

# Nesneye Y6nelik Yazılım M6uhendislięi (376)

---

*Dr. 6ęr. 6yesi Ahmet Arif AYDIN*

# Functional Decomposition Problems

---

## ❖ weak cohesion

- ❖ bir çok işlemin ve amacın gerçekleştirilmesi
- ❖ tek bir işlem üzerinde yoğunlaşılması
- ❖ bir metodun bir den fazla işlemi gerçekleştirmesi (ekle, sil, güncelle)

## ❖ tight coupling

- ❖ çok fazla bağımlılığın olması

## ❖ we want to build *highly cohesive* and *loosely coupled* systems.

# *Functional Decomposition & Object-oriented Programming*

---

## ❖ **Functional Decomposition**

### ❖ Ana program

- ❖ bütün işlem ve aşamalardan

- ❖ Değişiklik isteklerinden

- ❖ Program içerisindeki her bir varlığın işleyişinden sorumlu

## ❖ **Object-Oriented Design**

- ❖ Ana programda genel aşamalarını tanımlar

- ❖ Programdaki her bir nesnenin görevi, yapacağı iş tanımlı ve sorumluluğunda ve özelleştirilmiş

# Object-Oriented Design and Functional Decomposition

1. Kütüphane mevcut olan kitapları listele
2. Emanette olan ve süresi geçen kitapları alan kişilerin bilgilerini listele
3. Çıkış

↑

Kütüphane Otomasyonu

↓

Arama (Search)

Ekleme (Insert)

Güncelleme (Update)

Raporlar (Reports)

Ödünç Kitap

Admin (Yönetim)

Veritabanı

# Object-Oriented Paradigm: Abstraction (soyutlama)

- ❖ Bir varlığın belirli amaçları gerçekleştirmek , bir görevi veya problemi çözmek için sağlamış olduğu tanımlamadır.
  - ➔ Assembly dili makine dili için , yüksek seviyeli programlama dilleri assembly dili için bir *abstraction* dır.
  - ➔ Bir sınıf içerisinde tanımlanan **değişkenler, methodlar** , sınıfa erişmek için oluşturulan nesneler (**instance object** )

```
class Hesaplama {  
    public int z;  
    public void toplama(int x, int y) {  
        z = x + y;  
        System.out.println("Toplam:"+z);  
    }  
    public void cikarma(int x, int y) {  
        z = x - y;  
        System.out.println("Fark:"+z);  
    }  
}
```

```
public static void main(String args[]) {  
    int a = 20, b = 10;  
    Hesaplama n1 = new Hesaplama();  
    demo.toplama(a, b);  
    demo.cikarma(a, b);  
}
```

# Object-Oriented Paradigm: Information Hiding

---

- ❖ Bir metodun veya nesnenin detaylarının gizlenmesi işlemidir
  - ❖ *The process of hiding the details of an object or function*
  - ❖ *Mechanism for restricting access to some of the object's components.*

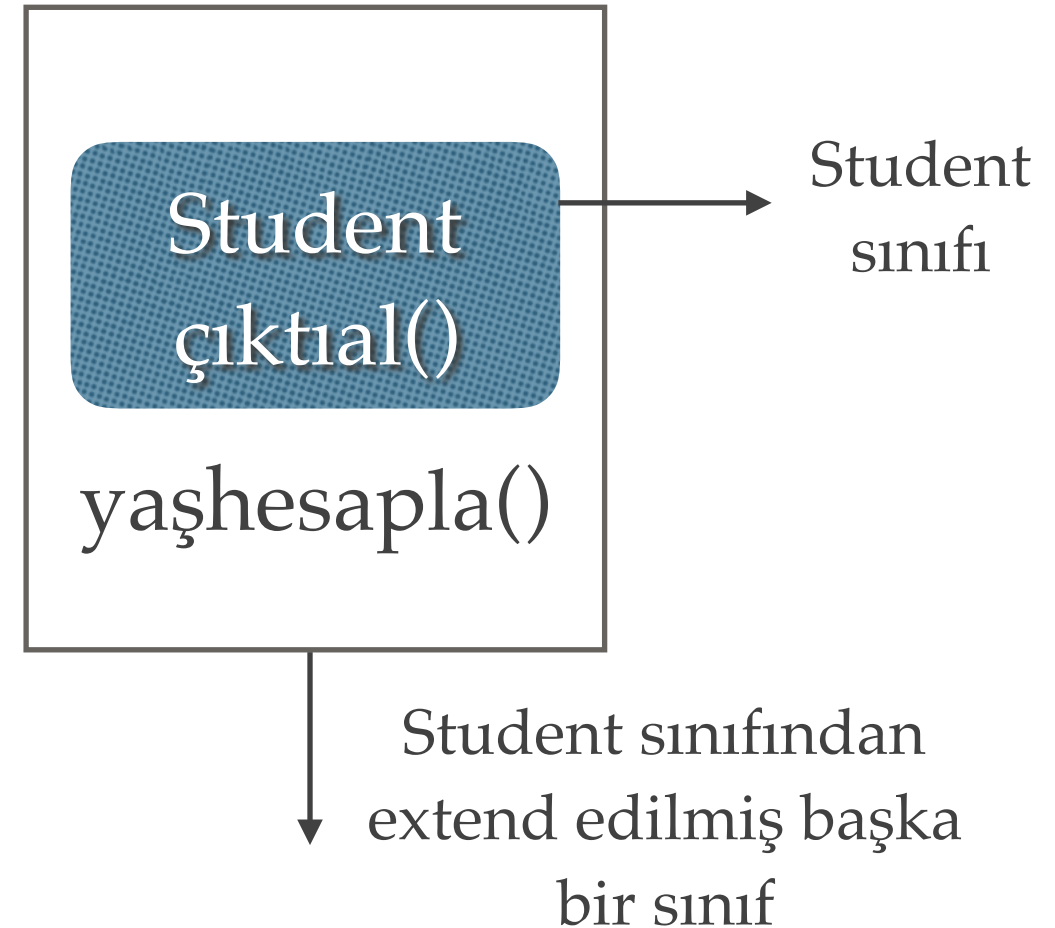
```
abstract public class çalışan {  
    abstract public void maaşhesapla();  
}
```

```
public class yönetici extends çalışan {  
    private int maaş;  
    private int katsayıdeger;  
  
    private void setkatsayı(){  
        this.katsayıdeger=25;  
    }  
    public int katsayıgönder(){  
        setkatsayı();  
        return katsayıdeger;  
    }  
}
```

