Yazılım Mühendisliği
Yazılım Yaşam Döngüsü

Dr. Öğr. Üyesi Emrah ÖZKAYNAK

Yazılım Yaşam Döngüsü Adımları

- ✓1. **Gereksinim Analizi (Requirement Analysis)**
- ✓2. **Fizibilite Çalışması (Feasibility Study)**
- ✓3. **Sistem Tasarımı (System Design)**
- ✓4. **Kodlama (Coding)**
- ✓5. **Test Etme (Testing)**
- ✓6. **Kurulum (Deployment)**
- ✓7. **Bakım (Maintenance)**
- ✓8. **Sonlandırma (Retirement)**

Kodlama

- Yazılım yaşam döngüsünde "kodlama" adımı, belirlenen gereksinimlere ve tasarıma uygun olarak yazılımın yazıldığı aşamadır. Kodlama adımı, çeşitli alt aşamalardan oluşur.

- 1. **Kodlama Planının Yapılması**
- 2. **Modül ve Sınıf Yapısının Oluşturulması**
- 3. **Fonksiyonların Geliştirilmesi**
- 4. **Veritabanı ve Diğer Harici Bağlantıların Kodlanması**
- 5. **Kodun Test Edilmesi ve Hata Ayıklama (Debugging)**
- 6. **Optimizasyon**
- 7. **Kodun Gözden Geçirilmesi (Code Review)**
- 8. **Dökümantasyon**

Kodlama - 1. Kodlama Planının Yapılması

Kodlama Planının Yapılması aşaması, yazılım geliştirme sürecinde kodlama adımının temelini oluşturan bir hazırlık sürecidir.

Bu aşamada, geliştirme sürecinin etkili ve düzenli bir şekilde ilerleyebilmesi için bazı önemli planlamalar yapılır

Kodlama - 1. Kodlama Planının Yapılması

✓**1. Programlama Dillerinin ve Araçların Belirlenmesi**

- **Programlama Dili Seçimi**: Projede kullanılacak programlama dili veya dillerini belirlemek, kodlamanın ilk adımıdır. Programlama dili, projenin ihtiyaçlarına, performans gereksinimlerine, ekibin bilgi birikimine ve hedef platformlara göre seçilir. Örneğin, yüksek performans gerektiren sistemlerde C++ veya Rust gibi diller tercih edilirken, web tabanlı projelerde JavaScript veya Python kullanılması yaygındır.
- **Geliştirme Ortamı (IDE) ve Araçların Belirlenmesi**: Kod yazmak için kullanılacak geliştirme ortamı (IDE), sürüm kontrol sistemi (örneğin, Git), hata ayıklama araçları ve test araçları gibi gerekli tüm yazılımlar ve araçlar belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓**2. Kodlama Standartlarının Belirlenmesi**

- **Kodlama Standartları**: Proje boyunca bir standart belirlemek, ekip üyelerinin kodu daha okunabilir ve anlaşılır şekilde yazmasını sağlar. Bu standartlar, isimlendirme kuralları (örneğin, değişken ve fonksiyon isimleri), kod biçimlendirme kuralları (girintiler, boşluk kullanımı) ve yorum ekleme kurallarını içerir.
- **Güvenlik ve Performans Standartları**: Projede uyulması gereken güvenlik ve performans standartları tanımlanır. Örneğin, şifreleme yöntemleri, güvenlik açıklarının nasıl ele alınacağı ve kodun hızlı çalışması için optimizasyon kriterleri belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓**3. Modüler Yapının ve Sorumlulukların Belirlenmesi**

- **Modül Yapısının Tanımlanması**: Yazılımın modüllere veya katmanlara ayrılarak geliştirilmesi, kodun daha düzenli ve yönetilebilir olmasını sağlar. Kodlama planında, hangi modüllerin hangi işlevleri gerçekleştireceği detaylandırılır. Örneğin, veritabanı işlemlerini yöneten bir modül, kullanıcı arayüzünü oluşturan bir modül vb. şeklinde ayrımlar yapılır.
- **Sorumlulukların Dağıtılması**: Ekip üyeleri arasındaki iş bölümü planlanır. Hangi geliştiricinin hangi modüllerden veya işlevlerden sorumlu olacağı belirlenir. Bu, ekip içindeki iş birliğini artırır ve süreçlerin daha hızlı ilerlemesini sağlar.

Kodlama - 1. Kodlama Planının Yapılması

✓**4. Zaman Çizelgesi ve İş Takviminin Oluşturulması**

- **Kodlama Zaman Çizelgesi**: Kodlama sürecinin hangi aşamalarla ve ne zaman tamamlanması gerektiği planlanır. Bu süreç, genellikle Agile veya Waterfall gibi bir yazılım geliştirme metodolojisine göre organize edilir.
- **Milestones (Kilit Noktalar)**: Belirli tarihlere göre tamamlanması gereken önemli aşamalar (milestones) belirlenir. Örneğin, ilk versiyonun tamamlanma tarihi, birim testlerin yapılması için son tarih gibi kilometre taşları belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓**5. Risklerin ve Önlemlerin Belirlenmesi**

- **Risk Analizi**: Kodlama sırasında karşılaşılabilecek teknik zorluklar, kaynak yetersizlikleri, güvenlik sorunları gibi potansiyel riskler belirlenir. Her riskin etkisi değerlendirilir ve önlemler planlanır.
- **Yedekleme ve Sürüm Kontrolü**: Kodun güvenliğini sağlamak ve verimliliği artırmak için sürüm kontrol sistemleri (örneğin, Git) kullanılır. Kodlama planı yapılırken, kod yedeklemelerinin nasıl ve ne sıklıkla yapılacağı belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓**6. Test Planının Hazırlanması**

- **Test Stratejileri**: Kodlama sırasında veya sonrasında yapılacak olan testlerin türleri belirlenir (örneğin, birim testi, entegrasyon testi, kullanıcı kabul testi). Test stratejisi ve testin kapsamı belirlenir.
- **Otomatik Test Araçlarının Seçilmesi**: Eğer proje otomatik test gerektiriyorsa, bu testlerin hangi araçlarla yapılacağı ve otomatikleştirilmiş testlerin kodlama sırasında nasıl kullanılacağı belirlenir.

Kodlama - 1. Kodlama Planının Yapılması

✓**7. Dökümantasyon Planı**

- **Dökümantasyonun Kapsamı**: Kodlama sürecinde veya sonunda hazırlanacak teknik dökümantasyonun kapsamı belirlenir. Kodun nasıl çalıştığını, hangi işlevleri yerine getirdiğini açıklayan belgeler hazırlanır.
- **Dökümantasyon Standartları**: Projeye uygun dökümantasyon standartları ve formatları belirlenir. Örneğin, hangi kodların yorumlanacağı, API belgelerinin hangi formatta yazılacağı, kullanım talimatlarının nasıl hazırlanacağı gibi konulara karar verilir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

- ✓**Modül ve Sınıf Yapısının Oluşturulması** aşaması, yazılımın bileşenlerine ayrılarak daha yönetilebilir ve anlaşılabilir hale getirilmesini sağlar.

