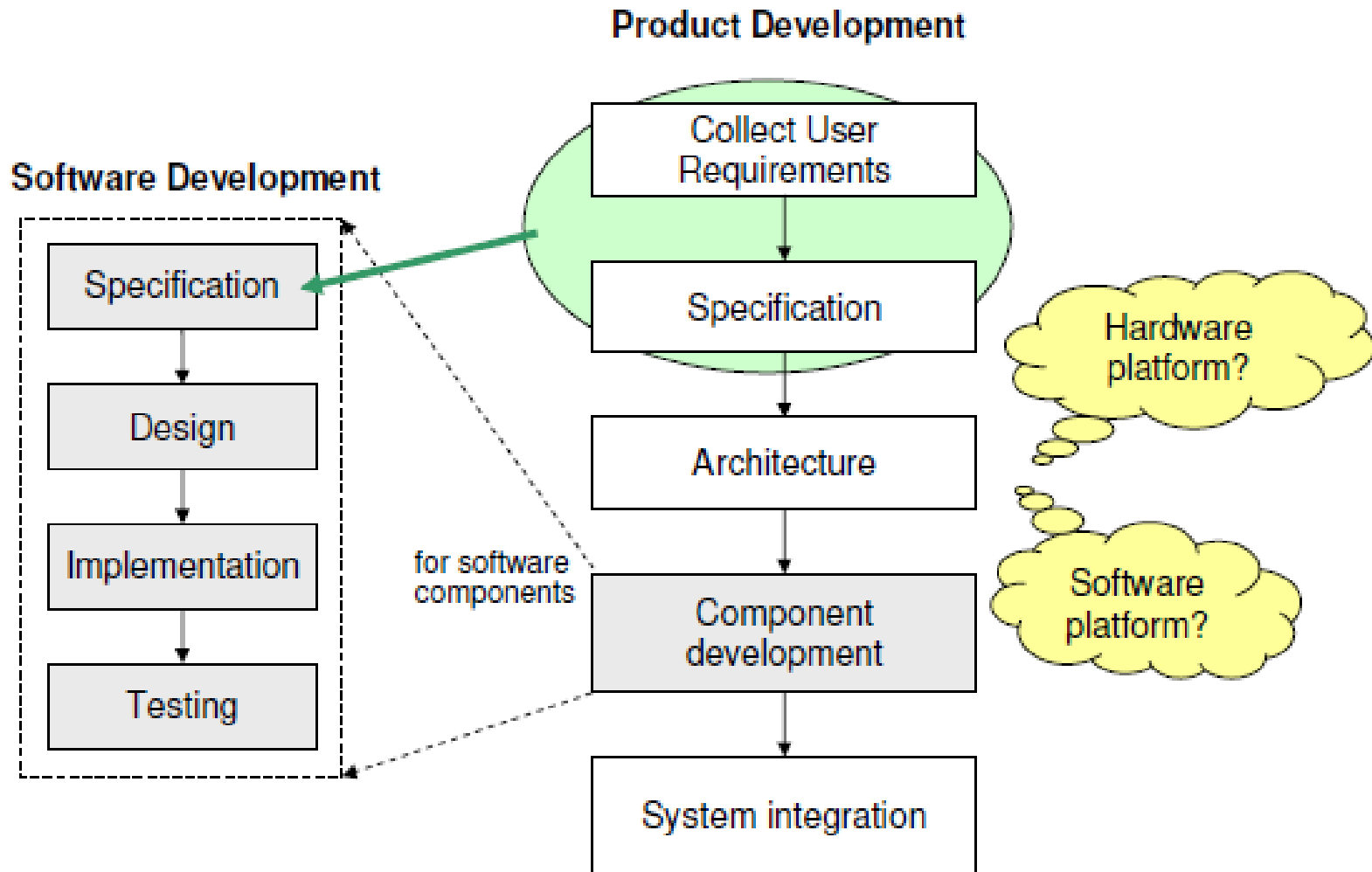
Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları



Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları



Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları

➤ Gereksinimler Analizi Safhası

- Uygulama kullanıcısı gereksinimlerine ve bazı teknik spesifikasyonlara dayanarak bir fizibilite çalışması yapılır.
- Analiz aşamasının çıktısı “yazılım gereksinim analizi dokümanı (şartname)”dır.

➤ Tasarım Safhası

- Tasarım aşaması sistemin gereksinim hedeflerine nasıl ulaşacağı konusundaki metodoloji ve araçlara odaklanır.
- Kullanıcı arayüzü tasarımı sanat, psikoloji, ergonomi ve programlama konularında karma bilgi gerektirir.

Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları

➤ Gerçekleştirme Safhası

- Tasarımdan sistem oluşturulan aşamadır.
 - Program yazılır.
 - Veri dosyaları oluşturulur.
 - Veri tabanları geliştirilir.
- Teoride:
 - Bir yazılım analisti (yazılım mimarı), gereksinimler ve tasarım aşamasında yazılım geliştirme sürecine dahil olur
 - Bir programcı tasarımı uygulayan programlar yazar
- Uygulamada: yazılım mimarları ve programcılar birbirinin yerine kullanılabilen terimlerdir.

Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları

➤ Test Safhası

- Doğrulama testi

- Sistemin gereksinimleri karşılamaının kontrolleri yapılır.

- Hata testi

- Sistem hatalarının (bug) bulunup düzeltilmesi adımlarını içerir.



BUG



DEBUG

Yazılım Yaşam Döngüsü

➤ Geleneksel Yazılım Geliştirme Safhaları

➤ Test Safhası

- Testin ayrıntı düzeyi:
- Modül testi – tek bir modülü test edin
 - Bir sistemdeki bir modülü test etmek için genellikle sistemdeki diğer modüllerin (saplamlar adı verilen) basitleştirilmiş versiyonları kullanılır.
- Birim testi – en küçük yazılım modülünü test edin
 - Zorunlu programlama dili için bu genellikle bir işlevdir
 - Test kodları, işlevi sınır ve/veya tekil noktalarda girişle çağırmak için yazılır
- Sistem testi – tüm sistemi test edin

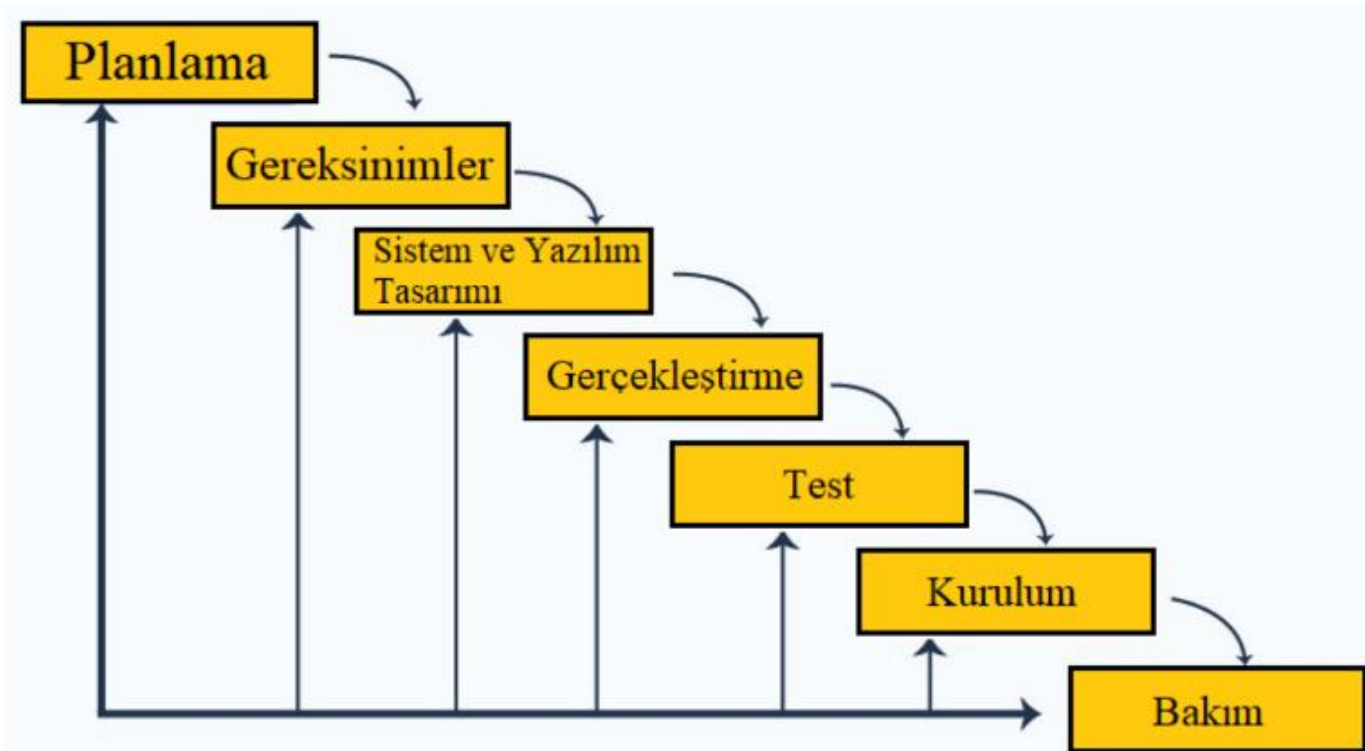
Yazılım Mühendisliği Metodolojileri

- Şelale/Çağlayan (Waterfall) Modeli
- Artımsal (Incremental) Model
- Evrimsel Prototipleme
- Tekrarlamalı (Iterative) Model
- Prototipleme
- Açık Kaynak Kod (Open-source Development)
- Çevik Metotlar (Agile Methods)
- Uç Programlama (eXtreme Programming-XP)

Yazılım Mühendisliği Metodolojileri

➤ Şelale/Çağlayan (Waterfall) Modeli

- Sıfırdan komple bir sistem tasarlamak için yazılım geliştirmenin dört aşamasının talimatlarını sıkı bir şekilde takip eder.

