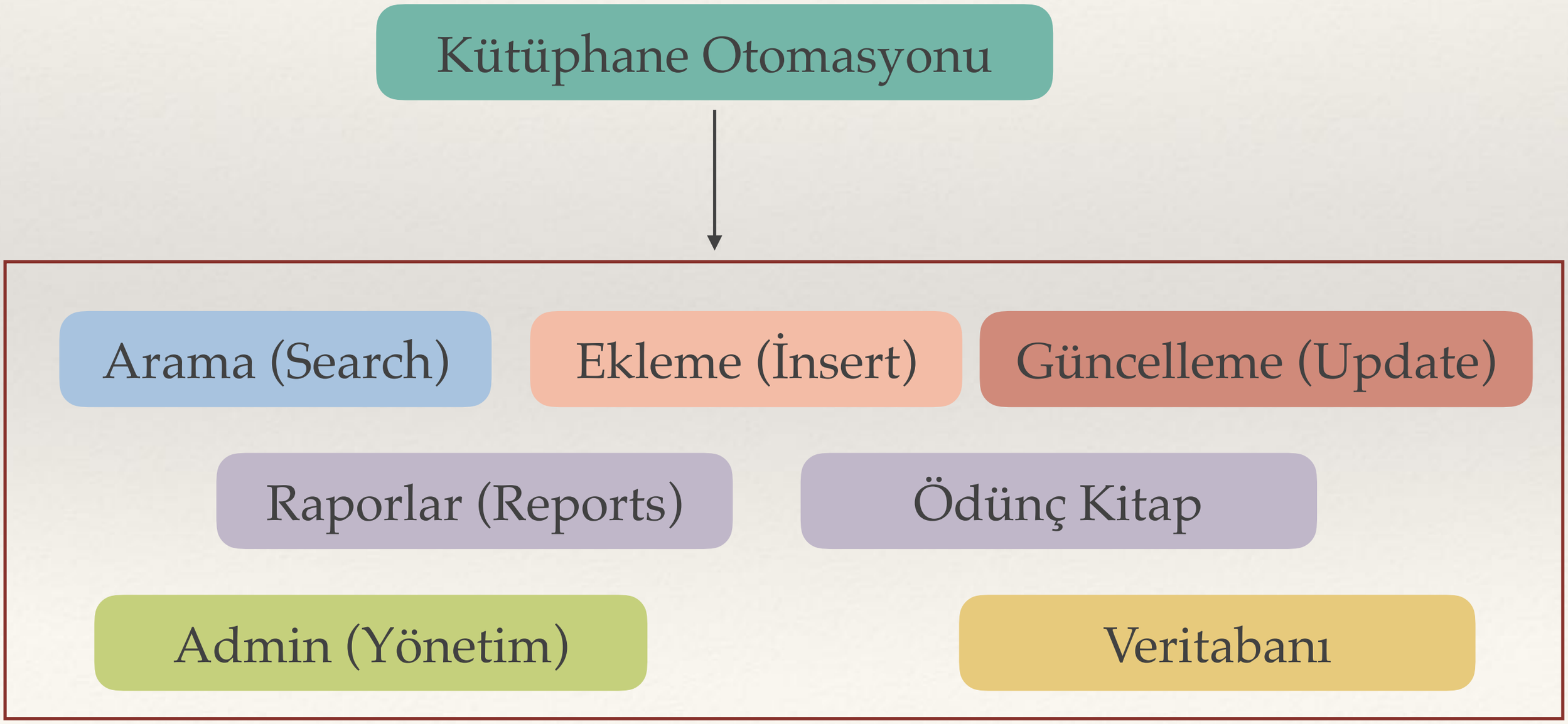
Nesneye Yönelik Yazılım Mühendisliği (376)

Yrd. Doç. Dr. Ahmet Arif AYDIN

Object-Oriented Design: Functional Decomposition

- Karmaşık bir problemi daha anlaşılır ve çözülebilir alt aşamalara ve parçalara ayrıştırıp çözme işlemidir (*process of taking a complex process and breaking it down into its smaller, simpler parts)*