- ✓Bu adım, yazılımın işlevlerini mantıksal olarak gruplandırarak modüller ve sınıflar oluşturmak için kritik bir süreçtir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**1. Modüler Tasarımın Planlanması**

- **Fonksiyonelliklere Göre Modüllerin Ayrılması**: Yazılımın işlevlerine göre mantıksal bileşenlere (modüllere) ayrılması gereklidir. Örneğin, bir e-ticaret uygulamasında “ürün yönetimi,” “kullanıcı yönetimi” ve “sipariş yönetimi” gibi işlevsel modüller olabilir. Bu sayede her modül kendi sorumluluğuna odaklanır.
- **Bağımlılıkların Belirlenmesi**: Modüller arasındaki bağımlılıkları ve ilişkileri belirlemek, yazılımın sağlam bir şekilde yapılandırılması için önemlidir. Modüllerin birbirine bağımlılığını azaltmak, gelecekteki güncellemeleri ve değişiklikleri kolaylaştırır.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**2. Sınıfların Tanımlanması**

- **Temel Nesnelerin Belirlenmesi**: Sınıflar, nesne tabanlı programlama (OOP) paradigmasıyla tasarlanan yazılımlarda, sistemin temel nesnelerini temsil eder. Bu aşamada, yazılımın ana varlıkları (örneğin, "Kullanıcı," "Ürün," "Sipariş") belirlenir ve bu varlıklar için sınıflar oluşturulur.
- **Sınıf Sorumluluklarının Belirlenmesi (Single Responsibility Principle)**: Her sınıf, tek bir işlevi yerine getirmekten sorumlu olmalıdır. Sınıfların sorumlulukları belirlenirken, sınıfın hangi işlevleri gerçekleştireceği ve hangi verileri içereceği tanımlanır. Örneğin, bir "Kullanıcı" sınıfı kullanıcı bilgilerini ve kullanıcıya ait işlemleri içerir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**3. Veri Yapıları ve Özelliklerin Belirlenmesi**

- **Sınıf Özellikleri (Attributes)**: Her sınıfın saklayacağı veri, yani özellikler belirlenir. Örneğin, bir "Kullanıcı" sınıfı için ad, soyad, e-posta gibi özellikler tanımlanabilir. Özellikler, sınıfın hangi veriyi temsil edeceğini gösterir.
- **Veri Tiplerinin Belirlenmesi**: Her özellik için uygun veri tipleri belirlenir. Örneğin, "Kullanıcı" sınıfındaki "doğum tarihi" özelliği bir tarih tipi olabilirken, "kullanıcı adı" ise metin tipi olabilir. Veri tiplerinin doğru seçilmesi, veri tutarlılığı ve kodun verimli çalışması için önemlidir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**4. Sınıf Metotlarının (Fonksiyonların) Belirlenmesi**

- **İşlevlerin Tanımlanması**: Her sınıfın, kendi verilerini işlemek için gereken işlevleri (metotları) belirlenir. Metotlar, sınıfın hangi işlevleri gerçekleştireceğini ve hangi işlemleri yapacağını belirler. Örneğin, "Sipariş" sınıfında sipariş durumu güncelleme, ürün ekleme veya çıkarma gibi metotlar tanımlanabilir.
- **Erişim Belirleyicilerinin Kullanılması (Encapsulation)**: Sınıfın özelliklerinin ve metotlarının nasıl erişileceği, yani hangi erişim belirleyicilerinin (public, private, protected gibi) kullanılacağı belirlenir. Bu, sınıfın verilerini dış müdahalelere karşı koruma altına almak için önemlidir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**5. Sınıflar Arası İlişkilerin Tanımlanması**

- **İlişki Türlerinin Belirlenmesi**: Sınıflar arasındaki ilişkiler belirlenir. Nesne tabanlı programlamada, sınıflar arasındaki ilişkiler dört ana türde olabilir: **Birleşim (Aggregation)**, **Bileşim (Composition)**, **Kalıtım (Inheritance)** ve **Birliktelik (Association)**. Örneğin, bir "Sipariş" sınıfı "Kullanıcı" sınıfıyla bir birliktelik ilişkisi kurabilir.
- **Kalıtım İlişkilerinin Kurulması**: Sınıflar arasındaki kalıtım ilişkileri, ortak özelliklerin ve işlevlerin bir üst sınıfa taşınmasıyla kurulur. Kalıtım, kod tekrarını azaltır ve kodun daha modüler hale gelmesini sağlar. Örneğin, "Müşteri" ve "Çalışan" sınıfları, ortak özellikleri "Kişi" sınıfından miras alabilir.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**6. Arayüz (Interface) ve Soyut Sınıfların Oluşturulması**

- **Arayüzlerin (Interfaces) Tanımlanması**: Belirli işlevlerin farklı sınıflar tarafından uygulanmasını sağlamak için arayüzler tanımlanır. Arayüzler, sınıfların belirli bir işlevselliği yerine getirmesi gerektiğini belirtir ve böylece kodda esneklik sağlar.
- **Soyut Sınıfların Tanımlanması**: Eğer sınıfların bazı ortak işlevleri olacak ama her sınıfın bu işlevleri farklı bir şekilde uygulaması gerekiyorsa, soyut sınıflar oluşturulabilir. Soyut sınıflar, belirli işlevlerin miras alınan sınıflar tarafından uygulanmasını zorunlu kılar.

Kodlama - 2. Modül ve Sınıf Yapısının Oluşturulması

✓**7. Modüller ve Sınıflar İçin Yorum ve Dökümantasyon Yazılması**

- **Yorum Eklemek**: Her sınıfın ve modülün işlevini açıklayan yorumlar eklenir. Bu, kodun daha sonra bakımını yapacak kişilerin veya yeni geliştiricilerin kodu anlamasını kolaylaştırır.
- **Dökümantasyon Hazırlanması**: Modüllerin ve sınıfların hangi işlevleri gerçekleştirdiğini ve nasıl kullanıldığını anlatan teknik dökümantasyonlar hazırlanır. Özellikle karmaşık sınıflar için dökümantasyon, projeye yeni katılan ekip üyelerinin süreci anlamasını hızlandırır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

- ✓**Fonksiyonların Geliştirilmesi** aşaması, yazılımın gerçekleştirmesi gereken işlevlerin kodlandığı süreçtir.

