9. Hafta
1. Grup
Başlıyoruz

11.2.4. Yürütme Ortamı

Yazılımlar ya tek bir bilgisayar üzerinde ya da birbirleriyle iletişim halinde olan bilgisayarlar üzerinde koşarlar. Tek bilgisayardaki yürütme ortamı genellikle işletim sistemidir. Bazen, bunun üzerinde bir başka katman daha konarak daha etkin çoklu programlama ortamı yaratılır (Java Virtual Machine bunlardan biridir). Dağıtık bilgi işlemeyi destekleyen çeşitli bilgisayar yazılım mimarileri vardır. Özellikle bir yerel alan ağı üzerindeki uygulama yazılımlarının bir arada ve uyum içinde çalışmasını sağlayan, onları alttaki bilgisayar ve ağ yapısından ayıran ara katman yazılımlarına daha önce değinmiştik. Aynı zamanda tek bilgisayarlı sistemlerde de kullanılabilen bu tür yazılımlar için tanımlanmış standartlardan bazılarını kısaca tanıyalım:

11.2.4.1. CORBA

CORBA (Common Object Request Broker Architecture) uluslararası boyutta yazılım ve donanım sağlayıcılarından oluşan *Object Management Group* (OMG) adlı ticari olmayan bir örgüt tarafından ortaya konan endüstriyel bir standarttır. Ticari firmalar, çeşitli isteklere göre belirlenip yayınlanmakta olan bu mimari standardına uygun olarak kendi yazılım ürünlerini geliştirmekte ve piyasaya sürmektedirler. Bu şekilde, yerel bilgisayar ağı mimarisinden ve donanımdan bağımsız olarak, dağıtık, nesneye yönelik programlama yapabilme olanağı getirilmektedir.

Uygulama yazılımları birer CORBA nesnesi olarak tanımlanır ve standart bir dil olan *Arayüz Tanımlama Dili* (Interface Definition Language-IDL) ile arayüzleri hazırlanır. IDL arayüz dosyası, bir nesnenin kullanılabilmesi için sunucunun sağladığı veri çeşitlerini, yöntemleri ve işlemleri açıklar. Bu arayüzler daha sonra C++, JAVA, ADA gibi çeşitli yüksek düzey programlama dillerine çevrilerek kullanılırlar. Bir CORBA nesnesi, diğer nesnelerin yerlerini, geliştirilme dil ve şekillerini bilmeden onların arayüzlerinde tanımlanmış işlemlerini kullanırlar. Her nesne, mimarinin kalbi olan *Nesne İsteme Aracısı* (Object Request Broker - ORB) ile haberleşmektedir. ORB çeşitli üretici firmalar tarafından geliştirilmiş olabilir, ancak nesneler standart bir arayüz kullandıklarından ORB değişikliği uygulama yazılımını etkilememektedir. ORB'un sağladığı çeşitli temel işlevler yanında OMG tarafından yararlı görülen Nesne Servisleri de standart hale getirilmiştir. Bu servislerden bazıları Tablo-7.1 de listelenmiştir.

Internet Inter-ORB Protocol (IIOP) CORBA İnternet üzerindeki geniş dağıtık uygulamalara erişimi sağlayan bir açık protokoldur. Bu protokol TCP/IP üzerine kurulur ve ORB lar arası iletişimi sağlar. IIOP farklı makineler üzerindeki nesnelerin haberleşmelerini sağlayarak bilgisayar ağı kavramını büyük boyutlara kadar genişletebilir. Halen Mart 2004 tarihli CORBA 3.0 Belirtimi yürürlükte olup bu konudaki son sürümler ve ilgili diğer belirtimler (IDL, CCM) www.omg.org adresinde bulunmaktadır.