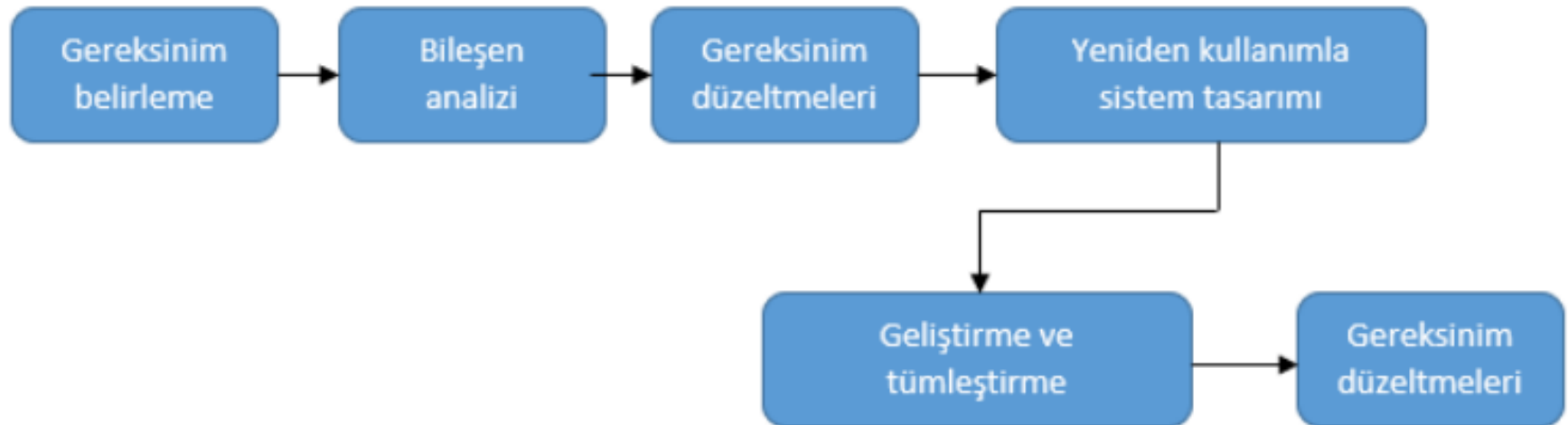
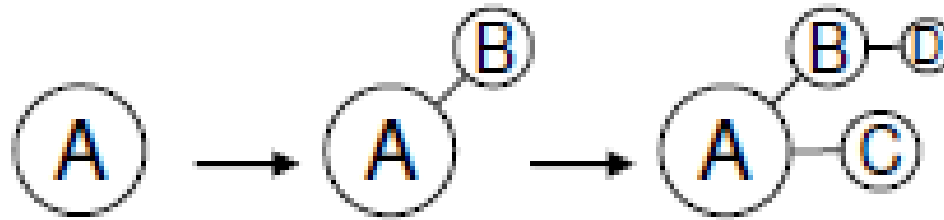


- Dinamik bir ortama tepki vermek için çok yavaştır.

Yazılım Mühendisliği Metodolojileri

➤ Artımsal (Incremental) Model

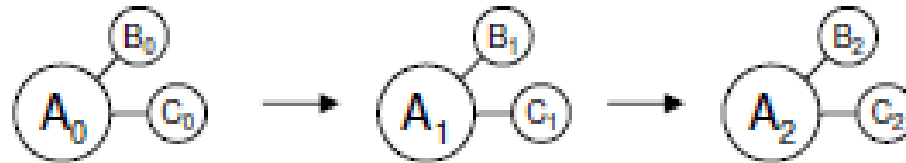
- Basitleştirilmiş bir sistemle başlanır ve adım adım giderek daha eksiksiz sistemler oluşturulur.



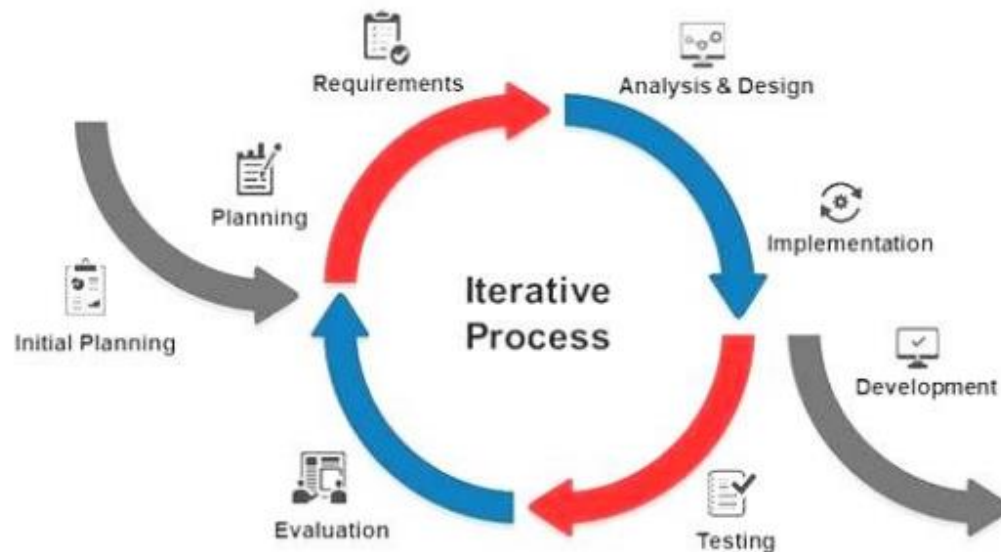
Yazılım Mühendisliği Metodolojileri

➤ Tekrarlamalı (Iterative) Model

- Basitleştirilmiş modüllere sahip tam bir sistemle başlanır ve adım adım daha fazla ve daha eksiksiz modüller oluşturulur.



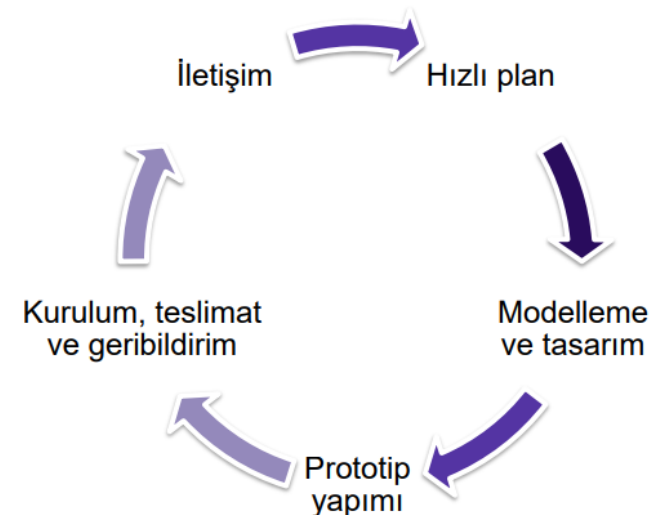
- Akılcı Birleştirilmiş Süreç: Rational Unified Process RUP



Yazılım Mühendisliği Metodolojileri

➤ Prototipleme

- Artımlı ve yinelemeli geliştirme modelleri için, nihai sistemin ilk sürümlerine genellikle prototip adı verilir.
 - Evrimsel prototip oluşturma – prototipler aşamalı olarak geliştirilecek ve sonunda nihai sistem haline gelecektir.
 - Tek kullanımlık prototip oluşturma – prototipler yalnızca iletişim ve tasarımın hızlı bir şekilde doğrulanması için kullanılır; daha sonra gerçek ürün için yeni bir uygulama kullanılacaktır.
 - Örneğin, kullanıcı arayüzü tasarımı için hızlı prototip oluşturma