- ✓Bu aşamada, her modül veya sınıf için planlanan işlevler ve bu işlevlerin çalışması gereken fonksiyonlar detaylandırılarak kodlanır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**1. Fonksiyon Gereksinimlerinin Belirlenmesi**

- **İşlevlerin Tanımlanması**: Her fonksiyonun ne yapması gerektiği belirlenir. Bu, fonksiyonun amacını, hangi işi gerçekleştirdiğini ve hangi problemi çözdüğünü netleştirir. Örneğin, bir "kullanıcı girişi" fonksiyonu, kullanıcının doğru kimlik bilgileriyle giriş yapıp yapmadığını kontrol eder.
- **Girdilerin ve Çıktıların Belirlenmesi**: Fonksiyonun hangi girdileri alacağı ve hangi çıktıları üreteceği tanımlanır. Bu, fonksiyonun çalışma mantığının anlaşılması için önemlidir. Örneğin, bir "fiyat hesaplama" fonksiyonu için ürün miktarı ve birim fiyat girdi olarak alınabilir, toplam fiyat ise çıktı olarak döndürülür.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**2. Fonksiyon İsimlendirme Standartlarına Uygun İsim Verilmesi**

- **Anlamlı ve Tutarlı İsimlendirme**: Fonksiyon isimlerinin ne iş yaptığını anlatacak şekilde seçilmesi önemlidir. Örneğin, “hesapla()” yerine, “toplamFiyatHesapla()” gibi daha açıklayıcı bir isim tercih edilmelidir.
- **Kodlama Standartlarına Uygunluk**: Projede kullanılan isimlendirme standartlarına (örneğin, camelCase veya snake_case) uygun olarak isimlendirme yapılır. Bu, kodun okunabilirliğini ve sürdürülebilirliğini artırır.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**3. Fonksiyonun İç Yapısının Tasarlanması**

- **Kontrol Akışının Belirlenmesi**: Fonksiyonun gerçekleştireceği işlemler sıralanır ve kontrol akışları (if-else, switch-case, döngüler gibi) tasarlanır. Örneğin, bir "sipariş durumu güncelle" fonksiyonu için kontrol akışı, sipariş durumuna göre değişebilir.
- **Koşullar ve Döngülerin Kullanımı**: Fonksiyonun işlemlerini gerçekleştirmek için gereken koşullar (if, else, switch) ve tekrarlayan işlemler için döngüler (for, while) tanımlanır. Kontrol yapıları, fonksiyonun iş akışını düzenli bir şekilde yönetir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**4. Fonksiyon İçerisinde Veri İşlemleri Yapma**

- **Veri Manipülasyonları**: Fonksiyon içerisinde, girdilerin işlenmesi, hesaplanması veya değiştirilmesi gibi veri işlemleri yapılır. Örneğin, bir "indirim hesapla" fonksiyonu, toplam fiyata indirim oranını uygulayarak yeni bir değer döndürebilir.
- **Yerel Değişkenler ve Parametrelerin Kullanımı**: Fonksiyon içinde kullanılan değişkenlerin ve parametrelerin doğru bir şekilde tanımlanması ve kullanılması önemlidir. Yerel değişkenler, fonksiyonun dışında kullanılmaz ve böylece kodun güvenliği ve okunabilirliği artar.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**5. Hata Yönetimi ve Hata Ayıklama (Debugging)**

- **Hata Kontrolleri (Error Handling)**: Fonksiyon içerisinde karşılaşılabilecek olası hatalar için hata yönetimi yapılır. Örneğin, bir "dosya aç" fonksiyonu dosya mevcut değilse veya erişilemiyorsa uygun bir hata mesajı ile kullanıcıyı bilgilendirebilir.
- **Hata Ayıklama (Debugging)**: Fonksiyon geliştirme sürecinde veya sonrasında hata ayıklama işlemi yapılır. Hata ayıklama, fonksiyonun beklenmedik durumlarda nasıl davrandığını görmek için önemlidir ve gerektiğinde sorunlu kod parçalarını düzeltmeyi sağlar.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**6. Fonksiyon Testlerinin Yazılması**

- **Birim Testleri (Unit Testing)**: Fonksiyonların tek başına doğru çalışıp çalışmadığını kontrol etmek için birim testleri yazılır. Her fonksiyon, beklenen sonuçları verip vermediği açısından test edilir. Örneğin, “toplamFiyatHesapla” fonksiyonu, belirli bir giriş için doğru toplam fiyatı verip vermediği açısından test edilebilir.
- **Sınır Durum Testleri**: Fonksiyonların olağan dışı durumlarda (örneğin, negatif veya sıfır değerler, çok büyük sayılar gibi) nasıl çalıştığını kontrol etmek için sınır durum testleri yapılır. Bu testler, fonksiyonun güvenilirliğini ve sağlamlığını ölçer.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**7. Optimizasyon ve Performans İyileştirmeleri**

- **Algoritma İyileştirmeleri**: Fonksiyonun verimli çalışması için kullanılan algoritmalar optimize edilir. Daha hızlı ve daha az kaynak tüketen algoritmalar tercih edilir. Örneğin, büyük bir veri kümesi üzerinde işlem yapacaksa, daha hızlı algoritmalar veya veri yapıları kullanılabilir.
- **Bellek Kullanımı ve Kaynak Yönetimi**: Fonksiyonun bellek kullanımı kontrol edilir ve mümkünse azaltılır. Gereksiz veriler bellekte tutulmaz ve belleği etkin kullanacak düzenlemeler yapılır. Büyük veri işlemlerinde belleğin sınırlı olduğu durumlarda bu oldukça kritiktir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**8. Kodun Anlaşılabilirliğini Artırmak için Yorumlar Eklemek**

- **Açıklayıcı Yorumlar**: Fonksiyonun içindeki önemli kod bölümlerine açıklayıcı yorumlar eklenir. Özellikle karmaşık işlemler gerçekleştiren fonksiyonlar için yorumlar, kodun ileride bakımını yapacak kişilerin kodu daha kolay anlamasını sağlar.
- **Dokümantasyon Standartlarına Uyum**: Yorumlar belirli bir dokümantasyon standardına (örneğin, Javadoc, docstrings gibi) uygun olarak yazılırsa, hem geliştirme sırasında hem de sonrasında daha kolay referans alınabilir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**9. Gözden Geçirme (Code Review)**

- **Kod İncelemesi (Code Review)**: Fonksiyon geliştirme işlemi tamamlandıktan sonra, kod bir veya birkaç ekip üyesi tarafından gözden geçirilir. Kod incelemesi, kodun okunabilirliği, hatalardan arındırılması ve kod standartlarına uygunluğunu sağlamak için yapılır.
- **İyileştirme Önerileri**: Gözden geçirme sırasında yapılan öneriler doğrultusunda fonksiyonlar yeniden düzenlenebilir ve geliştirilebilir.