11.2.4.2. DDS

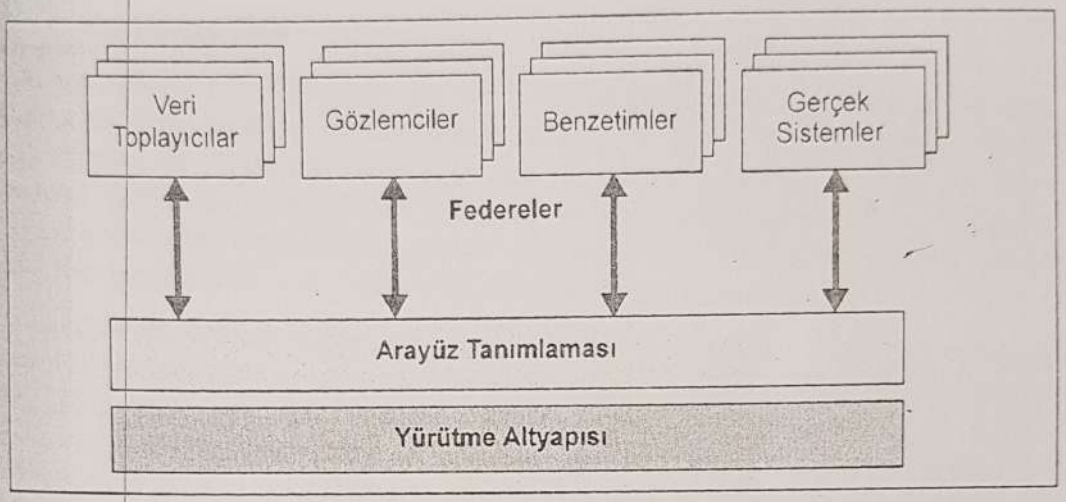
OMG tarafından oluşturulan bir diğer belirtim de Data Distribution Service (DDS - Sürüm 1.2, Ocak 2007) adını taşımaktadır. Bu standartta, "Veri-Merkezli Yayınla-Abone Ol (DCPS)" ve "Veri Yerel Tekrar Birleştirme Katmanı (DLRL)" tanımlanmaktadır. DCPS, belirli yöntemlerle tanımlanan *topic* adlı veri parçalarını yaymak, yayılan bu verilere belirli kurallara göre abone olup okumak yöntemine dayanmaktadır. Bu şekilde gerçek-zamanlı bir program, bir diğerinin yayınladığı bilgilerden yalnızca kendi ilgilendiklerine ulaşabilir; iletişim, platformdan bağımsız, etkin, nitelikli ve güvenilir olarak sağlanır. DCPS, uygulama programlarına belirli tipte verilerle işlem yapmak için arayüz sağlar. DLPL ise, yayınlanan verilerin otomatik olarak toplanıp uygulama tarafından istendiğinde okunabilir hale getirilmesini sağlar. DDS, verilerin hem yerel bilgisayarda, hem de ağ üzerinde ilgilenen abonelere ulaştırılmasını sağlayan bir ara katman belirtimidir. Bu belirtime uyan yazılımlar birbirleri ile rahatça haberleşebilirler.

Tablo-7.1. CORBA servisleri.

Servis	Açıklama
İsimlendirme (Naming)	Bu servis ile bir nesneye özel, tanımlayıcı bir isim verilmesi, kullanıcıların bu isim ile nesneye şebekenin her yerinden ulaşmaları sağlanır. Kullanıcı nesne, bir nesneyi ilk kullandığı anda, kodlayıcıdan saydam olarak, ORB'tan nesnenin bir adresini alır ve sonraki iletişimi doğrudan bu nesne ile yapar.
Olaylar (Events)	Nesneler arasında koşutzamanlı işlem çağrılarına ek olarak, tanımlı bir olay meydana geldiğinde bu olay, Olay Kanalı (Event Channel) adı verilen bir ortam üzerinden ilgilenen tüm nesnelere eşzamansız olarak dağıtılır.
Uyarma (Notification)	Bu servis Olay Servisi'ni genişleterek süzgeç düzeneğini gerçekleştirir.
Güvenlik (Security)	Bu servis kontrol listeleri ve ORB yeteneklerine erişimi kısıtlayan genel bir güvenlik sağlar.
Ara İşlemler (Transactions)	Bu servis ile bir veritabanına atomik olarak erişim denetimi sağlar.
Değişme (Trading)	İsimlendirme Servisi'ne ek olarak bir nesnenin adresini almak için süzgeç kullanımını destekler.
Yaşam Süreci (Life Cycle)	Nesnelerin yaratılmaları, kopyalanmaları, silinmeleri ve varlıkları hakkındaki bilgiler bu servis tarafından sağlanır.
Dışarıverim (Externalization)	Bu servis ile bir nesnenin iç yapısını bir sekizli dizisi haline getirip dışarıda bir yerde saklamak mümkün olmaktadır.
Lisanslama (Licensing)	Uygulama yazılımlarının yüklenmiş lisanslarını kontrol etmek üzere kullanılan bir servistir.
Zaman (Time)	Bu servis ile sistem zamanını alınıp çeşitli şekillerde kullanılabilir.

11.2.4.3. HLA

Yüksek Düzey Mimari (High Level Architecture - HLA), gerçekleştirilen benzetim yazılımlarının tekrar kullanılabilmelerini ve birlikte çalışabilmelerini sağlayan bir modelleme ve benzetim mimarisidir. HLA, eğitim, çözümleme, mühendislik, eğlence ve askeri uygulamalar gibi geniş bir uygulama alanına sahiptir. HLA herhangi bir yazılım tekniği olmadığı gibi, herhangi bir paket yazılımı veya programlama dilini kullanmayı da zorlamaz. HLA, değişik model ve benzetimlerin birbirleriyle, daha çok, C4I (Command, Control, Communication, Computers and Intelligence) sistemleri ile birlikte çalışabilmeleri ve tekrar kullanılabilmeleri için geliştirilmiş bir yapıdır. A.B.D. Savunma Modelleme ve Benzetim Ofisi (Defense Modelling and Simulation Office - DMSO) yönetiminde, Amerikan Savunma Bakanlığı bünyesinde geliştirilen ve bakımı yapılan benzetim yazılımlarının birlikte çalışabilmelerine ve tekrar kullanılabilmelerine destek vermek amacıyla oluşturulmuş olan HLA standardı, Ekim 2000'de IEEE tarafından açık bir standart (IEEE Standard 1516) olarak kabul edilmiştir. HLA, özellikle aşağıdaki uygulama alanları için geliştirilmiştir:



Şekil-7.4. HLA bileşenleri.