Yazılım Mühendisliği Metodolojileri

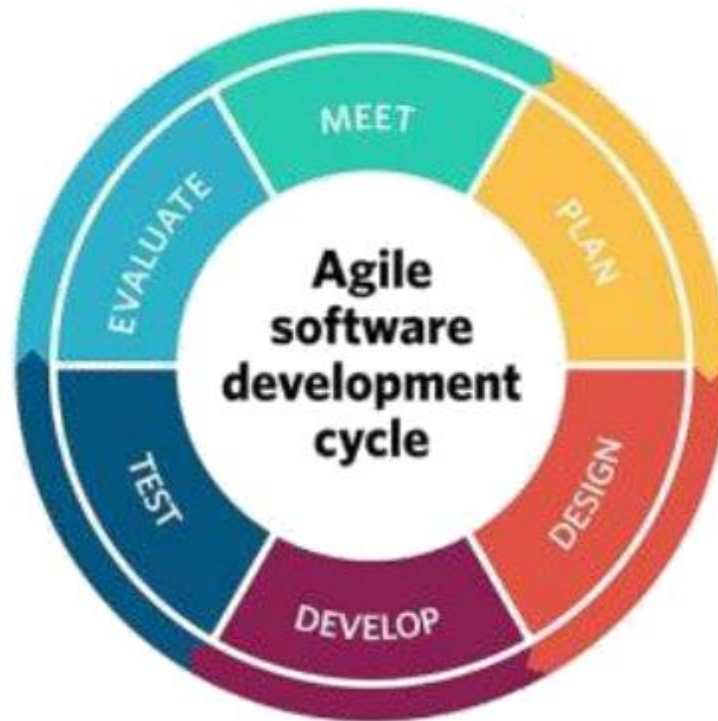
➤ Açık Kaynak Geliştirme (Open-source Development)



Yazılım Mühendisliği Metodolojileri

➤Çevik Metotlar (Agile Methods)

- Bir programlama projesi, eşit programcılardan oluşan bir ekibin esnek bir şekilde işbirliği yaparak tekrarlanan günlük analiz, tasarım, uygulama ve test döngüleri ile gerçekleştirilir.
- Uç Programlama (eXtreme Programming-XP)



Modülerlik

- Bir yazılım sistemi modüler olarak geliştirilmelidir.
- Yazılım yönetilebilir parçalara bölünmelidir.
- Bu parçaların her birine modül denir.
- Her bir modül yazılımın bir parçasıyla ilgilenir.
- Modüler tasarımın amacı birleşimleri en aza indirmek ve tutarlılığı en üst düzeye çıkarmaktır.
 - Birleşimler: Modüller arasındaki etkileşimler
 - Tutarlılık: Bir modül içindeki dahili bağlanma

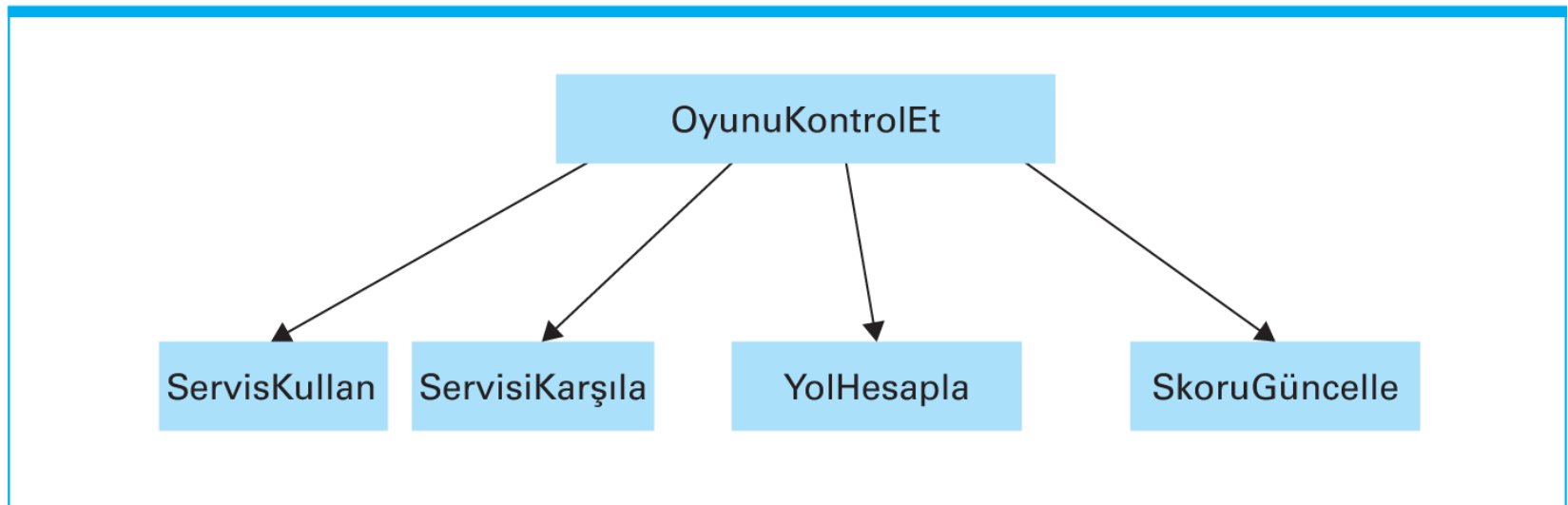
Modülerlik

- Yazılım modülerliği prosedür, nesne ve bileşen biçiminde gerçekleştirilebilir.
 - Prosedürler – emirsel paradigma
 - Anahtar bilgi: prosedür ilişkileri
 - Nesneler – nesne yönelimli paradigma
 - Anahtar bilgi: nesne örnekleri ve nesne işbirlikleri
 - Bileşenler - genellikle nesne yönelimli paradigma için yeniden kullanılabilir yazılım birimleri
 - Anahtar bilgi: bileşen arayüzleri

Modülerlik

➤ Modüler Gerçekleştirme

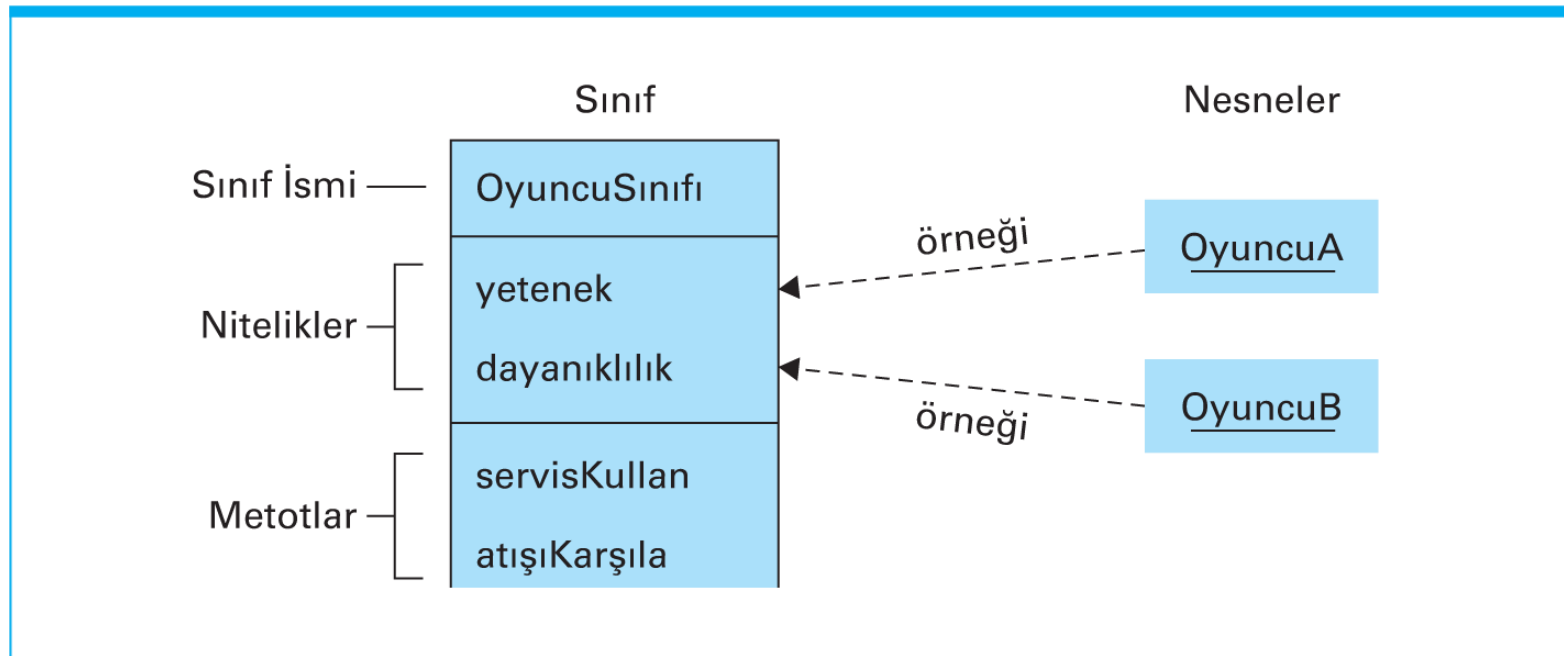
- Yazılım modülleri arasındaki ilişkileri ve etkileşimleri açıklamak için bazı diyagramları kullanabilirsek güzel olurdu.
- Yapı şeması - prosedürel tasarımın modülleri arasındaki ilişkileri görüntüler:
- Tenis oyununu simüle eden basit bir yapı şeması