Object-Oriented Paradigm: Abstraction

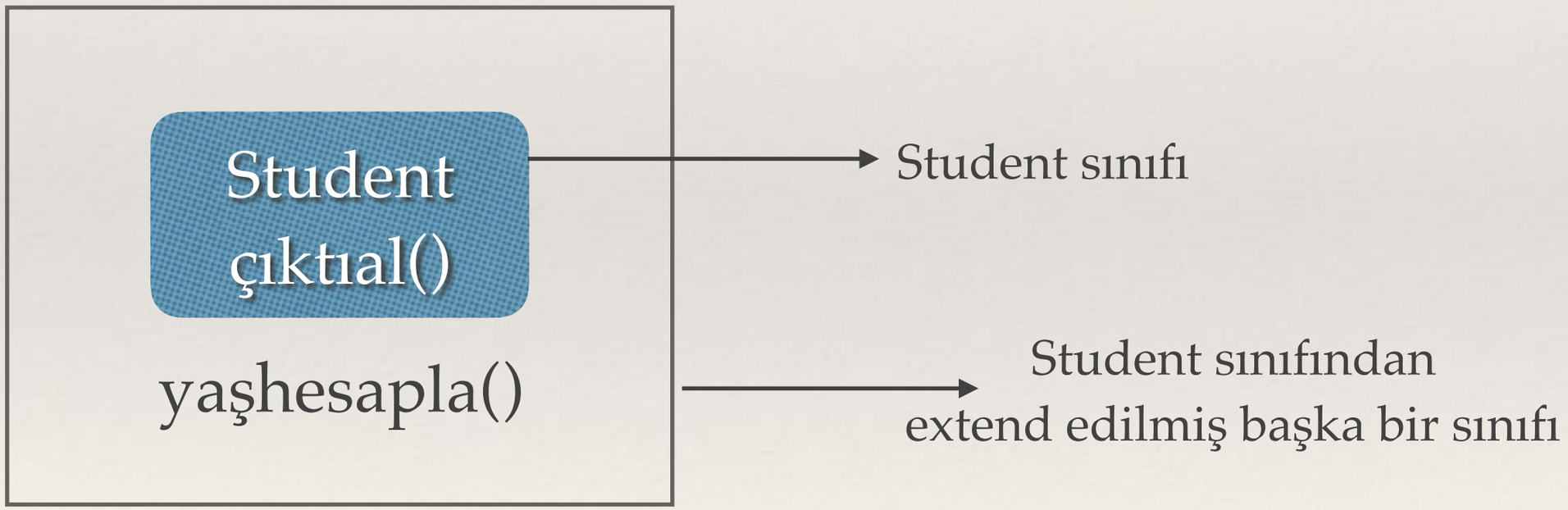
- Bir varlığın belirli amaçları gerçekleştirmek , bir görevi veya problemi çözmek için sağlamış olduğu tanımlamadır.
  - Assembly dili makine dili için , yüksek seviyeli programlama dilleri assembly dili için bir *abstraction* dır.
  - Bir sınıf içerisinde tanımlanan **değişkenler, methodlar** , sınıfa 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 cıkarma(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.çıkarma(a, b);
}
```

Object-Oriented Paradigm: Inheritance

- Bir sınıfın başka bir sınıftan özelliklerini ve metodlarını kalıtsal olarak devralmasıdır (p*rocess of acquiring properties of another class*)
- Kalıtım ile devralınan özelliklere yenileri eklenip oluşturulan yeni sınıf genişletilebilir (**extends**)



Object-Oriented Paradigm: Inheritance

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

    private void katsayı(){
       this.katsayı=25;
    }
   public int katsayıgönder(){
        katsayı();
       return katsayı;
    }
```

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: Encapsulation

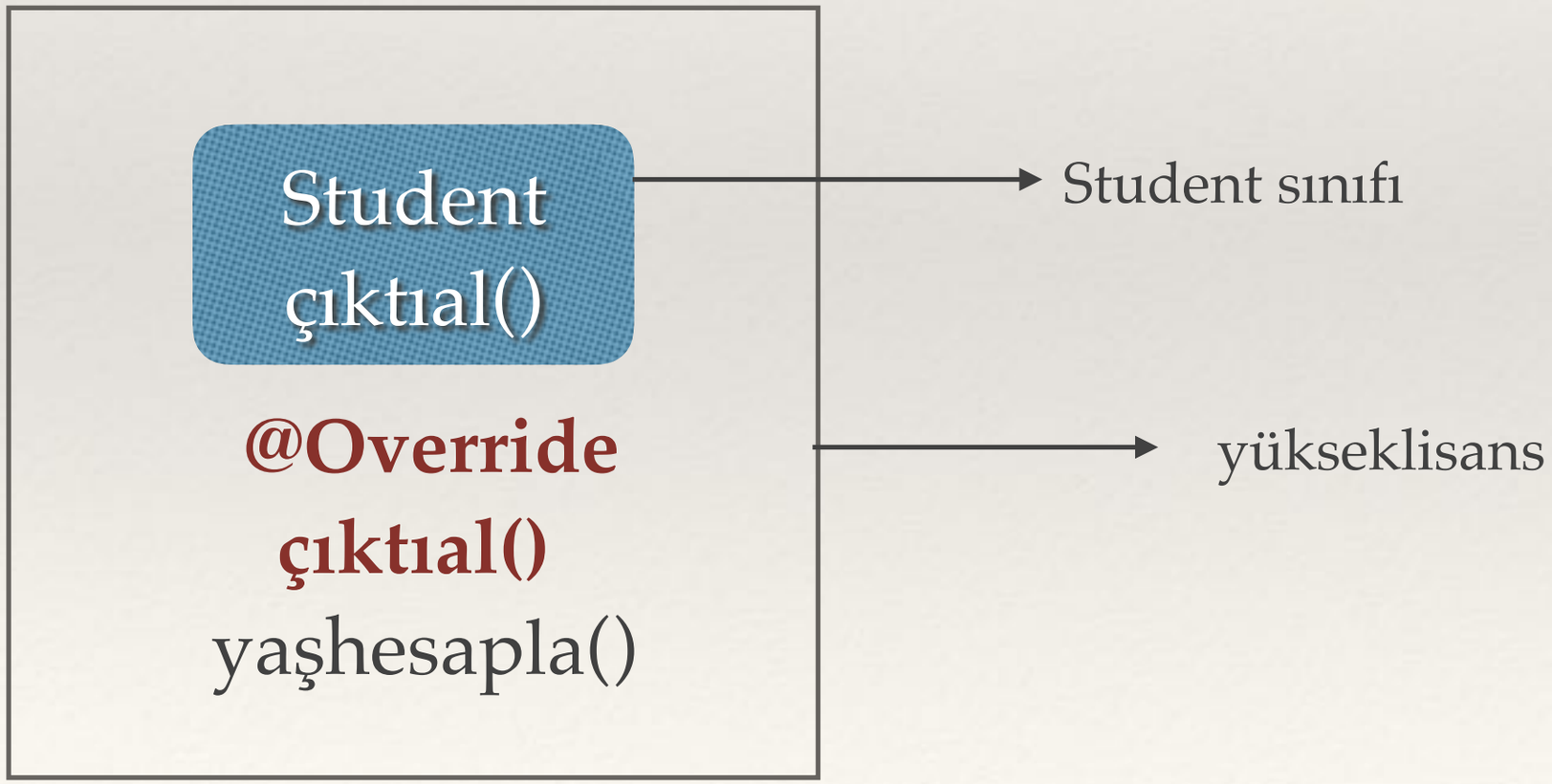
```
public class Student{
  private String name;

  public String getName(){
    return name;
  }
  public void setName(String name){
  this.name=name
  }
}
```

```
class Test{
    public static void main(String[] args){
      Student s=new Student();
      s.setName("kemal");
      System.out.println(s.getName());
    }
}
```

https://www.javatpoint.com/encapsulation

Object-Oriented Paradigm: Polymorphism

- 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: Polymorphism

```
public interface çalışan {
        public void rapor();
        public void maaş();
        public void katsayı();
}
```

```
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");
    }
}
```

```
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");
     }
}
```