Kodlama - 3. Fonksiyonların Geliştirilmesi

✓**10. Fonksiyonların Dökümantasyonu**

- **Fonksiyon Açıklamaları**: Fonksiyonun ne yaptığı, hangi parametreleri aldığı, ne tür bir çıktı verdiği gibi bilgiler dokümante edilir. Bu, hem ekip üyeleri için hem de yazılımın gelecekteki sürümleri için referans kaynağı sağlar.
- **Girdi ve Çıktı Açıklamaları**: Her parametrenin neyi ifade ettiği, hangi değerleri kabul edebileceği ve fonksiyonun döndüreceği sonuçlar açıklanır. Bu bilgiler, yazılımın sürdürülebilirliği için önemlidir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

- ✓**Veritabanı ve Diğer Harici Bağlantıların Kodlanması** aşaması, yazılımın ihtiyaç duyduğu veri depolama, veri erişim ve harici sistemlerle bağlantı kurma işlemlerinin kodlandığı adımdır.

- ✓Bu adımda, yazılımın veritabanına bağlanması, harici API'ler ve servislerle etkileşime geçmesi sağlanır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**1. Veritabanı Tasarımının ve Yapısının Belirlenmesi**

- **Veritabanı Seçimi**: Projenin ihtiyaçlarına göre uygun bir veritabanı türü (örneğin, ilişkisel veritabanı için MySQL, PostgreSQL veya NoSQL veritabanı için MongoDB, Redis) seçilir. Veritabanı türü, yazılımın veri yapısına, ölçeklenebilirlik ihtiyaçlarına ve performans gereksinimlerine göre belirlenir.
- **Tablo Yapısının Tasarlanması**: İlişkisel veritabanı kullanılıyorsa, gerekli tablolar, sütunlar ve veri türleri tasarlanır. Örneğin, bir e-ticaret uygulamasında “Kullanıcı,” “Ürün,” ve “Sipariş” gibi tablolar oluşturulabilir.
- **İlişkiler ve Yabancı Anahtarlar**: Tablolar arasında ilişkiler (birçoktan-bire, birden-çoğa) belirlenir ve yabancı anahtarlar tanımlanır. Bu, verilerin ilişkili bir şekilde düzenlenmesini sağlar. Örneğin, “Sipariş” tablosu ile “Kullanıcı” tablosu arasında bir ilişki kurulabilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**2. Veritabanı Bağlantısının Kurulması**

- **Bağlantı Yapılandırması**: Yazılımın veritabanına bağlanabilmesi için veritabanı bağlantı bilgileri (host, port, kullanıcı adı, şifre gibi) yapılandırılır. Bu bağlantı bilgileri genellikle yapılandırma dosyalarında veya çevre değişkenlerinde tutulur.
- **Bağlantı Kütüphanesi Seçimi**: Veritabanına bağlantıyı yönetmek için uygun kütüphane veya çerçeve (örneğin, Python için SQLAlchemy, Java için JDBC) seçilir ve entegre edilir. Bu kütüphaneler, veritabanı bağlantılarını ve işlemlerini daha güvenli ve düzenli bir şekilde yönetir.
- **Bağlantı Havuzu (Connection Pooling)**: Uygulamanın performansını artırmak için bağlantı havuzu kullanılması düşünülür. Bağlantı havuzu, aynı anda birden çok veritabanı bağlantısını yöneten ve her işlem için yeni bir bağlantı açma maliyetini düşüren bir tekniktir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓ **3. CRUD İşlemlerinin Kodlanması (Create, Read, Update, Delete)**

- **Veri Ekleme (Create)**: Yeni verilerin eklenmesi için gerekli SQL sorguları veya ORM (Nesne İlişkisel Eşleme) işlemleri kodlanır. Örneğin, yeni bir kullanıcı eklemek için bir INSERT sorgusu veya ORM kullanılıyorsa add_user() gibi bir fonksiyon oluşturulur.
- **Veri Okuma (Read)**: Verilerin okunması için gerekli sorgular veya ORM işlemleri yazılır. Bu adım, veri tabanından spesifik verilerin çekilmesi veya listelenmesi için kullanılır. Örneğin, tüm kullanıcıları listeleyen veya belirli bir ürün bilgisini getiren sorgular.
- **Veri Güncelleme (Update)**: Mevcut verilerin güncellenmesi için güncelleme sorguları veya işlemleri oluşturulur. Örneğin, kullanıcının e-posta adresini güncellemek için bir UPDATE sorgusu yazılabilir.
- **Veri Silme (Delete)**: Veritabanındaki verilerin silinmesi işlemlerini gerçekleştiren kodlar yazılır. Örneğin, eski sipariş kayıtlarını silmek için DELETE sorguları veya ORM metodları kullanılır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**4. Veri Güvenliğinin Sağlanması**

- **SQL Injection Önleme**: SQL enjeksiyonuna karşı güvenlik önlemleri alınır. Parametreli sorgular kullanmak ve doğrudan kullanıcı girdilerini SQL sorgularına eklememek gibi teknikler, veritabanı güvenliğini sağlar.
- **Veri Şifreleme ve Anonimleştirme**: Özellikle hassas bilgiler (örneğin, şifreler, kredi kartı bilgileri) için veri şifreleme veya maskeleme uygulanır. Bu bilgiler, şifrelenmiş veya anonimleştirilmiş halde saklanarak güvenlik sağlanır.
- **Yetkilendirme ve Erişim Kontrolleri**: Veritabanına erişim, belirli kullanıcı veya kullanıcı grupları için kısıtlanabilir. Verilere yalnızca belirli kullanıcılar erişebilmeli ve yalnızca belirli işlemleri gerçekleştirebilmelidir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**5. Veri Aktarımı ve Dönüştürme (ETL)**

- **Veri Alma (Extract)**: Eğer farklı kaynaklardan veri alınacaksa, bu verilerin nasıl alınacağı kodlanır. Harici bir API veya dosya sisteminden veri çekilecekse gerekli bağlantılar yapılır.
- **Veri Dönüştürme (Transform)**: Harici kaynaklardan gelen verilerin uygun formata dönüştürülmesi sağlanır. Bu, verilerin yazılımda kullanılabilir hale gelmesi için gerekebilir.
- **Veri Yükleme (Load)**: Dönüştürülen verilerin veritabanına veya hedef veri deposuna yüklenmesi işlemleri yapılır. Bu adımda, ETL işlemleri sırasında veri doğruluğunu sağlamak için çeşitli kontroller yapılır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**6. Harici API ve Servislere Bağlantı Kodlarının Yazılması**