Modülerlik

➤ Modüler Gerçekleştirme

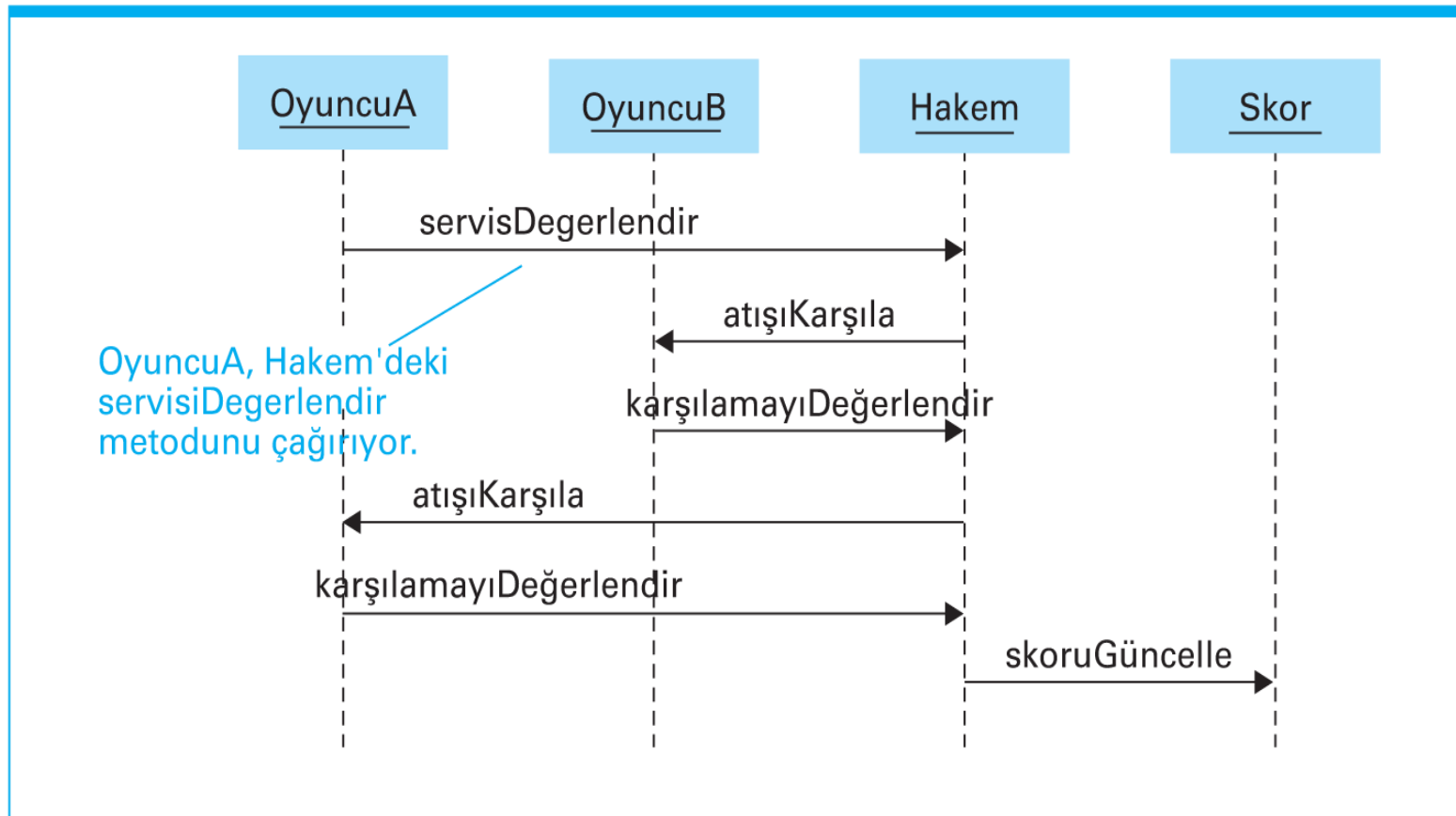
- Nesneye yönelik tasarımlarda modüller arasındaki ilişkileri ve etkileşimleri açıklamak yapı şemalarından daha fazlasını gerektirir.
- OyuncuSınıfı ve nesne ilişkileri



Modülerlik

➤ Modüler Gerçekleştirme

- OyuncuSınıfı ve nesne etkileşimleri



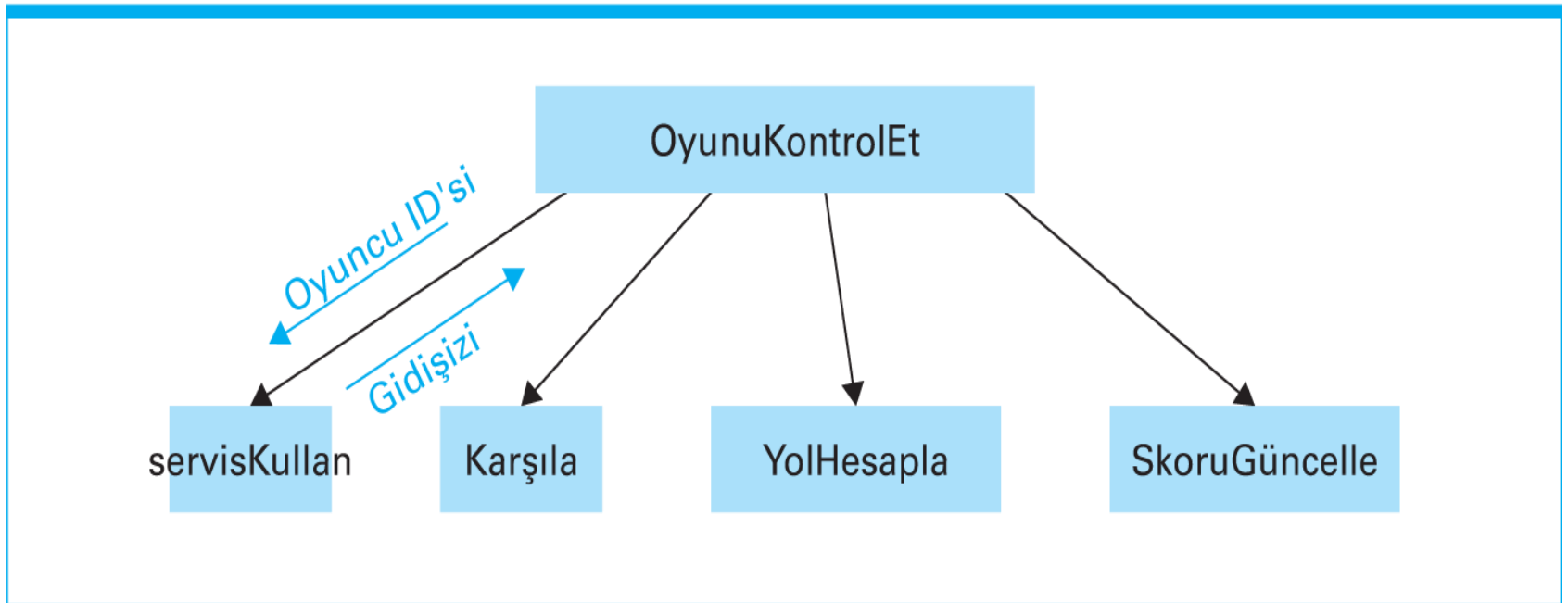
➤ Birleşim (coupling)

- Modüller arası birleşim yapılırken, birleşim olabildiğince azaltılmalıdır.
- Kontrol Birleşimi:
 - Bir modülün diğerine çalıştırma kontrolünü iletmesiyle gerçekleşir. (Fonksiyon çağırma benzeridir.)
- Veri Birleşimi:
 - Modüller arasındaki veri paylaşımını ifade eder.
- Örtülü Birleşim: Gizli birleşim hatalara neden olabilir
 - Küresel veriler: tüm modüllerin erişebildiği veriler
 - Yan etkiler: Arayanı tarafından hemen anlaşılamayan bir prosedür tarafından gerçekleştirilen eylem

Modülerlik

➤ Birleşim (coupling)

- Veri birleşimini de içeren yapı şeması



➤Tutarlılık (Cohesion)