# Object-Oriented Paradigm: Inheritance (kalıtım)

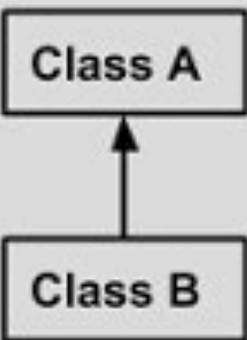
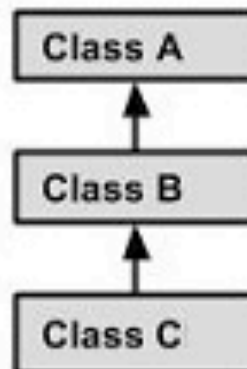
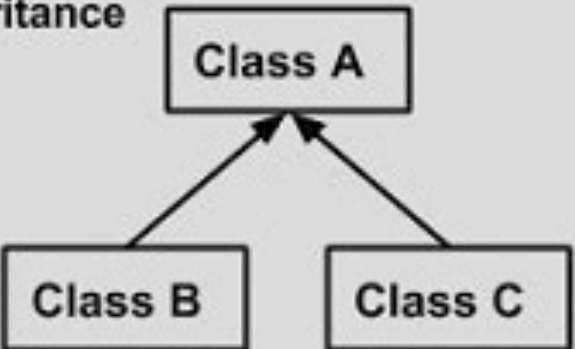
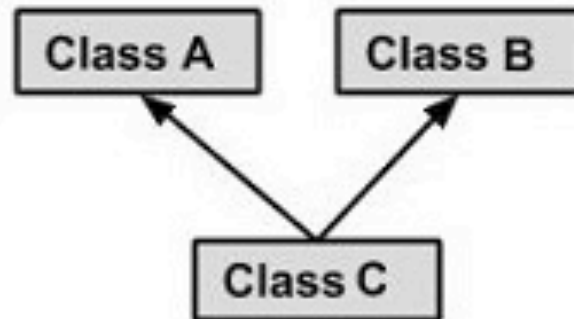
```
class Hesaplama {
    int z;
    public void toplama(int x, int y) {
        z = x + y;
        System.out.println("Toplam:"+z);
    }
    public void cikarma(int x, int y) {
        z = x - y;
        System.out.println("Fark:"+z);
    }
}

public class Dörtişlem extends Hesaplama {
    public void çarpma(int x, int y) {
        z = x * y;
        System.out.println("Çarpım:"+z);
    }
    public void bölme(int x, int y) {
        z = x / y;
        System.out.println("Bölme:"+z);
    }
}
```



```
public static void main(String args[])
{
    int a = 20, b = 10;
    Dörtişlem demo = new Dörtişlem();
    demo.toplama(a, b);
    demo.çıkarma(a, b);
    demo.çarpma(a, b);
}
```

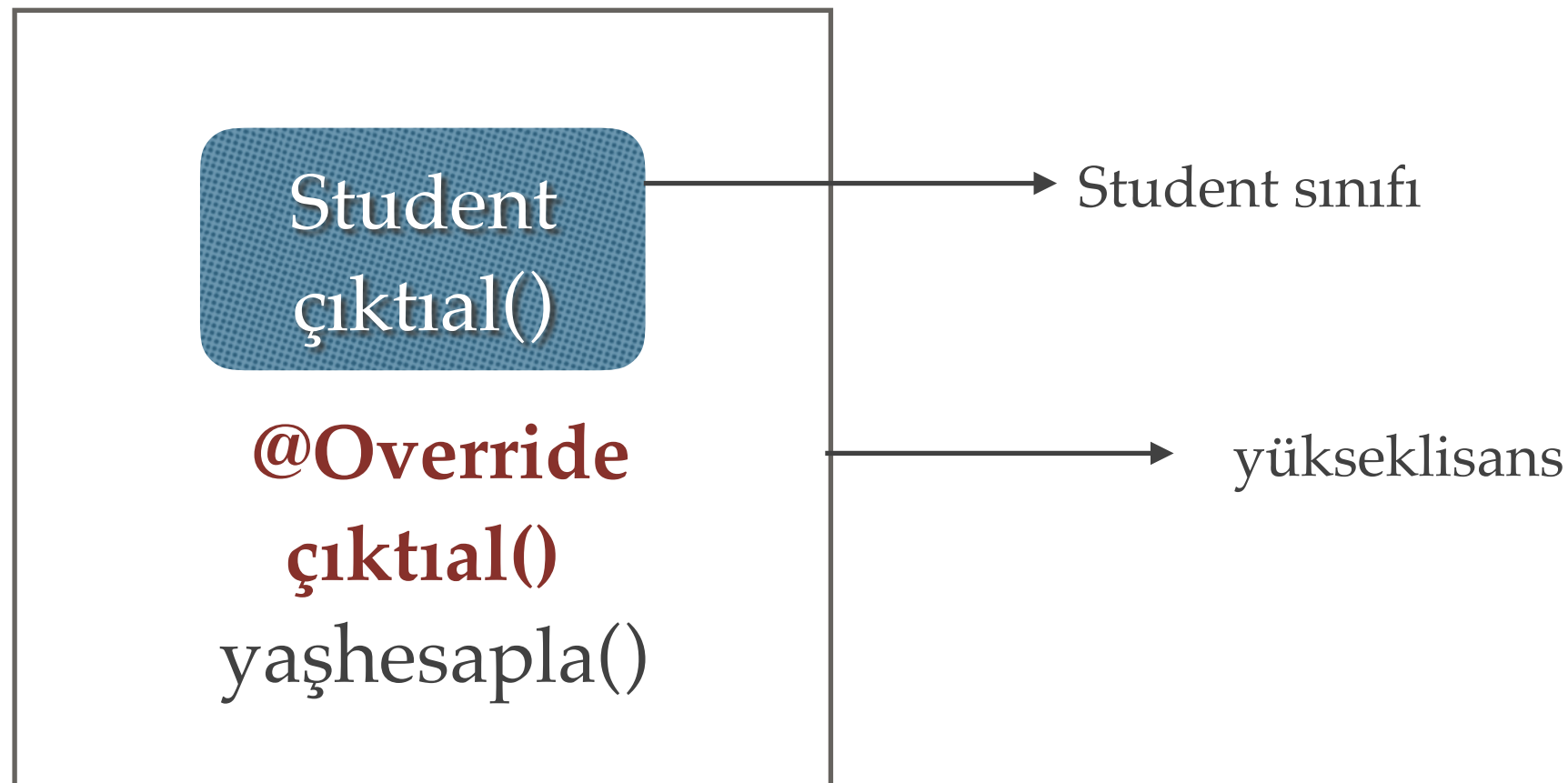
# Object-Oriented Paradigm: Inheritance (kalıtım)