- **API Erişim Katmanının Oluşturulması**: Harici bir API'ye bağlanmak için gereken kod yapısı ve fonksiyonlar yazılır. API anahtarı, URL, parametreler gibi bağlantı bilgileri tanımlanır. Örneğin, bir hava durumu API'sine bağlanmak için gerekli bağlantı ve istek bilgileri düzenlenir.
- **API Çağrılarının Yönetilmesi**: API çağrıları belirli aralıklarla yapılacaksa veya belirli durumlarda tetiklenecekse, bu iş akışları kodlanır. Örneğin, kullanıcı oturum açtığında bir kimlik doğrulama API'sine istek gönderilebilir.
- **Gelen Verilerin İşlenmesi**: Harici API veya servislerden dönen veri, gerekli işlemler için işlenir ve uygun formata dönüştürülür. Dönen JSON, XML gibi veri formatları uygun veri yapısına çevrilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**7. Veritabanı ve Harici Bağlantıların Hata Yönetimi**

- **Bağlantı Hatalarının Yönetilmesi**: Veritabanı veya API bağlantısı başarısız olduğunda hata yönetimi yapılır. Bağlantı kesildiğinde veya sunucuya erişim sağlanamadığında, kullanıcıya bilgilendirme mesajları gösterilir veya otomatik yeniden bağlanma işlemleri uygulanır.
- **Hata Günlüğü Tutma**: Veritabanı veya harici bağlantılarda karşılaşılan hataların kaydedilmesi için hata günlüğü (log) tutulur. Böylece hatalar daha sonra analiz edilebilir ve gerektiğinde çözüm üretilebilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**8. Performans Optimizasyonu ve Önbellekleme (Caching)**

- **Veritabanı Sorgularının Optimizasyonu**: Sorguların hızlı çalışması için veritabanı indeksleri, optimize edilmiş sorgular ve SQL analizleri yapılır. Gereksiz veri getirmekten kaçınılır ve büyük veri kümesi içeren sorgularda optimizasyon uygulanır.
- **Önbellekleme (Caching)**: Sık kullanılan veriler için önbellekleme yapılır, böylece her sorguda veritabanına başvurmaya gerek kalmaz. Redis gibi önbellek çözümleri, yazılımın performansını artırır ve veritabanı üzerindeki yükü azaltır.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**9. Bağlantı Testlerinin Yapılması**

- **Veritabanı Bağlantı Testleri**: Veritabanına yapılan bağlantıların doğruluğunu ve sürekliliğini test etmek için bağlantı testleri yapılır. Bu testler, bağlantının stabil ve hatasız çalıştığını garanti eder.
- **API ve Servis Testleri**: Harici API ve servislerin bağlantı testleri yapılır. API'den dönen verilerin beklenen formatta olup olmadığı kontrol edilir ve belirli durumlarda nasıl yanıt verdikleri test edilir.

Kodlama - 4. Veritabanı ve Diğer Harici Bağlantıların Kodlanması

✓**10. Dokümantasyon ve Yedekleme**

- **Veritabanı ve API Dokümantasyonu**: Veritabanı tablolarının yapısı, sütun açıklamaları ve kullanılan API'lerin dokümantasyonu hazırlanır. Bu bilgiler, geliştiricilerin ve bakım ekiplerinin veri yapısını anlamalarına yardımcı olur.
- **Yedekleme ve Kurtarma Prosedürleri**: Veritabanının yedeklenmesi ve acil durumlarda kurtarılabilmesi için yedekleme prosedürleri oluşturulur. Bu adım, verilerin güvenliği ve bütünlüğü açısından önemlidir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**1. Kodun Test Edilmesi**

✓Kodlama tamamlandıktan sonra, kodun beklendiği gibi çalıştığını doğrulamak için test işlemleri gerçekleştirilir. Bu aşamada, hem manuel hem de otomatik test yöntemleri kullanılabilir.

✓**a) Test Türleri**

- **Birim Testleri (Unit Tests):**
  - Yazılımın her bir bileşeninin (örneğin bir fonksiyonun) bağımsız olarak test edilmesidir.
  - Amaç, her birim kodun doğru çalıştığından emin olmaktır.
  - Örnek araçlar: JUnit (Java), pytest (Python).
- **Entegrasyon Testleri (Integration Tests):**
  - Farklı bileşenlerin birlikte çalışabilirliğini kontrol eder.
  - Veri akışı, API çağrıları gibi modüller arası iletişim test edilir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

- **Sistem Testleri (System Tests):**
  - Yazılımın tamamının test edilmesi.
  - Kullanıcı senaryoları üzerinden yazılımın bütünsel olarak doğru çalışıp çalışmadığı kontrol edilir.
- **Kabul Testleri (Acceptance Tests):**
  - Müşteri veya son kullanıcı tarafından yazılımın gereksinimlere uygunluğu test edilir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**b) Test Güdümlü Geliştirme (Test-Driven Development - TDD)**

- Önce testlerin yazıldığı, ardından bu testleri geçecek kodun geliştirildiği bir yöntemdir.
- Bu yöntem, yazılım hatalarını en aza indirgemek için güçlü bir yaklaşımdır.

✓**c) Test Otomasyonu**

- Test süreçlerinin otomatikleştirilmesi, tekrarlayan iş yükünü azaltır ve hızlı geri bildirim sağlar.
- Örnek araçlar: Selenium, Appium, Jenkins.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**2. Hata Ayıklama (Debugging)**

✓Hata ayıklama, test sırasında veya geliştirme sürecinde ortaya çıkan hataların nedenlerini bulma ve düzeltme sürecidir. Bu süreç, yazılımın hatasız ve doğru çalışmasını sağlamak için kritik öneme sahiptir.

✓**a) Debugging Süreci**

1.Hatayı Belirleme:

1. Testler sırasında ya da kullanıcıdan gelen geri bildirimler doğrultusunda hatalar tespit edilir.
2. Örnek: Hatalı bir çıktı, çökme, beklenmeyen davranış.

2.Hatanın İzolasyonu:

1. Hatanın kaynağını bulmak için kod parça parça incelenir.
2. Bunun için kod içindeki belirli noktalara loglama veya breakpoint'ler eklenir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**2. Hata Ayıklama (Debugging)**

3. Hatanın Analizi:

1. Hatanın temel sebebi analiz edilir. Bu, genellikle yanlış algoritma kullanımı, eksik gereksinimlerin kodlanması ya da veri türü uyumsuzluğu gibi nedenlere dayanabilir.