- Modüller arası birleşimi en aza indirmek ne kadar önemliyse her modüldeki içsel bağlanmayı olabildiğince yükseltmek o kadar önemlidir.
- Tutarlılık terimi bu içsel bağlanmayı veya başka bir ifadeyle bir modülün iç parçalarının ilişkisi derecesini ifade eder.
- Mantıksal tutarlılık:
 - Bir modüldeki eylemlerin ve bileşenlerin mantıksal benzerliği
- Fonksiyonel tutarlılık:
 - Her bileşen tek bir aktivitenin gerçekleştirilmesine odaklanır
 - Mantıksal uyumdan daha güçlü

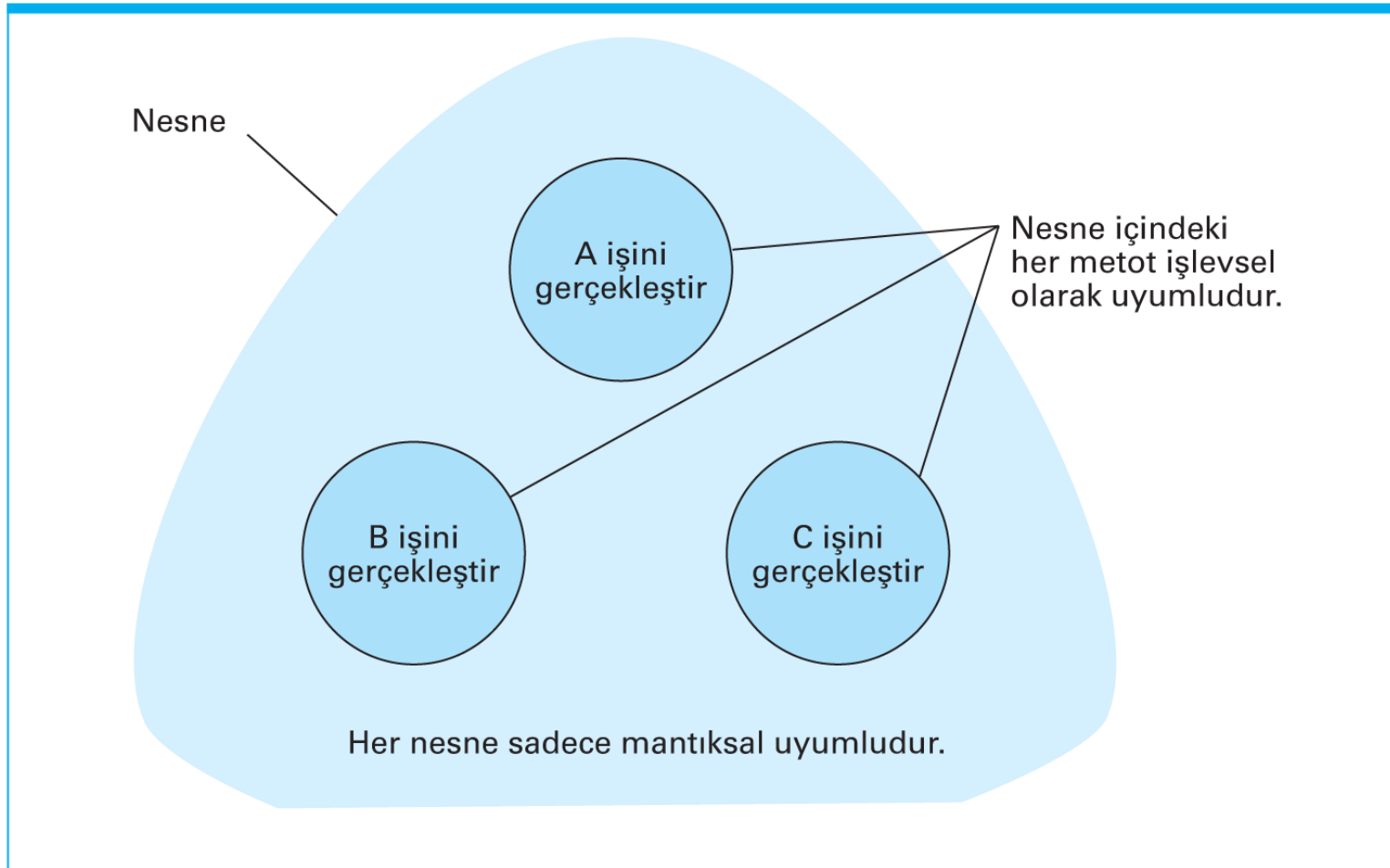
➤ Tutarlılık (Cohesion)

- Nesneye yönelik sistemlerde tutarlılık
 - Nesnenin tamamı (yani bir nesnedeki veri alanları ve yöntemlerin toplanması) mantıksal olarak uyumlu olmalıdır.
 - Her yöntem (yani bir yöntemde gerçekleştirdiğiniz görevler) işlevsel olarak uyumlu olmalıdır.

Modülerlik

➤ Tutarlılık

- Bir nesne içerisindeki mantıksal ve fonksiyonel tutarlılık



➤ Bilgi Saklama

- İyi modüler tasarımın yapı taşlarından biri bilgi saklama kavramında ortaya çıkmaktadır.
- **Bilgi saklama** yazılım sisteminin belli bir kısmına bilgi kısıtlanması olarak tanımlanmaktadır.

➤ Bileşenler

- Bileşenlerin geliştirilmesi ve kullanımındaki araştırmalar **bileşen mimarisi** (bileşen tabanlı yazılım mühendisliği) olarak bilinen alanın ortaya çıkmasına vesile olmuştur.
- Bu alanda programcının alışlagelmiş rolü bileşen birleştiricisi ile değişmiştir.
- **Bileşen birleştiricisi** yazılım sistemini birçok geliştirme ortamında önceden hazırlanmış bileşenlerden oluşturur.

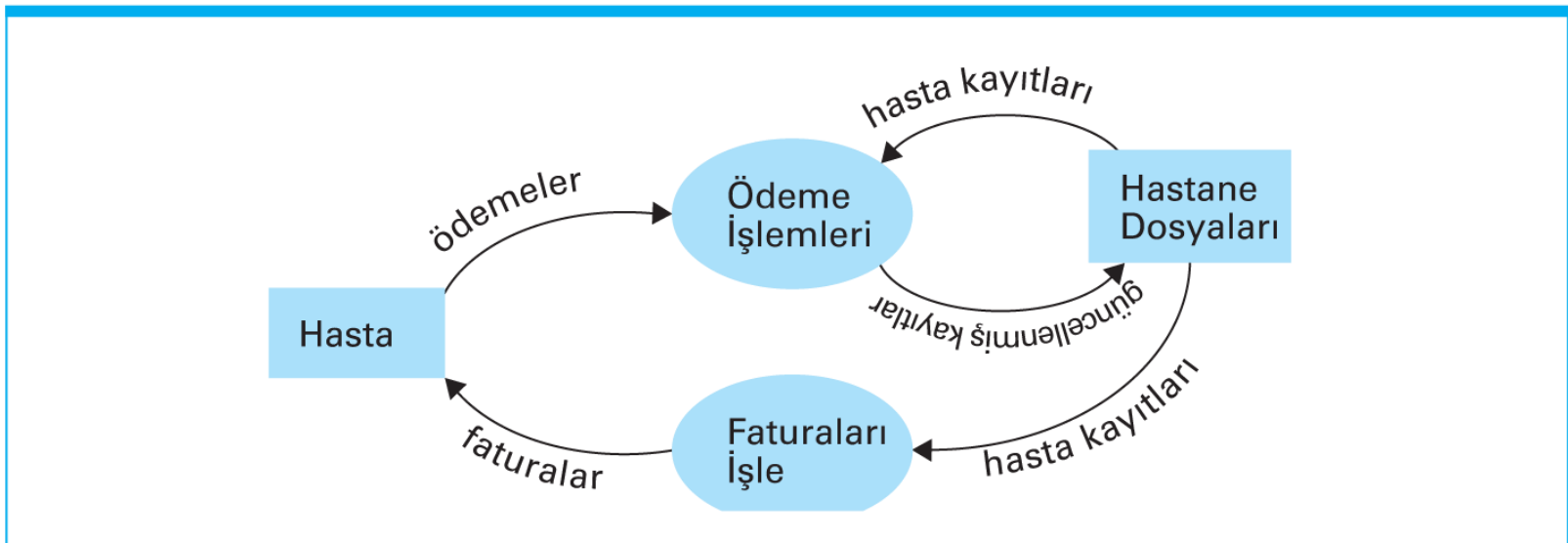
İş Araçları

- Veri Akış Diyagramı (Data Flow Diagram)
- Veri Sözlüğü (Data Dictionary)
- Birleştirilmiş Modelleme Dili (Unified Modelling Language – UML)
 - 1-to-1: Bire bir ilişki
 - 1-to-n: Birden çoğa ilişki
 - n-to-n: Çoktan çoğa ilişki
- Tasarım Kalıpları (Design Patterns)

İş Araçları

➤ Veri Akış Diyagramı

- Yazılımı prosedürler veya fonksiyonlar bakımından oluşturmak hedeflemesine rağmen bu fonksiyonları tanımlamanın bir yolu fonksiyonların kendilerinden ziyade idare edilecek veriyi ele almaktır.
- Buradaki teori verinin sistem boyunca nasıl hareket edeceğini incelemektir.