Single Inheritance	 <pre>graph BT; B[Class B] --&gt; A[Class A]</pre>	<pre>public class A {     ..... } public class B extends A {     ..... }</pre>
Multi Level Inheritance	 <pre>graph BT; C[Class C] --&gt; B[Class B]; B --&gt; A[Class A]</pre>	<pre>public class A { .....} public class B extends A {.....} public class C extends B {.....}</pre>
Hierarchical Inheritance	 <pre>graph BT; B[Class B] --&gt; A[Class A]; C[Class C] --&gt; A</pre>	<pre>public class A { .....} public class B extends A {.....} public class C extends A {.....}</pre>
Multiple Inheritance	 <pre>graph BT; C[Class C] --&gt; A[Class A]; C --&gt; B[Class B]</pre>	<pre>public class A { .....} public class B {.....} public class C extends A,B {     ..... } // Java does not support mutiple Inheritance</pre>



# Object-Oriented Paradigm: Polymorphism (çok biçimlilik)

- ❖ Bir sınıfın başka bir sınıftan metodlarını kalıtsal olarak devralıp kendine özel bir biçimde tekrar yazma işlemidir.
- ❖ *ability of an object to take on many forms*
- ❖ *to allow an entity such as a **variable**, a **function**, or an **object** to have more than one form (<http://searchmicroservices.techtarget.com/definition/object>)*



# Object-Oriented Paradigm: Encapsulation

---

- ❖ **Kapsülleme tasarım detaylarını gizlemek için kullanılan teknik veya programlama dili seviyesindeki mekanizmalardır**
  - ❖ *a set of language-level mechanisms or design techniques that hide implementation details of a class, module, or subsystem from other classes, modules, and subsystems*
  - ❖ a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- ❖ Kapsüllemeyi gerçekleştirmek için *information hiding* de kullanılır

# Object-Oriented Paradigm: Polymorphism (çok biçimlilik)

```
1 /**@author ahmetarifaydin*/
   package pkginterface;

   public interface çalışan {

       public void rapor();
       public void maaş();
       public void katsayı();
   }

package pkginterface;
/**@author ahmetarifaydin*/
public class işçi implements çalışan {

    @Override
    public void rapor() {
        System.out.println("isci rapor");
    }

    @Override
    public void maaş() {
        System.out.println("isci maaş");
    }

    @Override
    public void katsayı() {
        System.out.println("isci katsayı");
    }
}
```

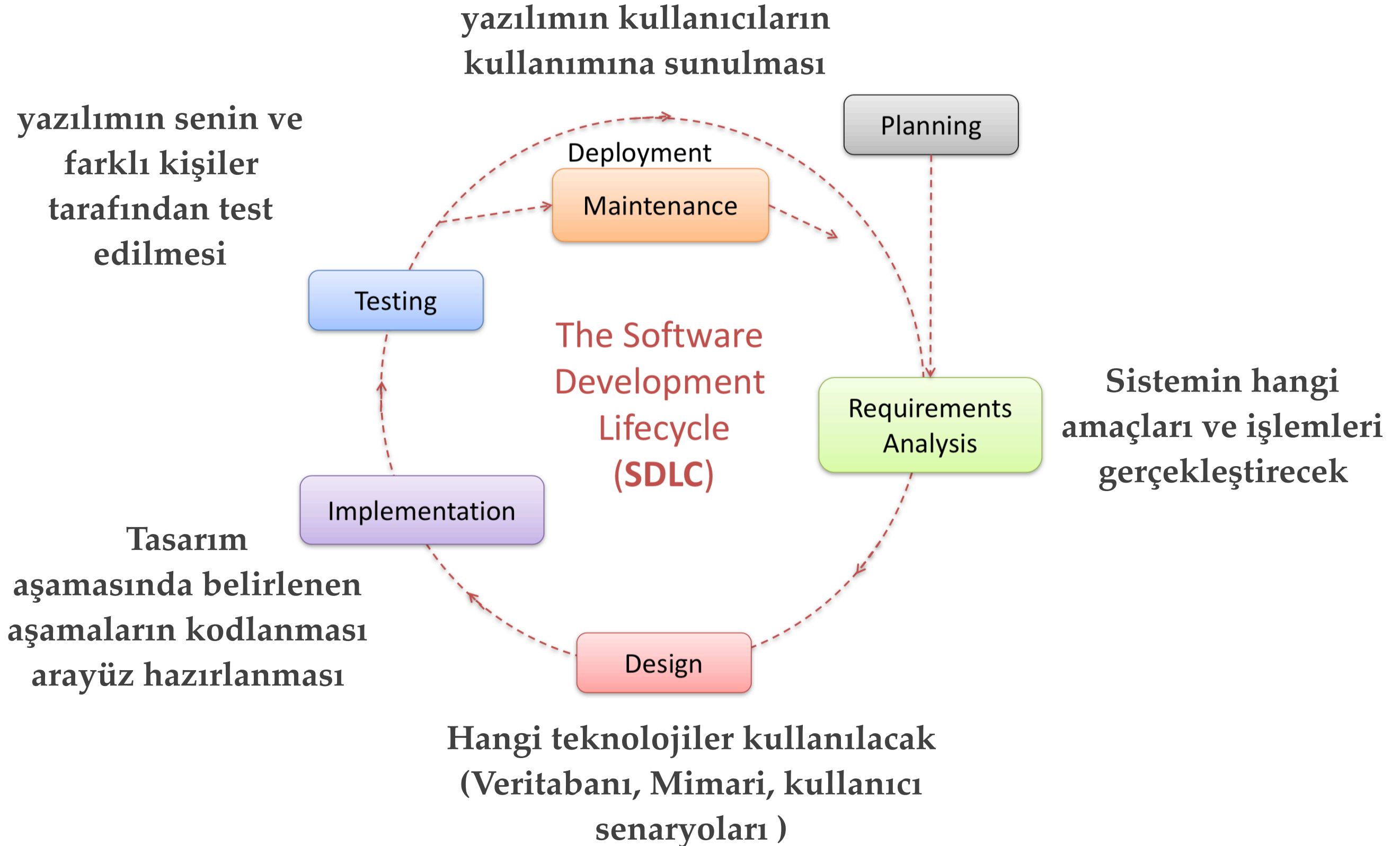
```
/**@author ahmetarifaydin*/
package pkginterface;

public class kadroluişçi extends işçi {

    @Override
    public void maaş() {
        System.out.println("kadrolu isci maaş");
    }

    @Override
    public void katsayı() {
        System.out.println("kadrolu isci katsayı");
    }
}
```