4. Hatanın Düzeltilmesi:

1. Analiz sonucunda kodda gerekli değişiklikler yapılır.

5. Düzeltmenin Test Edilmesi:

1. Yapılan değişikliğin sorunu çözüp çözmediği test edilir.
2. Yeni hatalar yaratmadığından emin olmak için regresyon testi yapılabilir.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

b) Debugging Yöntemleri

- **Manuel Debugging:**
  - Kodun adım adım çalıştırılarak hata kaynaklarının manuel olarak bulunması.
  - Örnek: Basit print komutları veya loglama teknikleri.
- **Araçlarla Debugging:**
  - IDE'lerin sunduğu hata ayıklama araçları kullanılır.
  - Breakpoint'ler ile kod belirli noktalarda durdurularak değişken değerleri incelenir.
  - Örnek araçlar: Visual Studio Debugger, PyCharm Debugger.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓ **c) Yaygın Hata Türleri**

- **Sözdizimi Hataları (Syntax Errors):**
  - Dilin kurallarına aykırı yazım hataları. Örnek: Unutulan noktalı virgül, yanlış parantez kullanımı.
- **Mantıksal Hatalar (Logical Errors):**
  - Kod doğru çalışsa bile beklenen sonucu vermemesi. Örnek: Yanlış bir koşul yazımı.
- **Çalışma Zamanı Hataları (Runtime Errors):**
  - Kod çalışırken oluşan hatalar. Örnek: Sıfıra bölme hatası, null referans hatası.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**d) Hata Ayıklama Teknikleri**

- **Binary Search Debugging:**
  - Kodun farklı bölümlerini kapatarak veya test ederek hatanın bulunduğu yeri daraltma.
- **Log Analizi:**
  - Kodun çalışması sırasında yazdırılan log mesajlarını inceleyerek hatanın nedenini tespit etme.
- **Stack Trace İncelemesi:**
  - Hata mesajları sırasında verilen "stack trace" bilgilerini kullanarak hangi fonksiyonların sorun yarattığını bulma.

Kodlama - 5. Kodun Test Edilmesi ve Hata Ayıklama (Debugging)

✓**3. Test ve Debugging Araçları**

- **Test Araçları:** JUnit, pytest, Selenium.
- **Debugging Araçları:** Visual Studio, Eclipse Debugger, Chrome Developer Tools.

Kodlama - 6. Optimizasyon

- **Optimizasyon İşlemi**, kodlama adımında yazılımın performansını, verimliliğini ve kaynak kullanımını iyileştirmek için yapılan işlemleri ifade eder.

- Bu süreç, kodun daha hızlı çalışmasını, daha az kaynak tüketmesini ve daha az hata üretmesini sağlar. Optimizasyon, yazılımın hem kullanıcı deneyimini artırır hem de maliyetleri düşürür.

Kodlama - 6. Optimizasyon

✓ **1. Optimizasyon İşleminin Amacı**

- Yazılımın hızını artırmak.
- Daha az bellek, işlemci ve ağ kaynağı kullanmak.
- Kullanıcıların daha sorunsuz bir deneyim yaşamasını sağlamak.
- Teknik borcu (technical debt) azaltarak sürdürülebilirliği artırmak.
- Yazılımın büyük veri veya kullanıcı yüklerinde de çalışabilirliğini sağlamak (ölçeklenebilirlik).

Kodlama - 6. Optimizasyon

2. Optimizasyon Çeşitleri

a) Kod Optimizasyonu

•Kodun karmaşıklığını azaltarak daha verimli çalışmasını sağlamak.

•Gereksiz veya yinelenen kod parçalarını kaldırmak.

•Daha iyi algoritmalar veya veri yapıları kullanmak.

•Örnekler:

- Zaman karmaşıklığını O(n2)O(n^2)O(n2)'den O(nlogn)O(n \log n)O(nlogn)'e düşürmek.
- Tekrar eden işlemleri önlemek için önbellekleme (caching) kullanımı.

b) Veri Tabanı Optimizasyonu

•Veri tabanı sorgularının daha hızlı çalışmasını sağlamak.

•Gereksiz sorguları ve büyük veri taşımasını önlemek.

•Örnekler:

- İndeksleme ile sorgu hızını artırmak.
- Gereksiz JOIN işlemlerini azaltmak.
- Normalizasyon ve gerektiğinde denormalizasyon kullanmak.

Kodlama - 6. Optimizasyon

c) Bellek Optimizasyonu

- Yazılımın daha az bellek tüketmesi sağlanır.
- Bellek sızıntılarının (memory leak) önüne geçilir.
- Örnekler:
  - Gereksiz değişkenleri temizlemek.
  - Daha hafif veri tiplerini kullanmak (ör. float yerine int).
  - Gerekmediğinde nesneleri heap yerine stack'te tutmak.

d) İletişim ve Ağ Optimizasyonu

- Sunucu ve istemci arasındaki veri trafiğini minimize etmek.
- Daha hızlı veri aktarımı sağlamak.
- Örnekler:
  - Daha küçük ve sıkıştırılmış veri paketleri kullanmak (ör. gzip).
  - Gereksiz API çağrılarını azaltmak.
  - CDN (Content Delivery Network) kullanımı.

Kodlama - 6. Optimizasyon

e) Algoritma ve İşlem Optimizasyonu

•Daha etkili algoritmalar ve yöntemler kullanarak yazılımın verimliliğini artırmak.

•Örnekler:

- Büyük veri setleriyle çalışırken paralelleştirme (multi-threading).
- Zaman alan işlemler için asenkron programlama (asynchronous programming).

f) Kullanıcı Arayüzü (UI/UX) Optimizasyonu

•Arayüz elemanlarının daha hızlı yüklenmesini sağlamak.

•Daha düşük gecikme süreleriyle kullanıcılara daha iyi bir deneyim sunmak.

•Örnekler:

- Görselleri optimize etmek (ör. PNG yerine WebP).
- Yavaş yükleme (lazy loading) tekniklerini uygulamak.

Kodlama - 6. Optimizasyon

3. Optimizasyon İşlemi Adımları

a) Mevcut Performansı Değerlendirme

•Performans ölçüm araçları kullanarak yazılımın darboğazlarını belirlemek.

•Örnek araçlar:

- Kod için: **SonarQube**, **Linting araçları**.
- Performans için: **New Relic**, **AppDynamics**.
- Veri tabanı için: **SQL Profiler**, **EXPLAIN** komutları.

b) Sorunları Tespit Etme

•Performansı etkileyen kritik noktaları belirleme.

•Örnekler:

- Yavaş çalışan SQL sorguları.
- İhtiyaç duyulmayan API çağrıları.
- Karmaşık ve büyük algoritmalar.

Kodlama - 6. Optimizasyon

c) Çözüm Önerileri Geliştirme

•Alternatif yöntemler, algoritmalar veya araçlar belirlenir.