Functional Decomposition Problems

- **weak cohesion**
  - bir çok işlemin ve amacın gerçekleştirilmesi
  - tek bir işlem üzerinde yogunlaşılmaması
  - bir metodun bir den fazla işlemi gerçekleştirmesi (ekle, sil, güncelle)
- **tight coupling**
  - çok fazla bağımlılığın olması

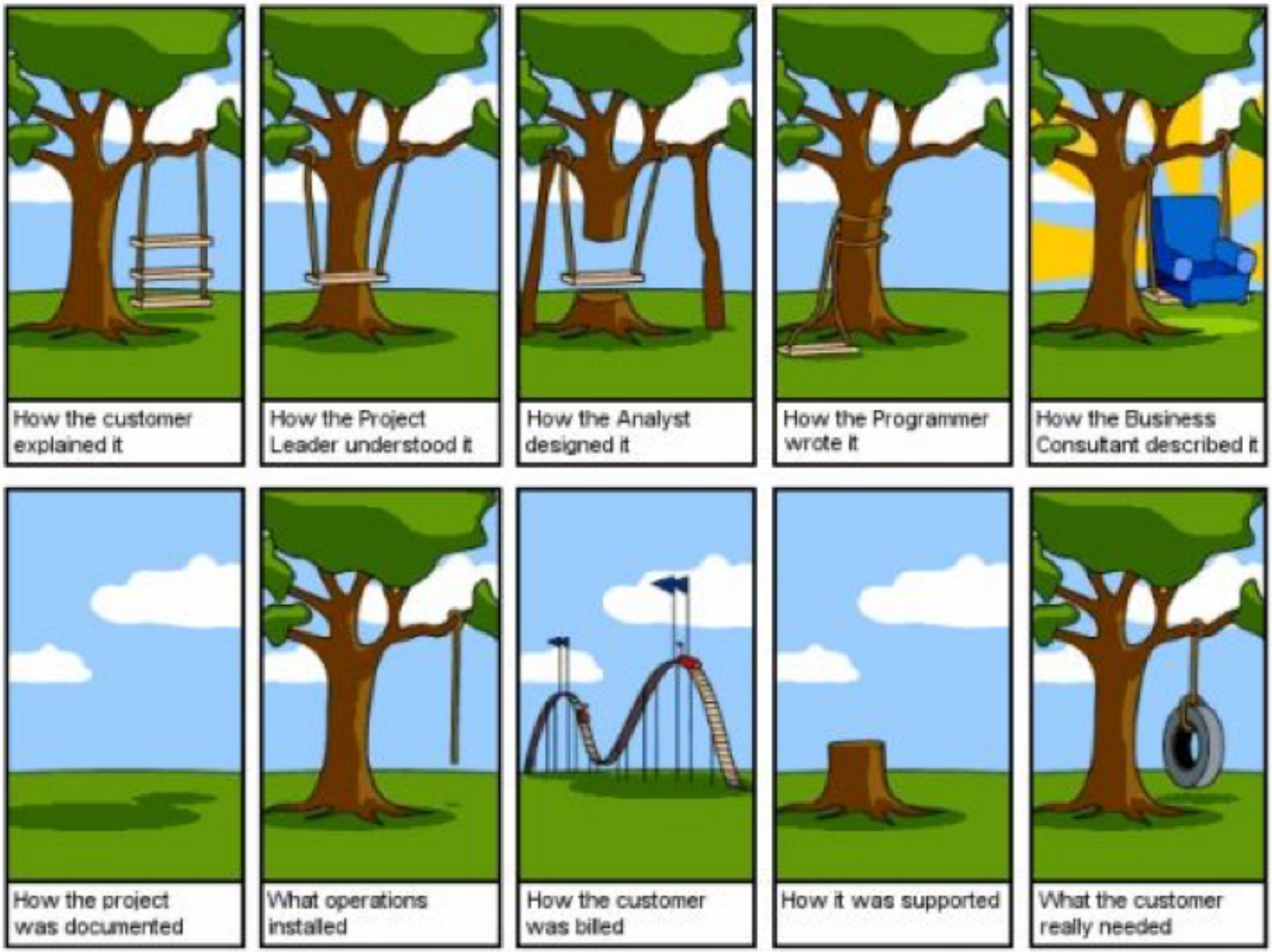
- **we want to build *highly cohesive* and *loosely coupled* systems.**

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 herbir nesnenin görevi, yapacagı iş tanımlı ve kendi sorumluluğunda