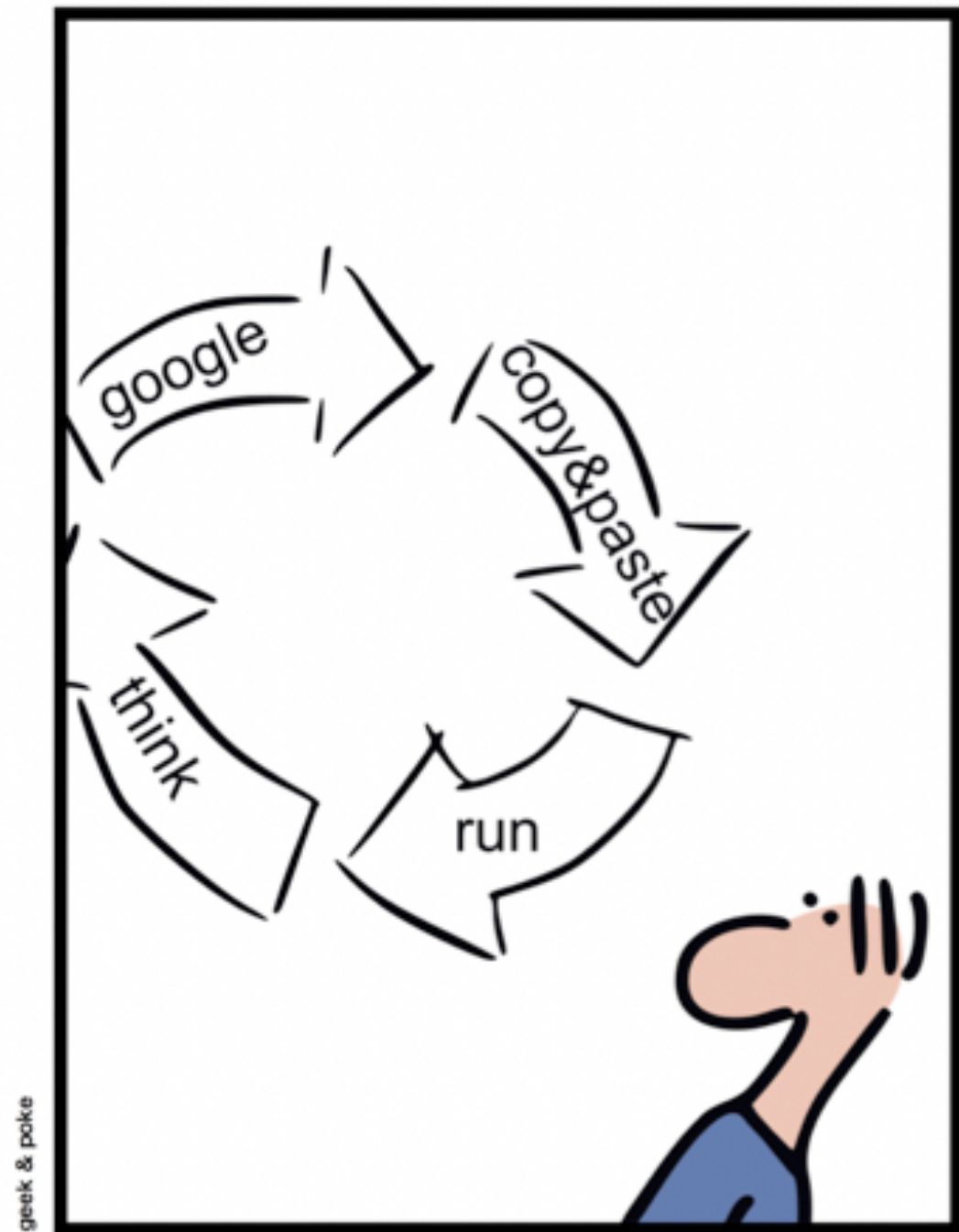
# Software Development Lifecycle (Yazılım Geliştirme Döngüsü)