İş Araçları

➤ Veri Sözlüğü

- Bir yazılım sistemi boyunca görünen veri öğeleri hakkındaki bilgilerin merkezi deposu

Data Dictionary

Data Dictionary outlining a Database on Driver Details in NSW

Field Name	Data Type	Data Format	Field Size	Description	Example
License ID	Integer	NNNNNN	6	Unique number ID for all drivers	12345
Surname	Text		20	Surname for Driver	Jones
First Name	Text		20	First Name for Driver	Arnold
Address	Text		50	First Name for Driver	11 Rocky st Como 2233
Phone No.	Text		10	License holders contact number	0400111222
D.O.B	Date / Time	DD/MM/YYYY	10	Drivers Date of Birth	08/05/1956

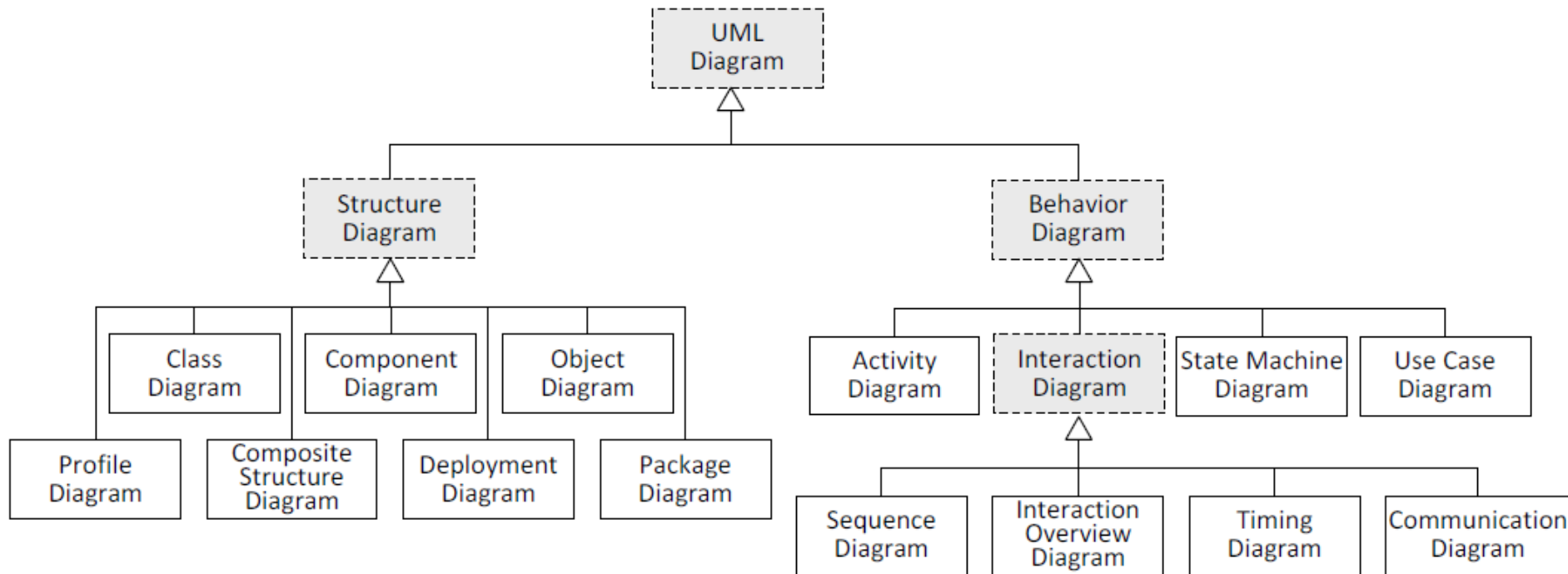
➤ Birleştirilmiş Modelleme Dili

- Veri akışı diyagramları ve veri sözlükleri nesneye yönelik paradigma ortaya çıkmasına yakın ve zorunlu paradigma popülaritesini kaybetmesine rağmen kullanışlı roller bulmaya devam eden yazılım mühendisliğinin teçhizat deposundaki araçlardı.
- Daha sonraki yıllarda nesneye yönelik paradigmayla beraber geliştirilen daha modern araçların derlemesi olan **Birleştirilmiş Modelleme Dili (Unified Modeling Language-UML)** gelişmiştir.
- UML, büyük karmaşık sistemleri belirlemek, görselleştirmek, oluşturmak ve belgelemek için kullanılan bir dildir.

İş Araçları

➤ Birleştirilmiş Modelleme Dili

- 14 tür UML 2.0 diyagramı vardır:
 - Gri kutular diyagram sınıflarıdır; beyaz kutular diyagram örnekleridir

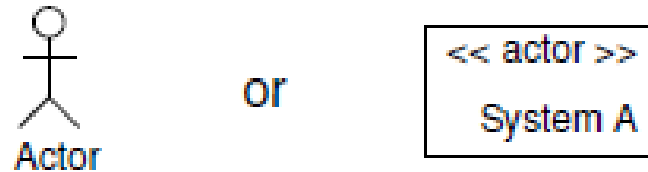


İş Araçları

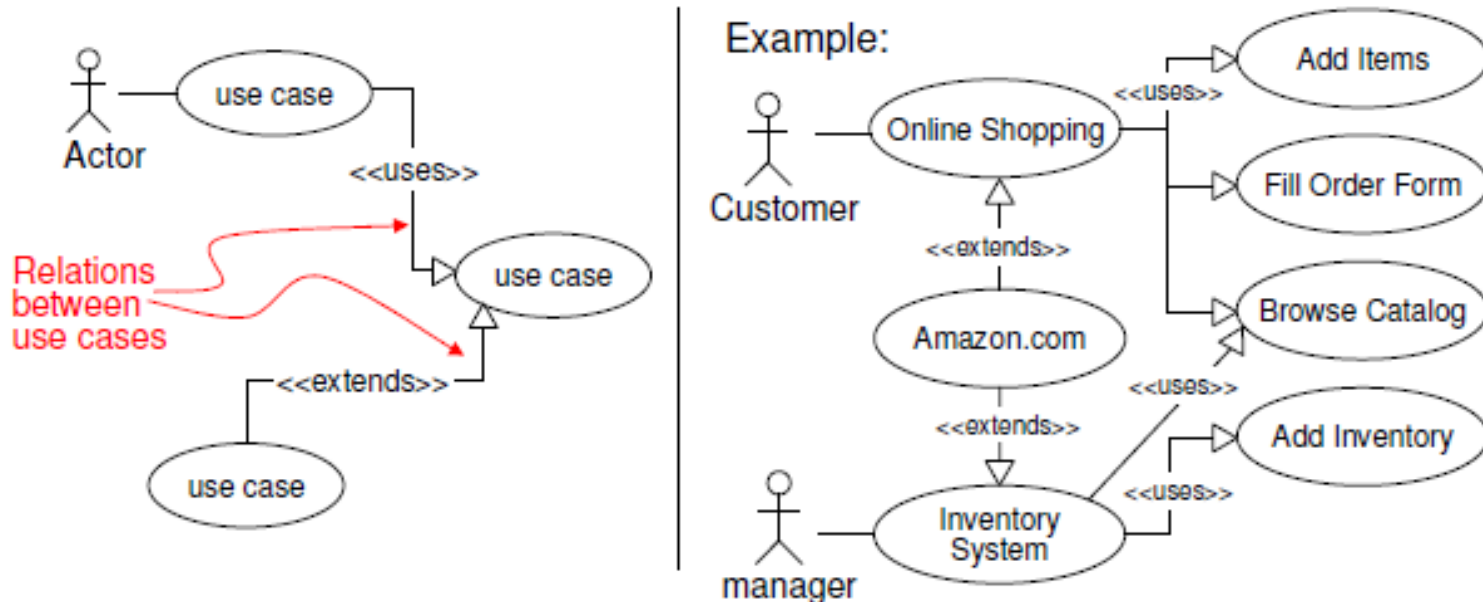
➤ Birleştirilmiş Modelleme Dili:

■ Kullanım Durumu Diyagramı (Use Case Diagram)