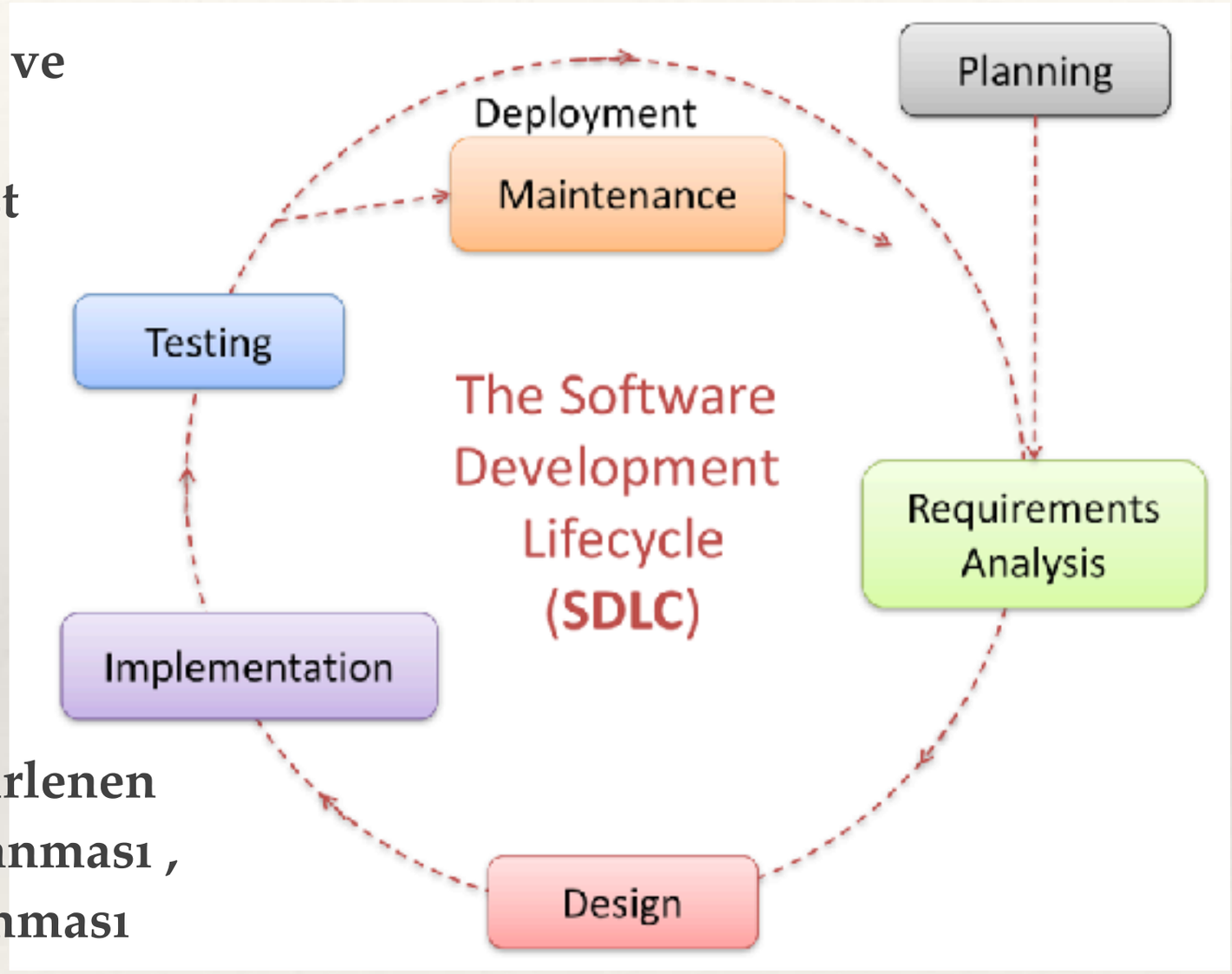
Requirements (İhtiyaçlar)



Software Development Lifecycle (Yazılım Geliştirme Döngüsü)

yazılımın kullanıcıların kullanımına sunulması

yazılımın senin ve farklı kişiler tarafından test edilmesi



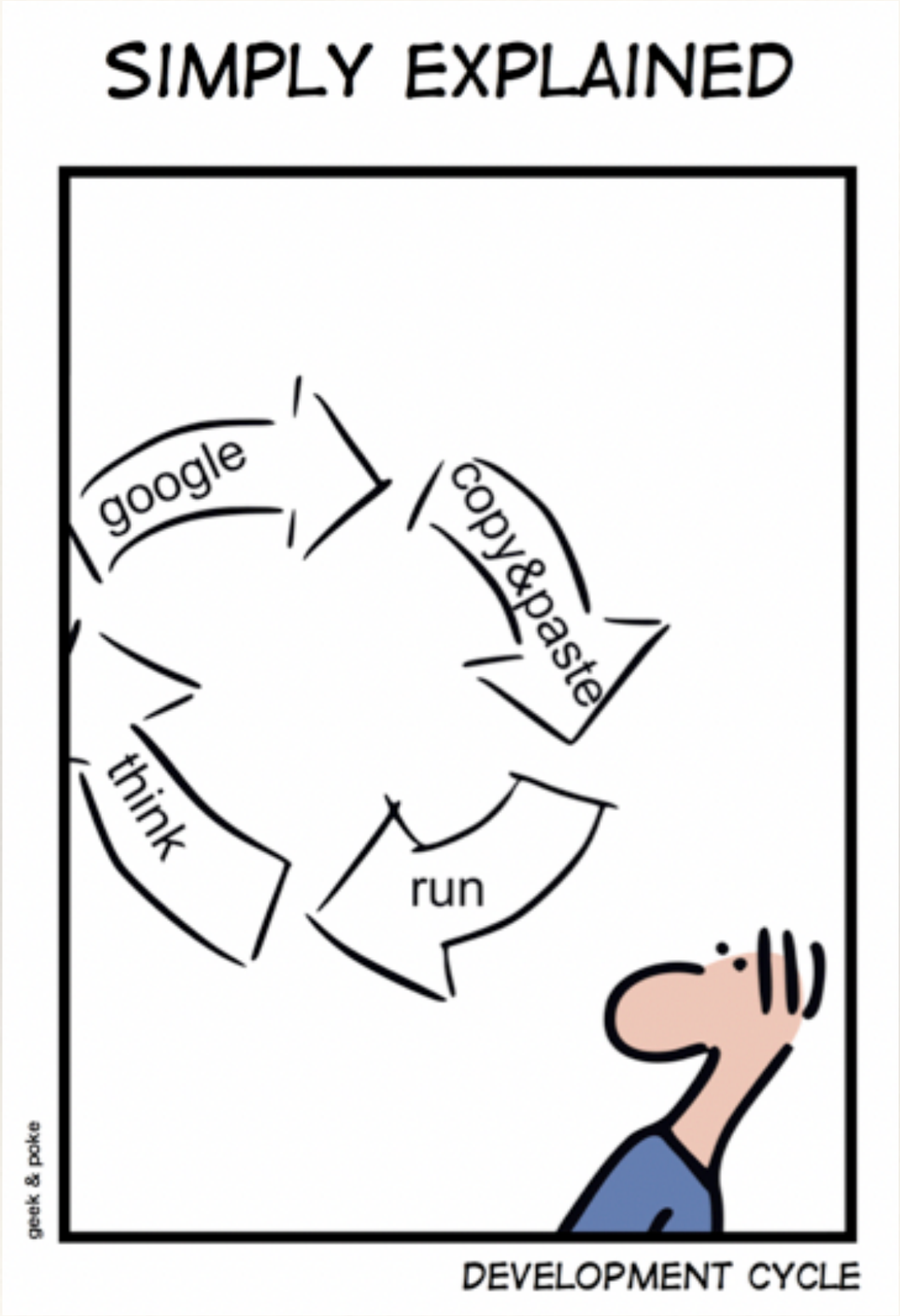
Sistemin hangi amaçları ve işlemleri gerçekleştirecek