# *Software Development Lifecycle (Yazılım Geliştirme Döngüsü)*

---

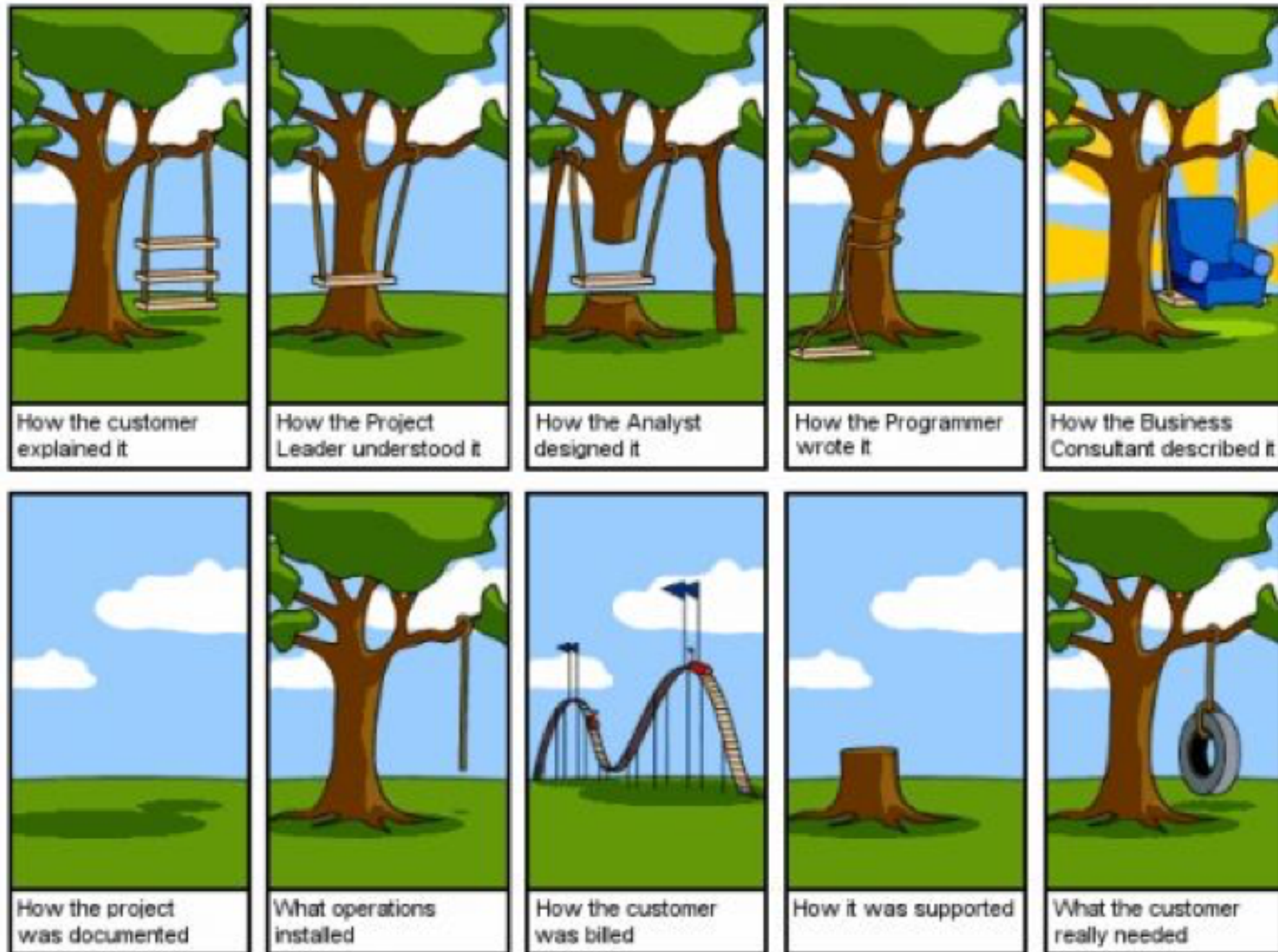
SIMPLY EXPLAINED



DEVELOPMENT CYCLE



# Software Development Lifecycle (Yazılım Geliştirme Döngüsü)



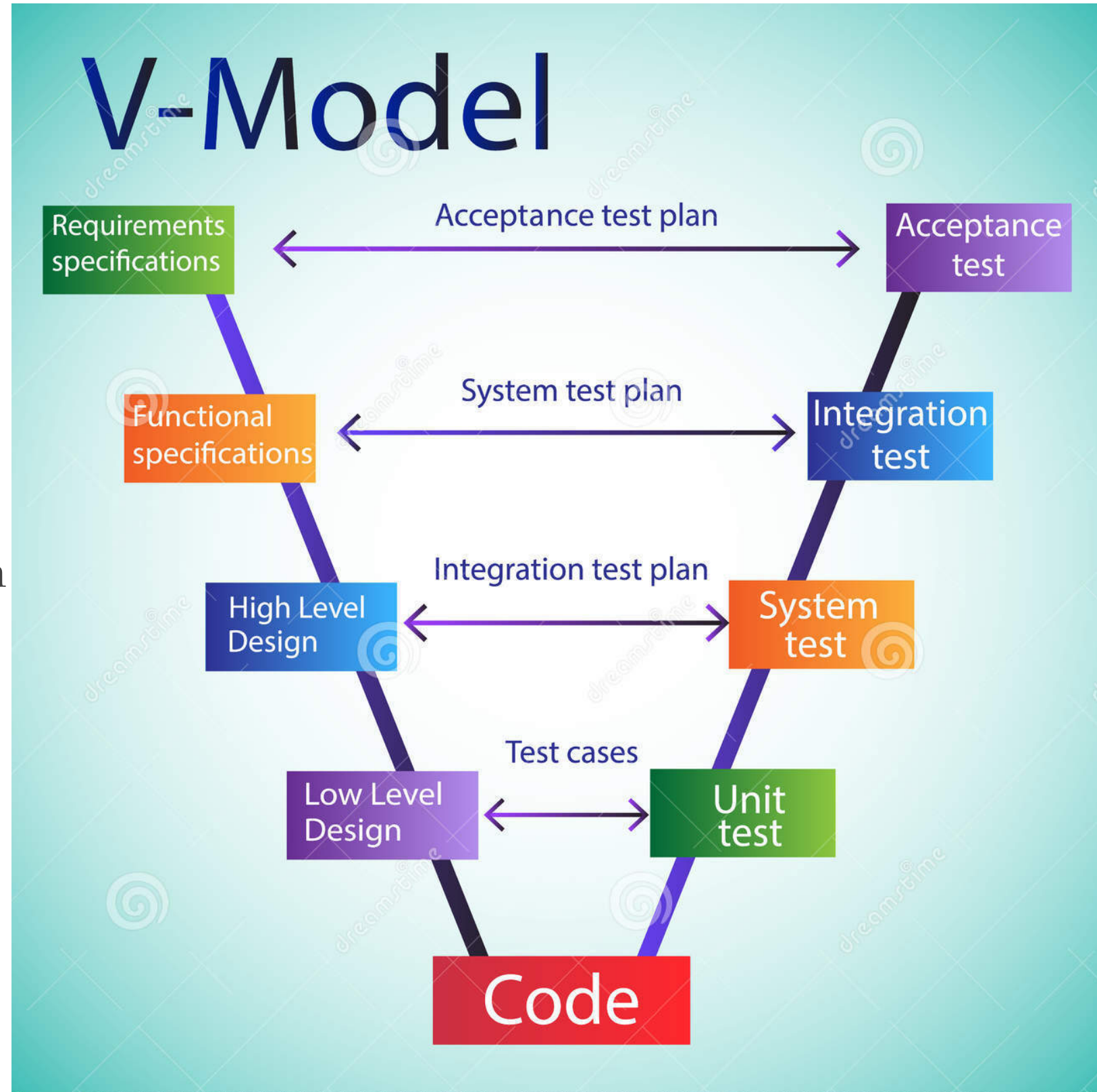
# Yazılım Geliştirme Metodları: Şelale (Waterfall)