- Aktörler (Actors): Sistemin kullanıcıları

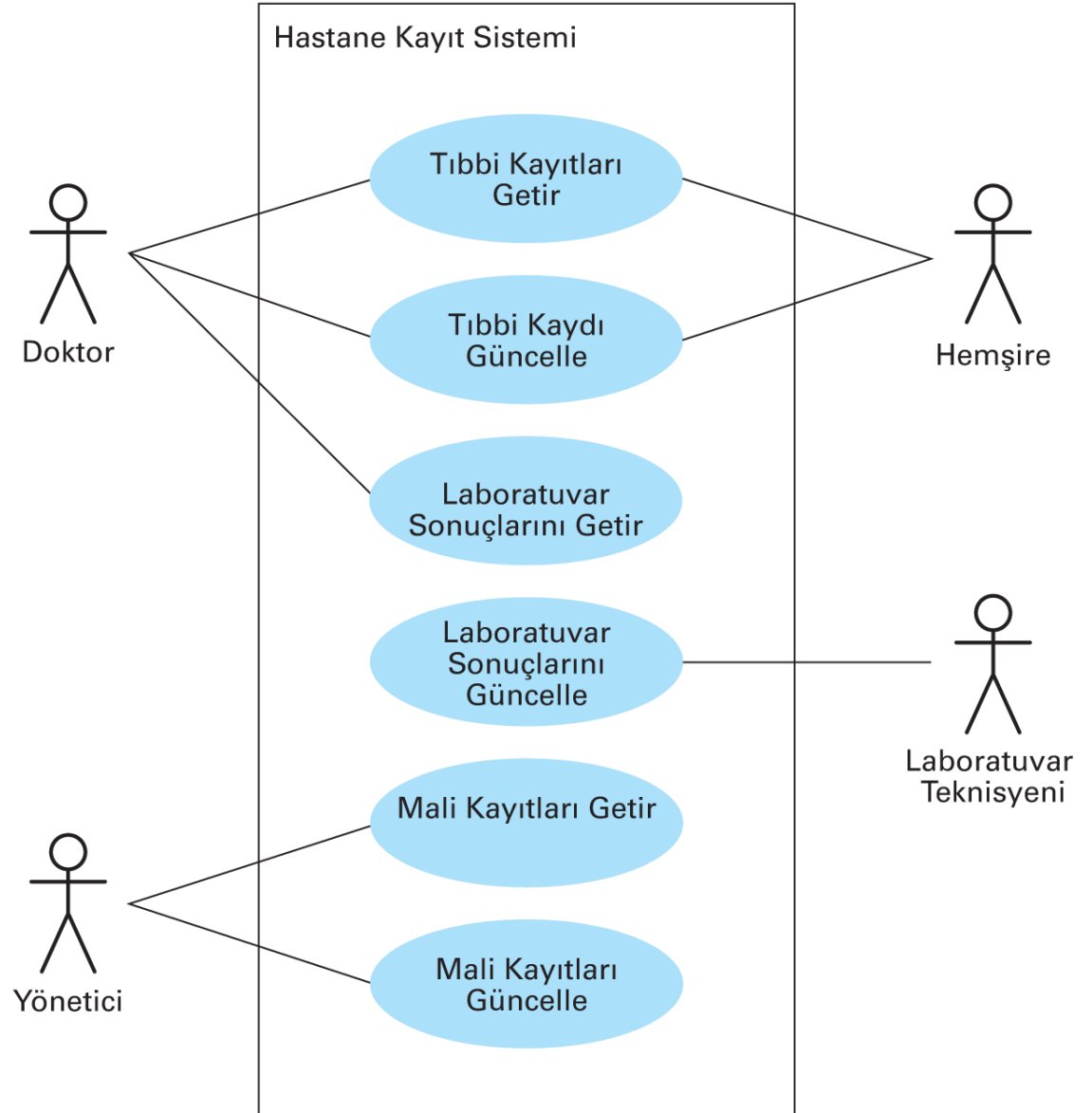


- Kullanım Durumları (Use cases): Sistemin fonksiyonları



İş Araçları

➤ Başka Bir Kullanım Durumu Diyagramı Örneği

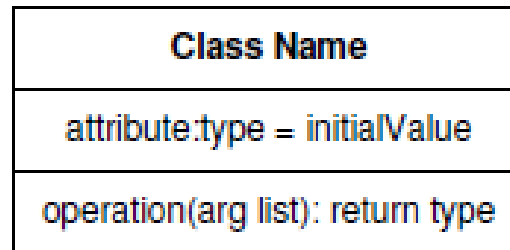


İş Araçları

➤ Birleştirilmiş Modelleme Dili

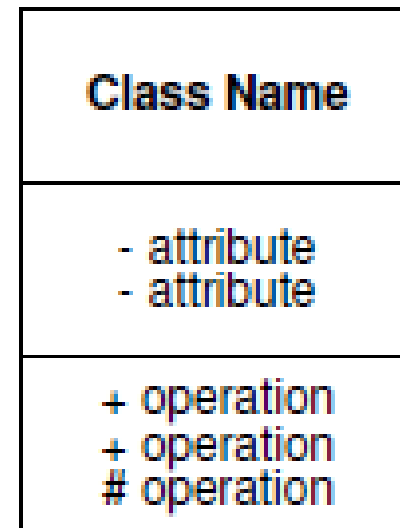
■ Sınıf Diyagramı (Class Diagram)

- Farklı sınıflar arasındaki ilişkileri (bağımlılıklar veya işbirlikleri) tanımlar
- Sınıf tanımı:



- Görünürlük (erişilebilirlik):

- + public erişim
- - private erişim
- # protected erişim

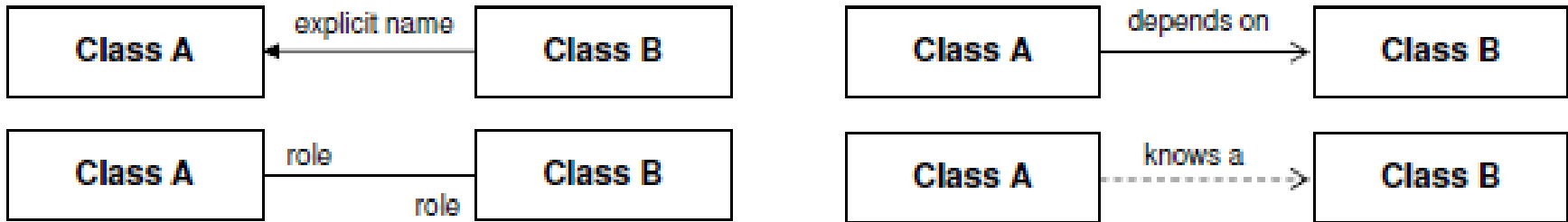


İş Araçları

➤ Birleştirilmiş Modelleme Dili

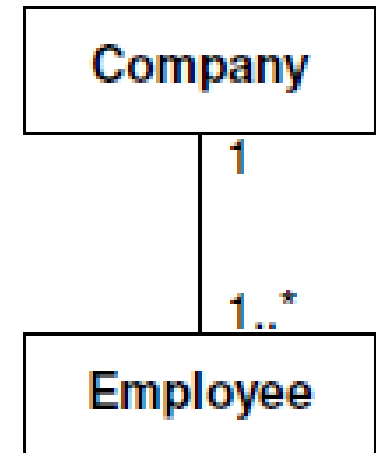
■ Sınıf Diyagramı (Class Diagram)

○ İlişkilerin tanımı:



○ Çeşitlilik:

- 1 birden fazla anlamına gelmez
- 0..1 sıfır veya bir anlamına gelir
- * çok anlamına gelir
- 0..* sıfır veya çok anlamına gelir
- 1..* bir veya daha fazla anlamına gelir



İş Araçları

➤ Birleştirilmiş Modelleme Dili

- Sınıf Diyagramı (Class Diagram)
 - Başka bir ilişki tanımı örneği



İş Araçları

➤Tasarım Kalıpları

- Yazılım mühendisleri için gücü giderek artan bir araç da tasarım kalıplarıdır.
- Bir **tasarım kalıbı** yazılım tasarımında tekrarlayan problemleri çözmek için önceden geliştirilmiş bir modeldir.
- Yineleme problemlerini çözmek için iyi tasarlanmış kalıpları yazılım mühendisliğinde sürekli devam eder.
- Mimaride Christopher Alexander'ın çalışmalarından esinlenilmiştir.

➤Tasarım Kalıpları

- Creational Patterns: Singleton Pattern, Factory Pattern, Abstract Factory Pattern, Builder Pattern, Prototype Pattern
- Structural Patterns: Adapter Pattern, Bridge Pattern, Filter Pattern, Composite Pattern, Decorator Pattern, Facade Pattern, Flyweight Pattern, Proxy Pattern
- Behavioral Patterns: Chain of Responsibility Pattern, Command Pattern, Interpreter Pattern, Iterator Pattern, Mediator Pattern, Memento Pattern, Observer Pattern, Null Object Pattern, Strategy Pattern, State Pattern, Visitor Pattern

➤Tasarım Kalıpları

- Tasarım kalıbı örnekleri:

- Bağdaştırıcı modeli: Bir modülün arayüzünü mevcut ihtiyaçlara uyarlamak için kullanılır.
- Dekoratör modeli: Aynı aktivitelerin birçok farklı kombinasyonu gerektiğinde ilgili karmaşıklığı kontrol etmek için kullanılır.