Tasarım aşamasında belirlenen aşamaların kodlanması , arayüz hazırlanması

Hangi teknolojiler kullanılacak (Veritabanı, Mimari, kullanıcı senaryoları)

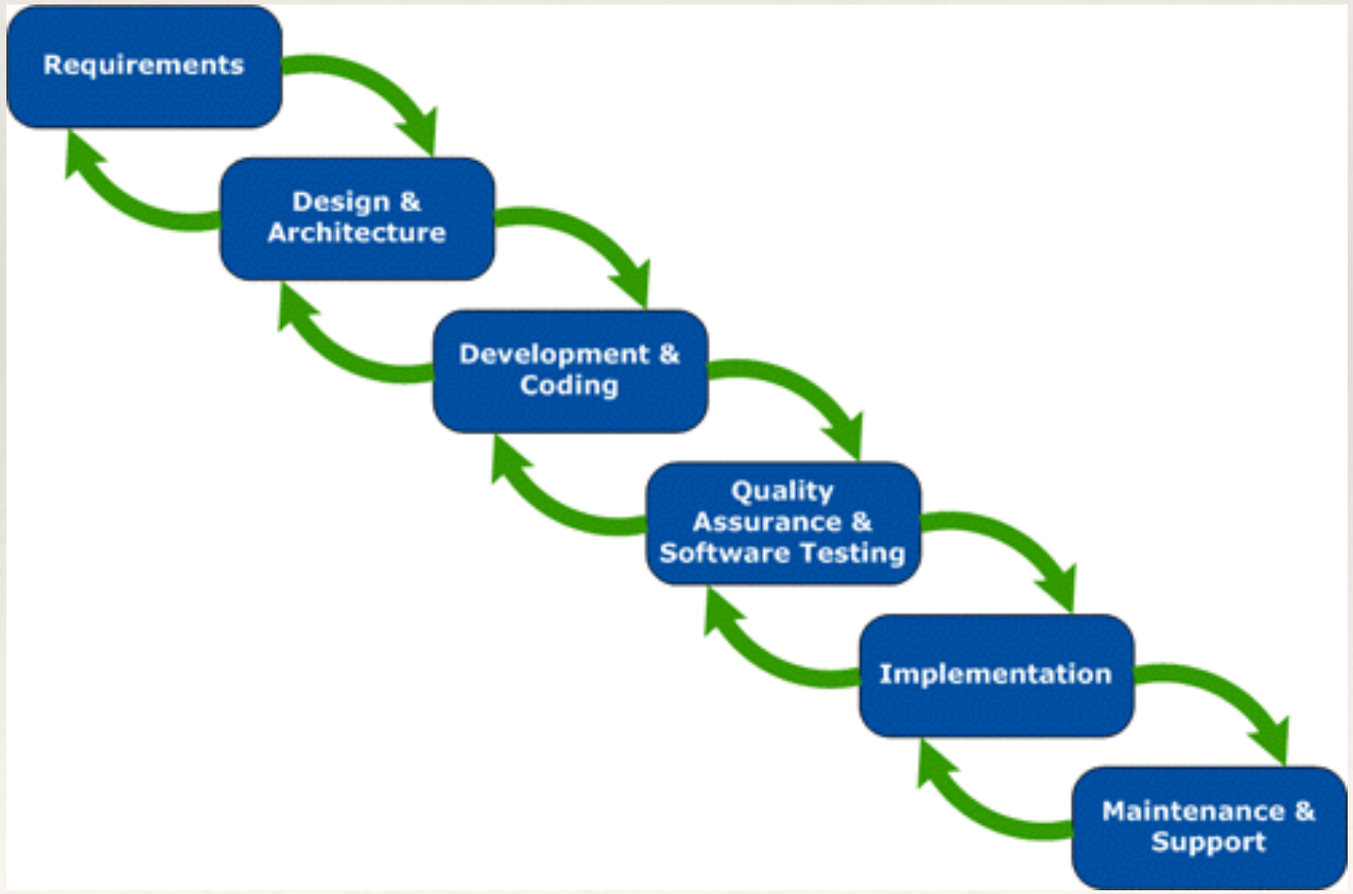
https://www.halvorsen.blog/documents/programming/software_engineering/software_engineering.php