- ❖ **Avantajlar:** Anlaşılması ve uygulanması kolay, her seferinde bir aşama üzerinde çalışılıyor
- ❖ **Dezavantajlar:** İhtiyaçların kesin olarak belirlenmesinin zor oluşu, kullanıcıların geliştirilen sistemin son şeklini kullanmaları, uzun süreli ve karmaşık projeler için uygun değil

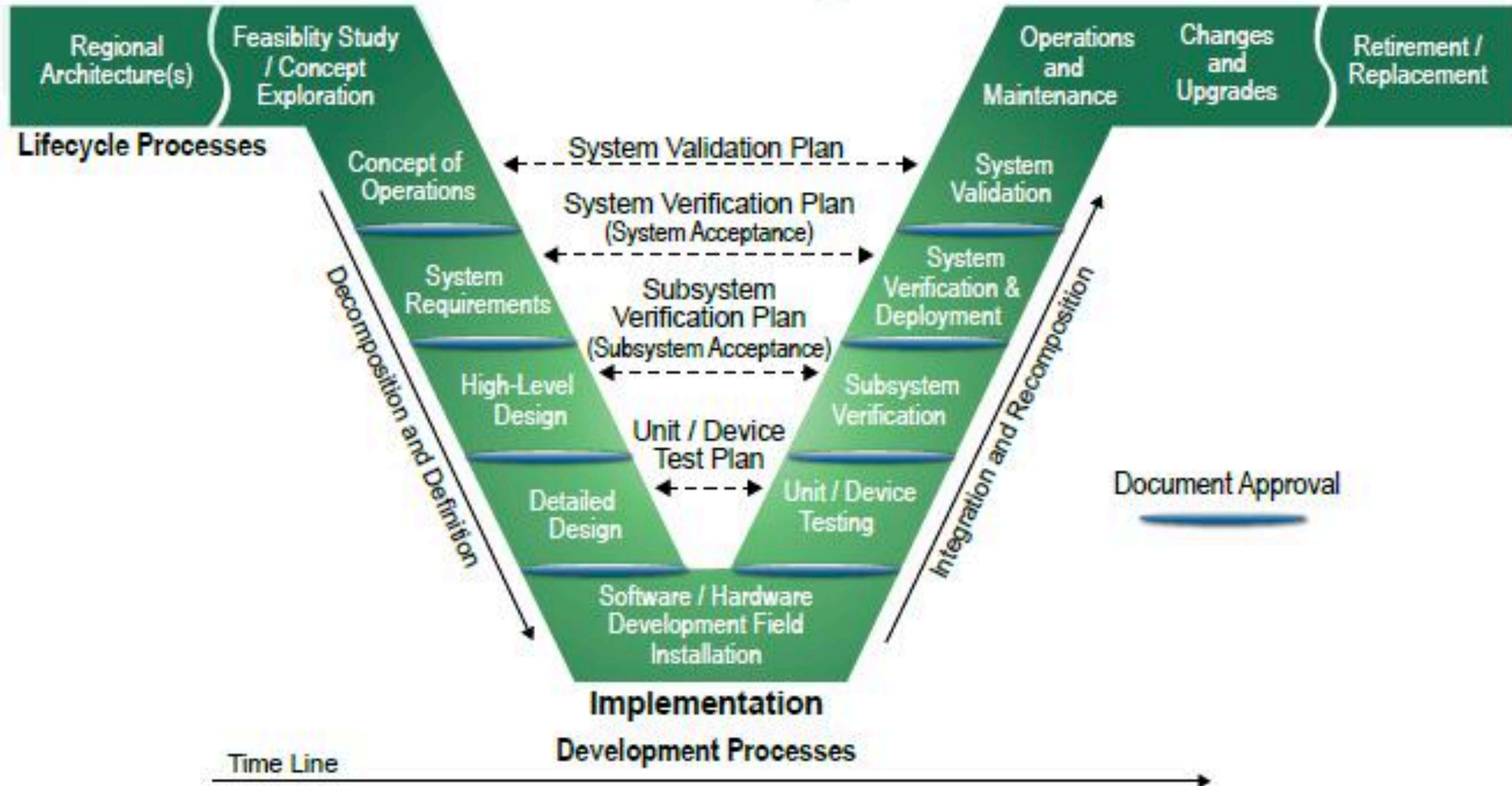
# Yazılım Geliştirme Metodları: V Model

- ❖ Waterfall metodunun geliştirilmiş hali
- ❖ Kısa süreli ve karmaşık olmayan projelerde kullanılabilir
- ❖ Verification and Validation
- ❖ En son aşamaya kadar çalışan bir örnek oluşturulmaz
- ❖