•Maliyet ve etkileri değerlendirilerek en uygun çözüm seçilir.

d) Kodun Düzenlenmesi

•Belirlenen çözümler uygulanır ve kodda gerekli değişiklikler yapılır.

•Kodun okunabilirliği ve sürdürülebilirliği korunur.

e) Test ve Doğrulama

•Optimizasyon sonrası yazılımın tüm işlevlerinin doğru çalıştığından emin olunur.

•Performans ve stres testleri yapılır.

f) Sonuçların İzlenmesi

•Optimizasyonun yazılım üzerindeki etkileri izlenir.

•Gerektiğinde ek iyileştirmeler yapılır.

Kodlama - 6. Optimizasyon

4. Optimizasyon Sürecinde Kullanılan Araçlar

Araç/Platform	Kullanım Amacı
SonarQube	Kod kalitesini ve güvenliğini analiz etme
JProfiler, YourKit	Bellek ve CPU performansı izleme
SQL Profiler	Veri tabanı sorgularını analiz etme
JMeter, Locust	Performans ve yük testi
Google Lighthouse	Web uygulamaları için hız ve kalite ölçümü

Kodlama - 6. Optimizasyon

5. Optimizasyon İşleminin Zorlukları

- **Kodun Karmaşıklığı:** Karmaşık kodların optimize edilmesi zordur ve yanlış optimizasyon yeni hatalara neden olabilir.
- **Zaman ve Maliyet:** Optimizasyon süreci ek zaman ve maliyet gerektirebilir.
- **Performans-Tasarruf Dengesi:** Performansı artırırken kodun okunabilirliği ve bakımı zorlaşabilir.
- **Ölçüm Hataları:** Optimizasyon öncesi ve sonrası etkileri doğru ölçümlemek zor olabilir.

Kodlama - 6. Optimizasyon

6. Optimizasyon İşlemi İçin En İyi Uygulamalar

1.Önceliklendirme: En kritik sorunlara odaklanın ve yazılımın kullanıcı deneyimini en çok etkileyen darboğazları çözün.

2.İzleme ve Analiz: Performans sorunlarını düzenli olarak analiz edin ve optimize edin.

3.Modüler Yaklaşım: Kodun optimize edilmesi gerektiğinde, modüler bir yaklaşım benimseyerek işlemi daha kolay hale getirin.

4.Test Otomasyonu: Optimizasyon sonrası yazılımın doğru çalıştığından emin olmak için test süreçlerini otomatikleştirin.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Kodun Gözden Geçirilmesi (Code Review), yazılım geliştirme sürecinde, bir kişinin veya bir ekip üyesinin yazdığı kodun başka bir geliştirici ya da ekip tarafından incelenmesi işlemidir.

Bu süreç, kodun kalitesini artırmayı, hataları en aza indirmeyi ve kodun standartlara uygunluğunu sağlamayı hedefler.

Kodun gözden geçirilmesi, yazılım projelerinin sürdürülebilirliği ve ekip içi iş birliği açısından önemli bir adımdır.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

1. Kodun Gözden Geçirilmesinin Amaçları

- **Hata Tespiti:** Kodun hatalara karşı kontrol edilmesi ve hataların erkenden tespit edilmesi.
- **Kod Kalitesini Artırma:** Kodun okunabilir, sürdürülebilir ve modüler olmasını sağlama.
- **Standartlara Uygunluk:** Kodun belirlenen kodlama standartlarına ve en iyi uygulamalara uygunluğunu kontrol etme.
- **Ekip Öğrenimi ve İş Birliği:** Geliştiriciler arasında bilgi paylaşımını teşvik etme ve daha iyi çözümler geliştirme.
- **Teknik Borcun Azaltılması:** Gelecekteki bakım sürecini kolaylaştırmak için kodun düzenli olmasını sağlama.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

2. Kod Gözden Geçirme Türleri

a) Resmi Kod Gözden Geçirme (Formal Code Review)

•Detaylı ve yapılandırılmış bir inceleme sürecidir.

•Belirli bir prosedür izlenir; örneğin, kod belgeleri hazırlanır ve ekip bir toplantıda kodu tartışır.

•Daha çok büyük projelerde ve kritik sistemlerde uygulanır.

b) Gayri Resmi Kod Gözden Geçirme (Informal Code Review)

•Geliştiriciler arasında genellikle kısa bir toplantı veya birebir yapılan incelemelerdir.

•Daha hızlı ve esnek bir süreçtir.

•Örnek: Bir geliştiricinin kodunu yazdıktan sonra başka bir geliştiriciye göstermesi.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

c) Çift Programlama (Pair Programming)

- İki geliştiricinin aynı anda aynı kod üzerinde çalıştığı süreçtir.
- Kod yazılırken eş zamanlı olarak incelenir.
- Özellikle Agile metodolojilerinde sıkça kullanılır.

d) Araç Destekli Gözden Geçirme (Tool-Assisted Code Review)

- Kod gözden geçirme araçları kullanılarak yapılan inceleme.
- Örnek araçlar: GitHub Pull Requests, Bitbucket, GitLab, Crucible.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

3. Kod Gözden Geçirme Süreci

Adım 1: Kodun Hazırlanması

•Kodun düzgün bir şekilde formatlanması ve gereksiz dosyaların temizlenmesi.

•Kodun belgelendirilmesi; bu, gözden geçiren kişinin süreci anlamasını kolaylaştırır.

•Kodun çalışma şeklinin kısa bir özeti hazırlanır.

Adım 2: Gözden Geçirilecek Kodun Seçimi

•Küçük ve odaklanmış bir kod parçacığı seçilir (örneğin, bir fonksiyon veya modül).

•Büyük kod parçalarının gözden geçirilmesi daha zor ve zaman alıcıdır.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Adım 3: Gözden Geçiricinin Belirlenmesi

•Kodun, proje bağlamını iyi bilen bir ekip üyesi veya teknik lider tarafından incelenmesi tercih edilir.

•Bazı durumlarda, ekip dışından bağımsız bir uzman da bu sürece dahil edilebilir.

Adım 4: Kodun İncelenmesi

•Gözden geçiren kişi aşağıdaki unsurları kontrol eder:

- **Kod Kalitesi:** Kod okunabilir mi, anlaşılırlığı yüksek mi?
- **Hata ve Sorunlar:** Mantıksal hatalar veya eksiklikler var mı?
- **Performans:** Kod verimli çalışıyor mu, daha iyi bir çözüm uygulanabilir mi?
- **Güvenlik:** Güvenlik açıkları veya tehditler içeriyor mu?
- **Uyumluluk:** Kod, ekip standartlarına veya projedeki diğer kodlara uyumlu mu?

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

Adım 5: Geri Bildirim Verme

•Gözden geçirme sonuçları geliştiriciye geri bildirilir.