Software Development Lifecycle (Yazılım Geliştirme Döngüsü)



Software Development Methods: Waterfall

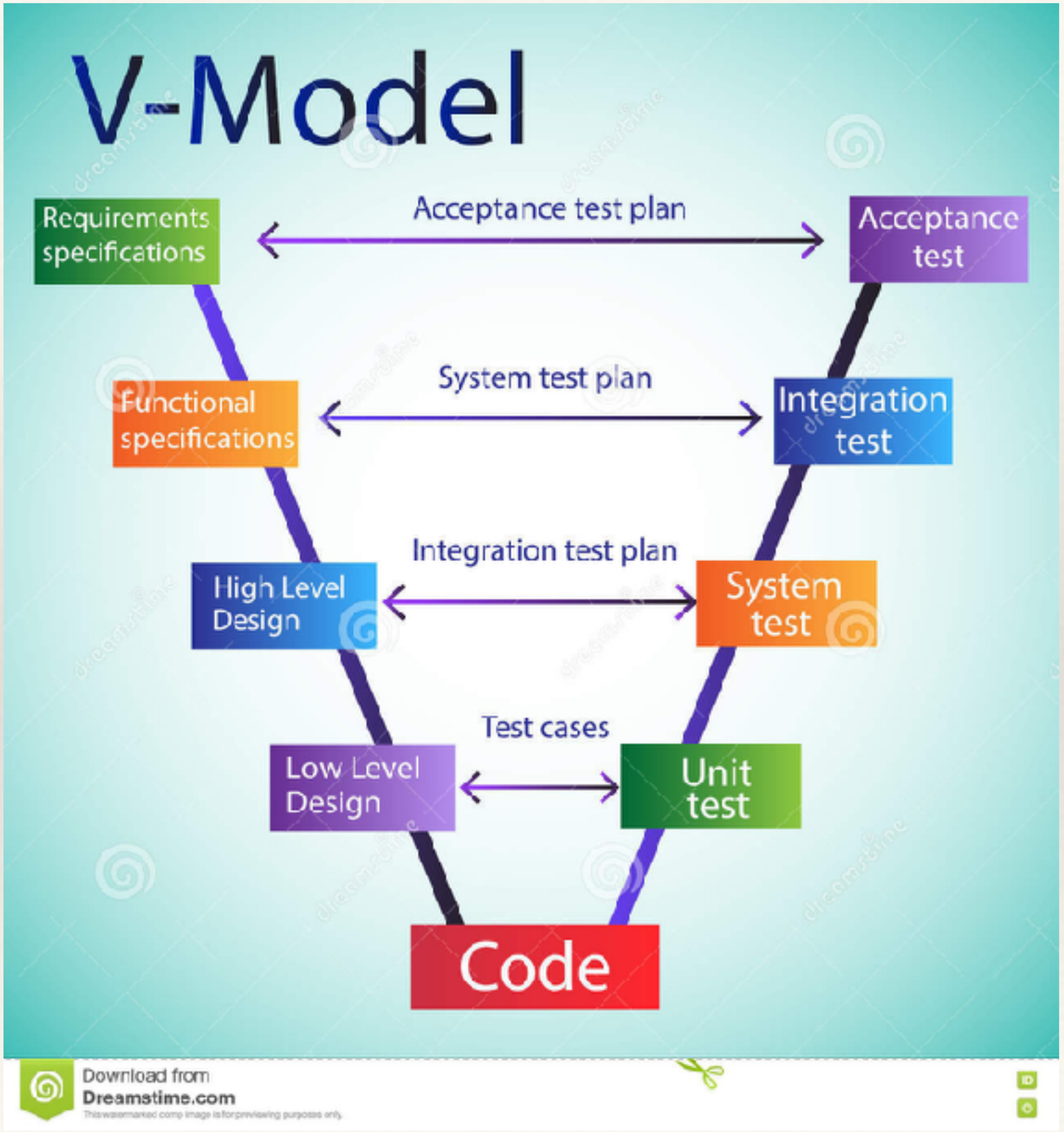
- Her bir aşama üzerinde detaylı olarak çalışılır ve aşama tamamlanınca bir sonraki aşamaya geçilir.
- **Avantajlar:** Anlaşılması ve uygulanması kolay, her seferinde bir aşama üzerinde çalışılıyor,
- **Dezavantajlar**: İhtiyaçların kesin olarak belirlenmesinin zor oluşu, kullanıcıların geliştirilen sistemin son şeklini kullanmaları, uzun süreli ve karmaşık projeler için uygun değil



https://complextester.wordpress.com/2012/08/07/spiral-prototype-fish-bone-sashimi/