# Yazılım Geliştirme Metodları: V Model



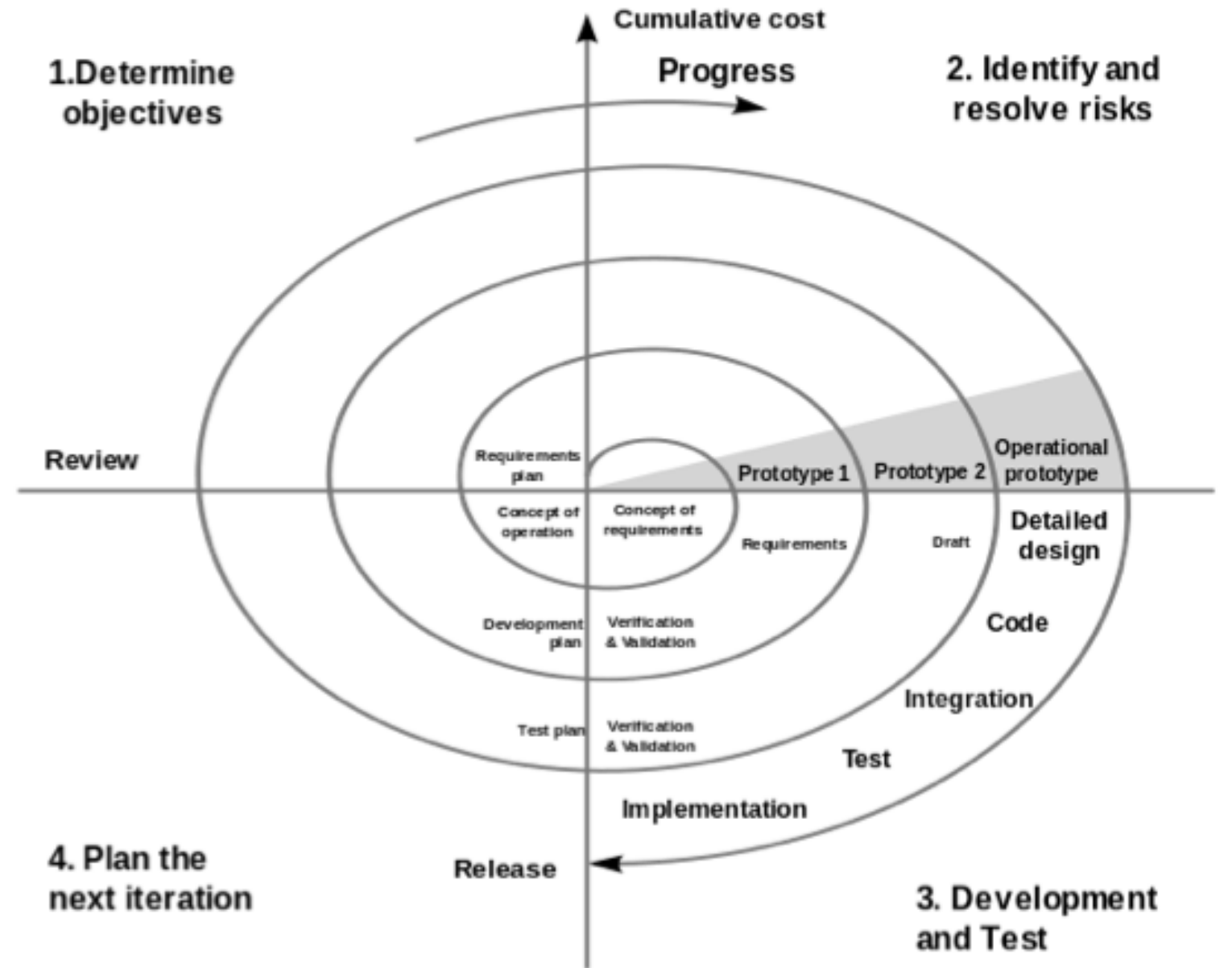
# Yazılım Geliştirme Metodları: Spiral

## ❖ Sarmal (Spiral) Model

- ❖ Değişim isteklerine cevap verir
- ❖ kullanıcılara erkenden prototipi görme ve kullanma imkanı sağlar
- ❖ Yazılım eklemeli olarak devam eder

## ❖ Dezavantajlar

- ❖ Yönetimi zordur
- ❖ Proje belirlenen sürede tamamlanamayabilir
- ❖ Çok yoğun dokümantasyon gerekmektedir

