**Nedir?**

**Yararlar zararlar**

**Güncel yaklaşımlar**

**Öncelikle nedir?**

Yeniden kullanım tüm mühendislik dallarında kendine önemli yerler edinmiştir. Bu sayede daha kısa zamanda daha çok üretim yapıp, daha verimli çalışabilmek mümkün olmaktadır. Yazılımın yeniden kullanılabilirliği; gereksinimler, tasarımlar, test durumları vs.. yeni yazılımların üretilmesinde kullanılması olarak tanımlanmaktadır.

**Yararları :**

1.Zaman Tasarrufu: Daha önce geliştirilmiş ve test edilmiş bileşenlerin veya kütüphanelerin tekrar kullanılması, yeni projelerin daha hızlı bir şekilde tamamlanmasını sağlar

2. Maliyet Tasarrufu: Yeniden kullanılabilir yazılım bileşenleri veya kod parçaları, yeni yazılımın geliştirilmesi için her seferinde sıfırdan başlama ihtiyacını azaltır.

3. Kalite ve Güvenilirlik: Yeniden kullanılabilirlik, daha önce test edilmiş ve olgunlaştırılmış bileşenlerin yeni projelerde kullanılmasını sağlar.

4. Standartlaştırma: Yeniden kullanılabilirlik, yazılım geliştirme sürecini standartlaştırmayı teşvik eder.

5. Kaynak Verimliliği: Yazılım geliştirme ekibinin kaynaklarını daha etkili bir şekilde kullanmasını sağlar. Yeniden kullanılabilir kod veya bileşenler, yazılımın farklı parçalarında tekrar tekrar kullanılabilir.

**Zararları:**

1. \*Uyum Sorunları:\* Yeniden kullanılan bir bileşen veya kütüphane, yeni bir projeye tam olarak uyum sağlamayabilir. Bu uyumsuzluklar, ek çaba gerektirerek zaman ve kaynak kaybına neden olabilir.

2. \*Bakım Zorlukları:\* Yeniden kullanılabilir yazılımın bakımı, her projede güncel ve güvenli kalmasını sağlamak için önemlidir. Farklı projelerde kullanılan bileşenlerin bakımı karmaşık olabilir.

3. \*Veri Gizliliği ve Güvenlik Sorunları:\* Yeniden kullanılan bileşenler, hassas verilere veya güvenlik zafiyetlerine sahip olabilir. Bu, gizlilik ve güvenlik sorunlarına yol açabilir.

4. \*Kod Karmaşıklığı:\* Yeniden kullanılabilirlik, bazen kod karmaşıklığını artırabilir. Çünkü her projenin ihtiyaçları farklı olabilir, bu da çok amaçlı bir bileşenin daha karmaşık hale gelmesine yol açabilir.

5. \*İlgili Dokümantasyon Eksikliği:\* Yeniden kullanılan bileşenlerle ilgili yetersiz veya eksik dokümantasyon, bu bileşenlerin doğru bir şekilde kullanılmasını zorlaştırabilir.

6. \*Dil ve Platform Kısıtlamaları:\* Yeniden kullanılabilirlik, belirli bir programlama dili veya platforma bağlı olabilir. Bu, yazılımın taşınabilirliğini veya çoklu platformlarda kullanılabilirliğini kısıtlayabilir.

7. \*Mülkiyet ve Lisans Sorunları:\* Yeniden kullanılabilir bileşenlerin mülkiyet hakları ve lisansları karmaşık olabilir. Bu, projenin yasal ve mülkiyet sorunlarıyla karşılaşmasına neden olabilir.

Güncel yaklaşımlar

Yazılım mühendisliğindeki tüm güncel yaklaşımlar, Software Reuse'un artırılmasını hedeflemektedir. Yeniden kullanım tarihine baktığımızda, ilk olarak fonksiyonlar bazında yeniden kullanım göze çarpmaktadır. Bu Software Reuse'ın en basit tekniğidir. Bu tip Software Reuse'da mühendislerin bilgisine bağımlılık fazla olduğundan çok verimlilik sağlanamamaktadır. Bir sonraki Software Reuse aşaması kütüphanelerdir. Kütüphaneler, kullanıcılarına bir tür arayüz sunabilmektedirler ve bu sayede Software Reuse'da kolaylık sağlarlar. Ancak kütüphanelerin yapılandırılabilirlikleri düşüktür ve kütüphane kullanarak geliştirilen uygulamalar, bu kütüphanelere çok bağımlı kalmaktadırlar. İskeletle (Çatı, Framework), Software Reuse konusunda bir adım öndedirler. İskeletlerde, kütüphanelerdekinin aksine kontrolün terse dönmesi kavramı mevcuttur. Kütüphaneler kullanılırken, yazılımcı hakimiyeti elinde tutmaktadır. İskeletlerde ise durum farklıdır. İskeleti kullanan, ondan bir istekte bulunur ve iskelet kendi içinde çesitli mekanizmaları - olaylar - tetikleyerek bu istekleri gerçekleştirir.

Yukarıda bahsettiğimiz üç tip Software Reuse da esasen kodun yeniden kullanımıdır. Bu kod nesne yönelimli (object oriented), bileşen yönelimli (component oriented) veya sadece yordamsal (procedural) olabilmektedir. Burada, tahmin edilebileceği gibi yordamsal kod çok alt seviye ve Software Reuse adına kullanışsız kalmaktadır. Nesne yönelimli Software Reuse, devrim niteliği taşımaktadır. Miras (inheritance) sayesinde çok yoğun Software Reuse sağlanabilmektedir.

**Example**

Object oriented üzerinden bir örnek verecek olursak: Sınıf-2, Sınıf-1’in özelliklerine sahip olsun. Bu sayede Sınıf-1’in tüm özellikleri Sınıf-2 için kullanılabilir olacaktır. Ancak bu yaklaşımın da hataya ve esnekliğine zarar verecek gelişmelere açıklığı vardır. Sınıf-1’de yapılan bir değişiklik Sınıf-2’ye doğrudan yansıyacağından istenmeyen sonuçlara sebep olabilir. Nesne tabanlı iskeletlerde (Object oriented framework) bu sıkıntıya sıkça rastlanmaktadır ve bu sorun kırılgan miras yapılı sınıf problemi olarak bilinmektedir (Fragile base class problem).

**Bileşen Odaklı**

Nesneye yönelim paradigmasından sonra bileşen yönelimli yaklaşımların literatürde yer edinmesi ile bileşen tabanlı iskeletler ortaya çıkmaya başlamıştır. Bileşen teknolojisi Software Reuse için büyük öneme sahiptir. Bileşenler tanımlı arayüzleri, ki bu arayüzler sadece bileşenlerin sağladığı veya sunduğu servisleri tanımlamakla kalmayıp; performans, zamanlama, servis kalitesi (Quality of Service, QoS) gibi farklı gerekleri barındırabilme özelliğine sahiplerdir.

Buradaki en önemli nokta, yeniden kullanım için miras değil bileşim (composition) kullanılmasıdır. Bilindiği üzere miras, derleme zamanında sabitlenmektedir. Ancak bileşim, çalışma zamanında da değiştirilebilir. Bu sayede bileşen tabanlı iskeletler nesne tabanlı iskeletlere göre çok büyük esneklik kazanabilmektedir.