Software Development Methods: V Model

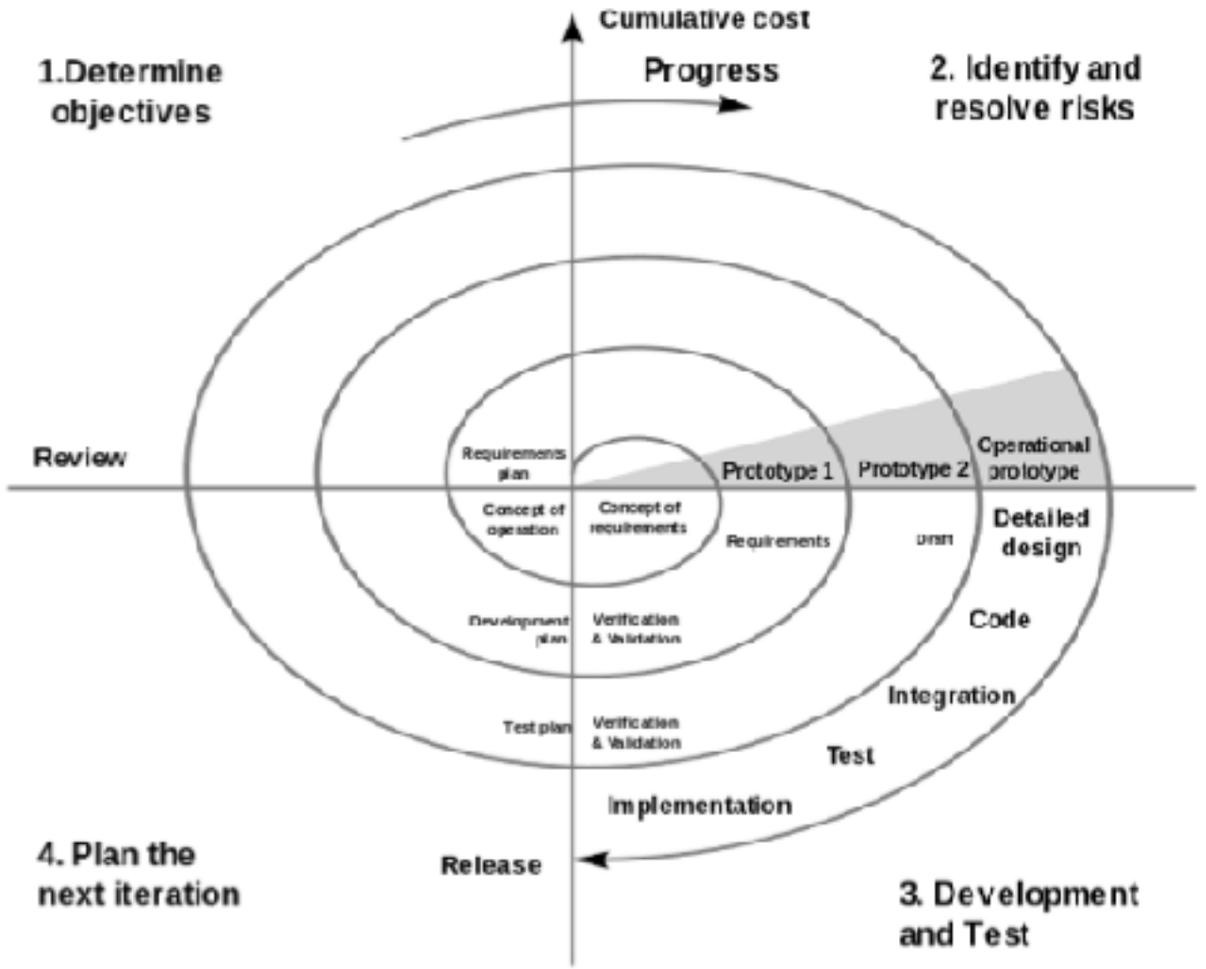
- Waterfall metodunun geliştirilmiş hali



Software Development Methods: Spiral

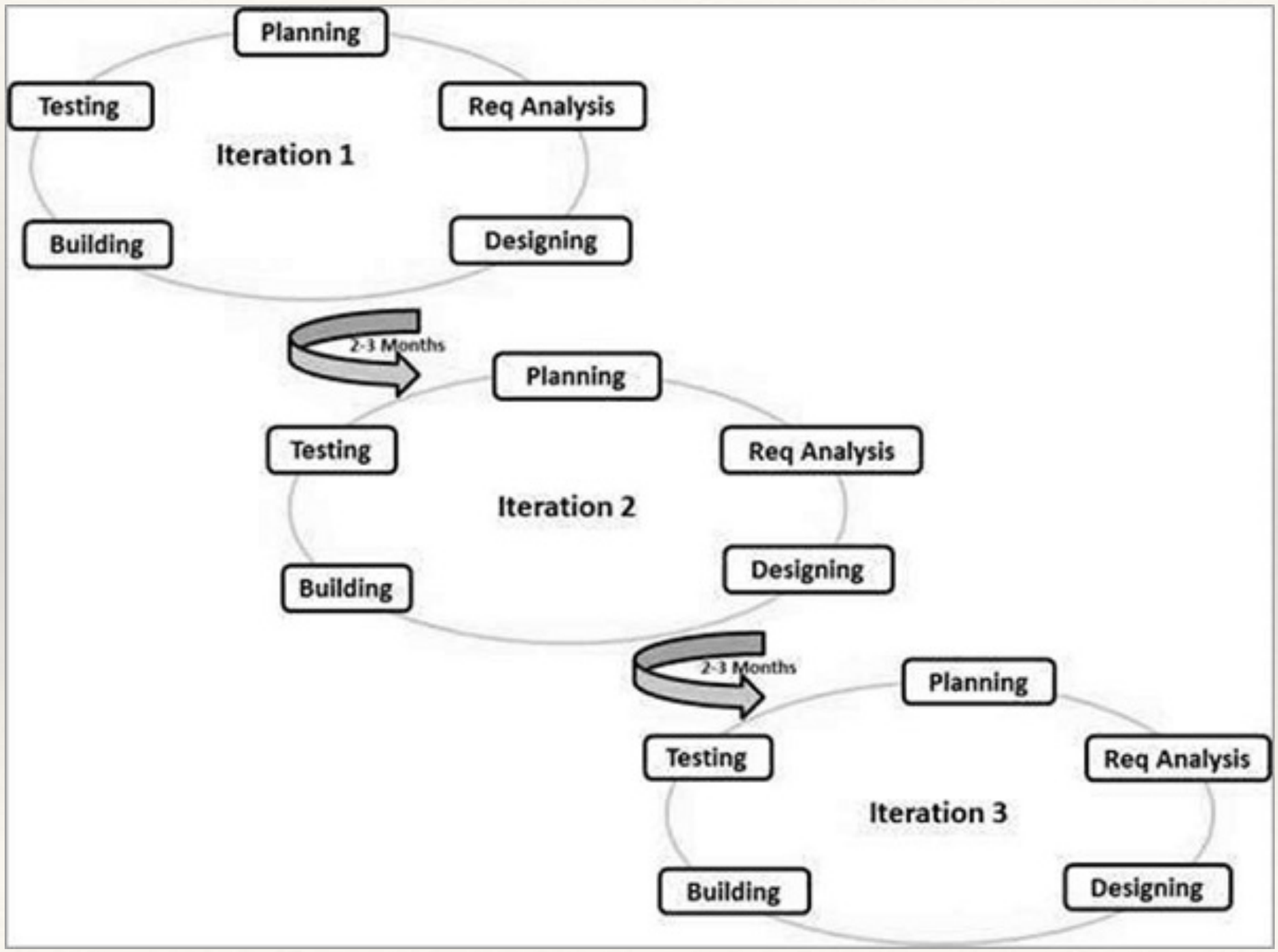
- **Sarmal (Spiral) Model**
  - Değişim isteklerine cevap verir
  - kullanıcılara erkenden prototipi görme ve kullanma imkanı sağlar
  - Yazılım eklemeli olarak devam eder