•Geri bildirimler yapıcı ve açıklayıcı olmalıdır.

•Örnek bir geri bildirim:

- **İyi:** "Burada bu algoritma doğru çalışıyor, ancak performans açısından şu alternatif algoritmayı düşünebilirsin."
- **Kötü:** "Bu kod yanlış, baştan yap."

Adım 6: Revizyon

•Geliştirici, gözden geçirme sırasında belirtilen sorunları veya eksiklikleri düzeltir.

•Revize edilen kod tekrar gözden geçirilir.

Adım 7: Onay ve Birleştirme

•Kod, gerekli düzenlemeler yapıldıktan sonra onaylanır ve kod deposuna (repository) birleştirilir.

Kodlama - 7. Kodun Gözden Geçirilmesi (Code Review)

4. Kod Gözden Geçirme Sürecinde Kullanılan Araçlar

Araç/Platform	Amacı
GitHub Pull Request	Kod değişikliklerini inceleme ve yorumlama
GitLab Merge Request	Kod birleşme işlemlerini yönetme
Crucible	Resmi kod inceleme süreci sağlama
Phabricator	Ekip bazlı gözden geçirme ve yönetim
SonarQube	Kod kalitesi ve güvenliği analiz etme

Kodlama - 8. Dökümantasyon

Kodlama Adımında Dökümantasyon İşlemi

Dökümantasyon, yazılım geliştirme sürecinde, yazılımın işleyişi, yapısı, kullanım kılavuzları ve teknik detaylarının yazılı olarak kaydedilmesi işlemidir.

Kodlama aşamasında yapılan dökümantasyon, kodun anlaşılabilirliğini artırır, ekip içi iletişimi kolaylaştırır ve yazılımın bakımını ve geliştirilmesini daha sürdürülebilir hale getirir.

Kodlama - 8. Dökümantasyon

1. Dökümantasyonun Amacı

- **Kodun Anlaşılabilirliğini Artırmak:** Kodun işlevlerini ve yapısını kolayca anlamak için açıklamalar sağlamak.
- **Ekip İşbirliğini Kolaylaştırmak:** Yeni ekip üyelerinin projeye hızlı bir şekilde uyum sağlamasına yardımcı olmak.
- **Bakımı Kolaylaştırmak:** Yazılımın ileride güncellenmesi, genişletilmesi veya düzeltilmesi gerektiğinde süreçleri hızlandırmak.
- **Kullanıcı Rehberi Sağlamak:** Son kullanıcılar için yazılımın nasıl kullanılacağını açıklamak.
- **Yasal ve Uyumluluk Gereksinimlerini Karşılamak:** Belirli sektörlerde yasal düzenlemelere uygunluğu sağlamak için teknik belgeler oluşturmak.

Kodlama - 8. Dökümantasyon

2. Dökümantasyon Türleri

a) Kod Düzeyinde Dökümantasyon

•Kod içerisine eklenen açıklamalar ve yorumlardır.

•Örnekler:

- Fonksiyon açıklamaları (ne işe yarar, hangi parametreleri alır, ne döndürür).
- Karmaşık algoritmaların açıklamaları.
- Kullanılan kütüphaneler ve modüller hakkında bilgi.

b) Teknik Dökümantasyon

•Yazılımın teknik yapısını ve mimarisini açıklar.

•İçerik:

- Sistem mimarisi (modüller, veri akışları, sistem bileşenleri).
- Kullanılan teknolojiler.
- API dokümantasyonu (endpointler, veri formatları, yanıtlar).
- Veri tabanı şemaları.

Kodlama - 8. Dökümantasyon

c) Kullanıcı Dökümantasyonu

•Son kullanıcıların yazılımı doğru şekilde kullanabilmesi için hazırlanır.

•İçerik:

- Kurulum kılavuzları.
- Kullanım talimatları.
- Yaygın hatalar ve çözümleri (SSS bölümü).

d) Proje Yönetimi Dökümantasyonu

•Yazılım geliştirme sürecindeki işlerin planlanması ve takibi için oluşturulur.

•İçerik:

- İş gereksinimleri ve kapsam belgeleri.
- Proje zaman çizelgesi.
- Sprint notları (Agile projelerde).
- Test planları ve sonuçları.

Kodlama - 8. Dökümantasyon

e) Bakım ve Destek Dökümantasyonu

•Yazılımın bakımını üstlenecek kişiler için teknik detaylar ve yönergeler sağlar.

•İçerik:

- Güncellenmesi gereken modüller.
- Log dosyalarının analizi.
- Performans sorunlarını çözme adımları.

Kodlama - 8. Dökümantasyon

3. Dökümantasyon Süreci

Adım 1: Dökümantasyon Planlama

•Projenin kapsamına uygun dökümantasyon türleri belirlenir.

•Hangi dökümantasyon türlerinin kimler tarafından oluşturulacağı ve ne zaman tamamlanacağı planlanır.

Adım 2: İçerik Hazırlama

•**Hedef Kitle:** Dökümantasyonun, okuyucunun bilgi düzeyine uygun bir dil ve ayrıntı seviyesi kullanılarak hazırlanması gerekir.

- Örneğin, teknik belgeler mühendisler için yazılırken kullanıcı kılavuzları daha sade bir dilde yazılır.

•Kod ile ilgili temel bilgilerin yanında, yazılımın genel amacı ve işlevselliği hakkında bilgiler eklenir.

Kodlama - 8. Dökümantasyon

Adım 3: Görselleştirme

•Karmaşık süreçlerin daha kolay anlaşılması için görsel araçlar kullanılabilir:

- Diyagramlar (örneğin, UML diyagramları).
- Akış şemaları.
- Örnek ekran görüntüleri.

Adım 4: Revizyon ve Güncelleme

•Dökümantasyon, düzenli olarak gözden geçirilir ve yazılımın güncellenmesi durumunda ilgili bölümler değiştirilir.

•Dökümantasyonun eski veya hatalı bilgiler içermediğinden emin olunur.

Adım 5: Paylaşım

•Dökümantasyon ekip üyeleri, son kullanıcılar veya ilgili taraflarla paylaşılır.

•Paylaşım platformları: Confluence, Notion, Google Docs, GitHub Wiki.

Kodlama - 8. Dökümantasyon

4. Dökümantasyon Sürecinde Kullanılan Araçlar

Araç/Platform	Amacı
Markdown	Basit ve okunabilir teknik dokümanlar yazma.
Doxygen	Koddan otomatik dökümantasyon oluşturma.
Swagger	API dökümantasyonlarını oluşturma.
Confluence, Notion	Proje belgelerini merkezi bir yerde tutma.
PlantUML, Lucidchart	Diyagramlar ve akış şemaları oluşturma.