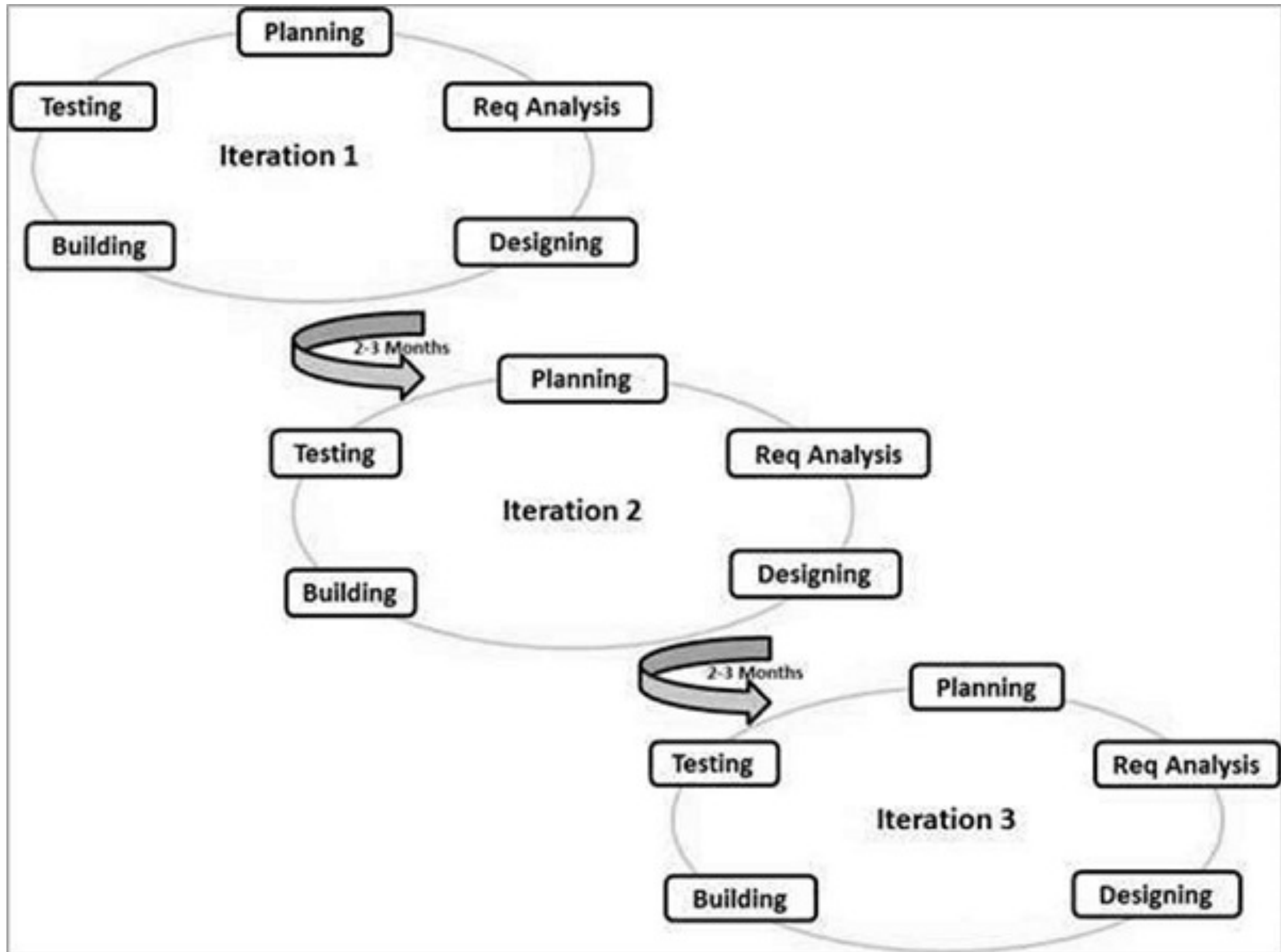


# *Yazılım Geliştirme Metodları: Agile (Çevik)*

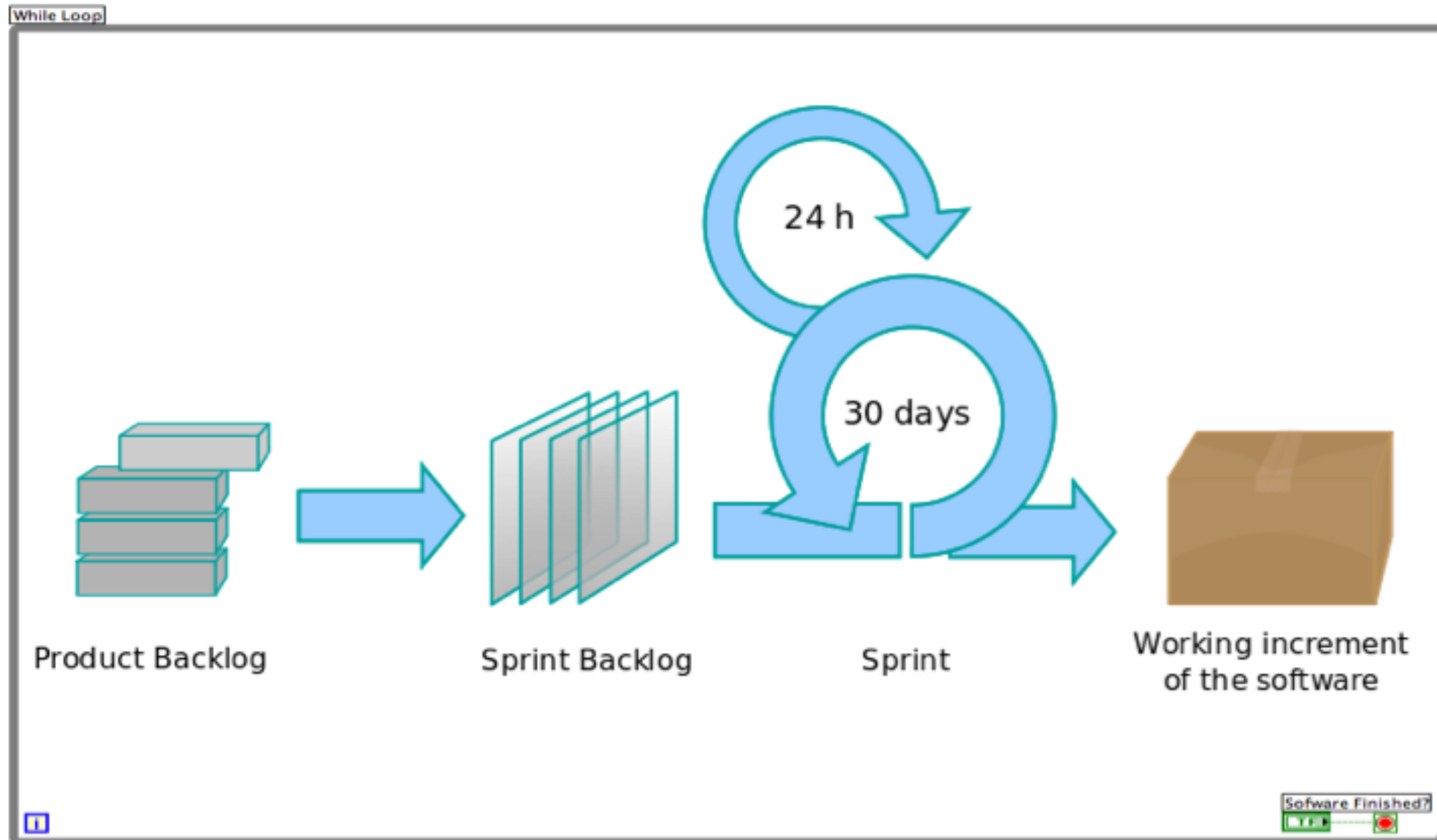
---

- ❖ Projeler için planlama , ihtiyaç tanımlanması ve kullanılacak yöntemler farklıdır
- ❖ Problem alt aşamalara bölünüp eklemeli (incremental) olarak gerçekleştirilir
- ❖ Popüler Agile metodlar:
  - ❖ Rational Unified Process (1994)
  - ❖ Scrum (1995)
  - ❖ Crystal Clear
  - ❖ Extreme Programming (1996)
  - ❖ Adaptive Software Development
  - ❖ Feature Driven Development
  - ❖ Dynamic Systems Development Method (DSDM) (1995)

# Yazılım Geliştirme Metodları: Agile (Çevik)



# Software Development Methods: Scrum





# Software Development Methods: Extreme Programming (XP)

---

## Planning/Feedback Loops