- **Dezavantajlar**
  - Yönetimi zordur
  - Proje belirlenen sürede tamamlanamayabilir
  - Çok yogun dokümantasyon gerekmekte

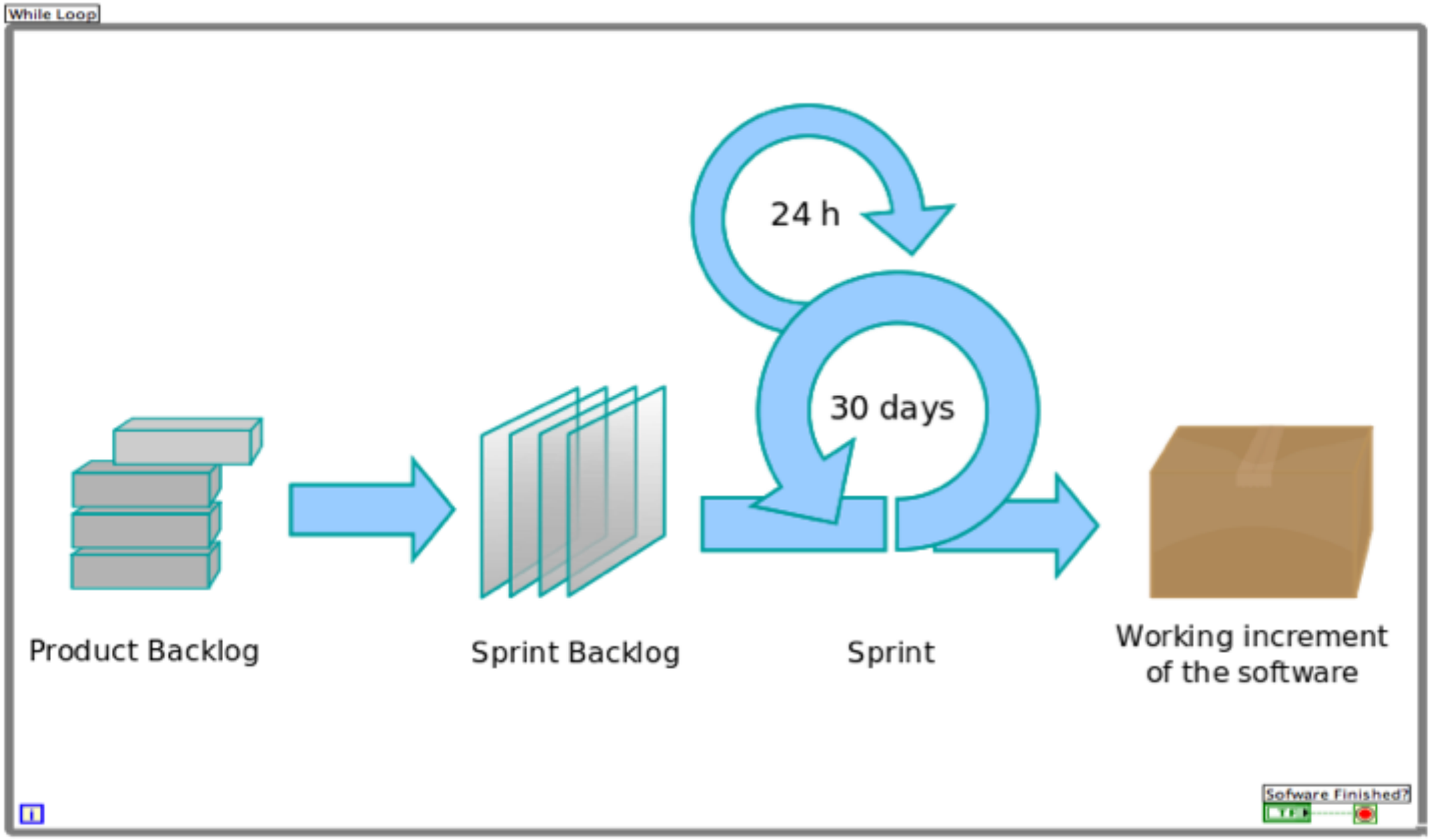


Software Development Methods: Agile

- Projeler için planlama , ihtiyaç tanımlanması ve kullanılacak yöntemler farklıdır
- Problem alt aşamalara bölünüp eklemeli (incremental) olarak geçekleştirilir



Software Development Methods: Scrum



Software Development Methods: Extreme Programming (XP)