➤Tasarım Kalıpları

- Tasarım kalıplarının avantajları
 - Kalıplar birbirleri ile ilişkilendirilebilir.
 - Ortak bir kelime hazinesi sağlar.
 - Tasarım Kalıpları, nesneye yönelik programlamada etkili bir yöntemdir.
 - Kalıpların standartlaştırılması.
 - Zamanla, kalıpların katalogları yapılabilir.

Kalite Güvencesi

- İlk yıllarda, gerçekleştirme esnasında meydana gelen programlama hatalarının düzeltme odaklıydı.
- Yazılım hatalarının çoğalması, maliyetlerin ve teslim tarihlerinin aşılması yazılım kalite kontrolü metotlarının geliştirilmesi ihtiyacını doğurmuştur.
- Günümüzde, hata ayıklama sürecinin çok daha ötesine taşınmıştır.
 - Geliştirme sürecindeki her adım ileride başvurulmak üzere doğru bir şekilde belgelenmeli.
 - Çeşitli katılımcıların iştirak ettiği denetimler kullanılmalıdır. (gereksinim denetimi, tasarım denetimi vb.)

Kalite Güvencesi

➤ Kalite Güvencesinin Kapsamı

- Yazılım geliştirme firmaları yazılım kalite güvencesi (software quality assurance – SQA) grupları tertip etmektedirler.
- Bu gruplar firma tarafından kabul edilen kalite kontrol sistemlerini yönetmek ve güçlendirmekle görevlidirler.

Kalite Güvencesi

➤Yazılım Testi

- Yazılım kalite güvencesi bütün geliştirme sürecine yayılan bir konu olarak biliniyorken programların test edilmesi ve doğrulanması hala bir araştırma konusu olarak devam ediyor.
- Yazılım mühendisleri yazılımda sınırlı sayıda testle birlikte hataları ortaya çıkarma olasılığını geliştiren test metodolojileri geliştirmişlerdir.
- Bunlardan biri yazılımdaki hataların kümelenmeye meylettiğinin gözlemlenmesine dayanmaktadır.

➤ Yazılım Testi Stratejileri

- Glass-box testi
 - Pareto ilkesi – genellikle büyük bir sistemdeki yalnızca az sayıda yazılım modülü sorunludur.
 - Temel yol testi – test verileri, bir yazılım sistemindeki tüm olası yürütme dallarının test edilmesine olanak sağlamalıdır.
- Black-box testi
 - Boundary value analysis
 - Beta testing
 - Alfa testing

Dokümantasyon

- Bir yazılım sistemi insanlar tarafından kullanılamadığı ve bakımının yapılamadığı sürece bir işe yaramaz. Bu yüzden dokümantasyon son yazılım paketinin önemli bir parçasıdır.
- Kullanıcı dokümantasyonu
 - Yazılı kitaplar
 - Online yardım modülleri
- Sistem dokümantasyonu
 - Kaynak kod
 - Tasarım dokümanları
- Teknik dokümantasyon
 - Kurulum, düzenleme, güncelleme vb. için

İnsan-Makine Ara Yüzü

- İnsanlara bir yazılım sistemini soyut bir araç olarak kullanma imkanı verilmelidir. Bu araç uygulaması kolay ve insan kullanıcılarla sistem arasındaki iletişim hatalarını en aza indirmek için tasarlanmış olmalıdır.
- İyi ara yüz tasarımının önemi şu şekilde vurgulanmaktadır: bir sistemin ara yüzü kullanıcı üzerinde sistemin diğer herhangi bir özelliğinden daha güçlü bir izlenim vermelidir.
- Bir yazılım sisteminin, onu kullanan insanın rahatlığı için tasarlanmış bir araç olduğu fikri
- Ergonomi – insanların fiziksel yetenekleri
- Cognetics (Kognetik) – insanların zihinsel yetenekleri
 - Alışkanlıklar
 - Dikkat

Yazılım Sahipliği ve Yükümlülük

- Yazılım geliştiricilerin ürettikleri yazılım üzerinde bir sahiplik derecesine ihtiyaçları vardır.
- Bu gibi sahipliği sağlamak için yasal girişimler fikri mülkiyet kanunu kategorisi kapsamındadır.
- Bu kategori patent ve telif hakkı kanunu temeline dayanır.
- Aslında patent veya telif hakkının amacı ürün geliştiricilerine ürünlerini sahiplik haklarını koruyarak piyasaya sürme fırsatı vermektir.

Yazılım Sahipliği ve Yükümlülük

➤ Copyright – Telif hakkı

- Fikri mülkiyetin sahipliğini koruyarak bir ürünün satışına izin verir
- Tüm çalışmalarda:
 - Özellikler
 - Kaynak kodu
 - Son ürünbu kapsamdadır.

Yazılım Sahipliği ve Yükümlülük

➤ Fikri Mülkiyet

■ Yazılım lisansı

- Kullanıcıya mülkiyeti aktarmadan belirli izinler veren yasal bir sözleşme sağlar.

■ Patentler

- Yeni ve kullanılabilir olduğunu ortaya koymalı ayrıca benzer geçmişe sahip başkaları için açık olmadığını göstermelidir.
- İşlem pahalı ve zaman alıcıdır.



How the customer explained it



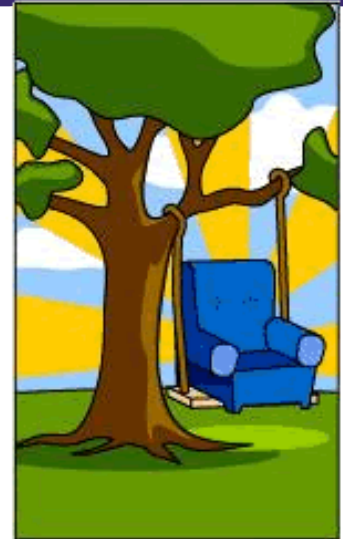
How the Project Leader understood it



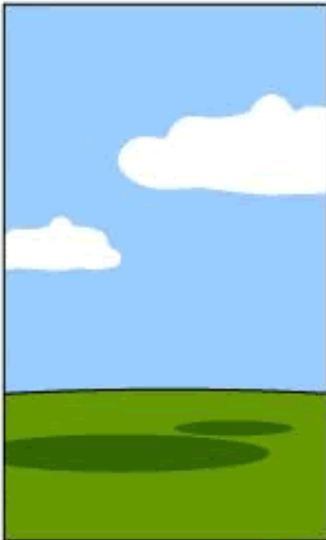
How the Analyst designed it



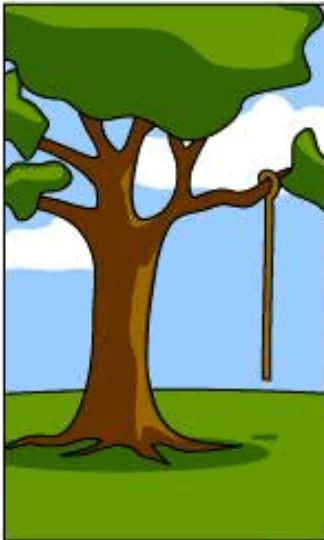
How the Programmer wrote it



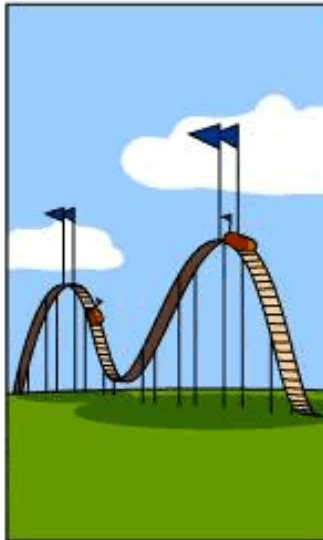
How the Business Consultant described it



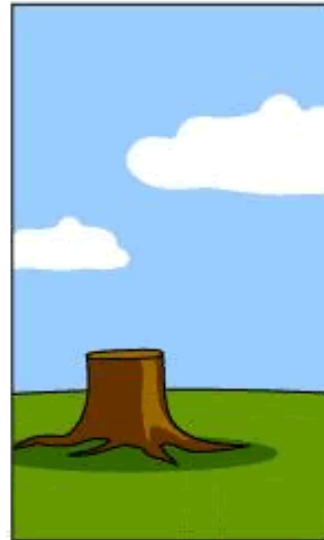
How the project was documented



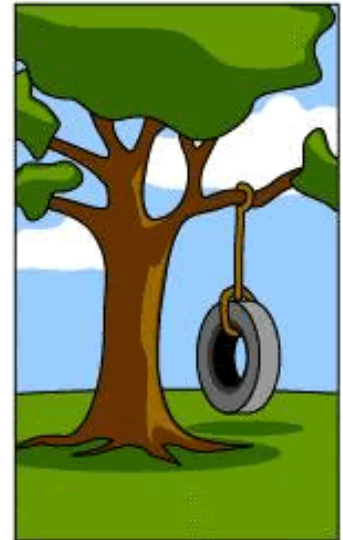
What operations installed



How the customer was billed



How it was supported



What the customer really needed