- Analitik benzetimler
- Eğitim amaçlı benzetimler
- Askeri sistemler ve C4I arayüzleri
- Test ve değerlendirme amacıyla kullanılan benzetimler
- Mühendislik amacıyla kullanılan benzetimler
- Üretim sistemlerinin benzetimleri
- Eğlence dünyası için dağıtık oyunlar

HLA'yı oluşturan üç temel işlevsel bileşen Şekil-7.4 te gösterilmektedir. Bunlar:

- **Federasyon**
  Federe olarak adlandırılan benzetim nesnelerinin bir amaca yönelik olarak bir araya getirildiği yapıya *federasyon* adı verilmektedir. Federeler, kendi başına çalışan ve diğer federeler ile çalışma sırasında HLA standardına uygun olarak veri değişimi ve nesne paylaşımında bulunabilen benzetimlerdir. Benzetimler haricinde veri toplama ve izleme yapan destek uygulamaları da federe olarak adlandırılırlar. Federasyon kuralları, bir HLA federasyonunun içindeki benzetimlerin düzgün etkileşimini sağlayan ve sorumluluklarını belirleyen kurallardır. Ayrıca gerçek sistemler veya platformlar da HLA'nın sağladığı esnek yapı sayesinde kurallara uyarak birer federe şeklinde federasyonlara katılabilirler.
- **Yürütme Altyapısı**
  Yürütme Altyapısı (Run-Time Infrastructure - RTI), dağıtık bir işletim sistemi gibi çalışarak farklı platformlarda koşan federelerin etkileşimi ve tüm federasyonun yönetimi için gerekli olan destek işlevlerini sağlar. Yürütme sırasında federeler arasındaki tüm veri iletişimi ve etkileşimler Yürütme Altyapısı üzerinden yürütülür. Yürütme Altyapısı, benzetimlere sağlanan servisleri ve bu servislere federeler tarafından nasıl ulaşılacağını belirler. Ayrıca, federeler tarafından altyapıya sağlanması gereken çağrı yordamlarını tanımlar.

- **Arayüz Tanımlaması**
  Arayüz Tanımlaması, federelerin ve Yürütme Altyapısı'nın birbirleri ile iletişimi sırasında kullanılacak standart yöntemleri belirler. Bu yöntemler toplam altı yönetim alanına bölünmüştür:
  - Federasyon yönetimi
  - Bildirim yönetimi
  - Nesne yönetimi
  - Ait olma yönetimi
  - Zaman yönetimi
  - Veri dağıtım yönetimi

11.2.4.4. Ticari Ara Katmanlar

Çeşitli yazılım firmalarının, araştırma kurumlarının ve üniversitelerin geliştirdiği çok sayıda ara katman ürünleri bulunmaktadır. Bunların bir kısmı belirli bir işletim sistemini destekleyerek başarımını en üst düzeye çıkarmaktadırlar. Bir kısmı da daha geniş kapsamlı olarak sunulmaktadır. Birçok ürün aşağıdaki ortak özellikleri desteklemektedir:

- Geniş çaplı iletişimin gerektiği durumlarda, İnternet'in küresel erişim gücünü kullanabilecek uygulama geliştirebilme yeteneği
- Yazılım geliştiricilerin yeniden eğitimine gerek duyulmadan, hızlı bir şekilde yeni geliştirme yapabilme ve ürünü pazara çabuk sunabilme
- Tüm işlevselliğin en az kod yazarak sağlanması
- İşlevsel olarak tüm katmanların halen kullanılan herhangi bir sistemle uyum içinde çalışabilme yeteneği
- Bilgi işlem uzmanlarına yazılım öğelerinin iletişimi ile uğraşmak yerine iş problemlerini çözme zamanı sağlaması
- Programlama dilinden ve donanımdan bağımsızlık
- İstenen dağıtık ve paralel uygulamayı kolaylıkla oluşturma, yönetme ve gerektiğinde değiştirebilme.

Ticari ara katmanlar çok çeşitli amaçlar taşıdıkları ve genellikle özel olarak geliştirildikleri için burada daha fazla ayrıntıya girmeyeceğiz.

11.3. Kodlama Biçimleri

Tasarım ne kadar iyi yapılırsa yapılsın onu hayata geçiren işlem kodlamadır. Günümüzde kodlama, yazılım geliştirme sürecinin hala büyük emek gerektiren, önemli, fakat bazıları için biraz can sıkıcı bir evresidir. Birçok dosya ile uğraşmak, binlerce satır kod yazmak, bunları doğru yazmak, derleyiciden başarıyla geçirmek, test etmek ve sonunda da doğrulamak büyük emek gerektirir. Bu emeğin iyi bir ürüne dönüşebilmesi için bazı biçimlere uyulması, bazı kuralara dikkat edilmesi gereklidir. Şimdi bunlardan bazılarına değinelim:

11.3.1. Kodlama Dili

Yazılım geliştirmede Türkçe ya da bir başka dil kullanımı çoğu zaman tartışma konusudur. Belgelendirme ve kodlamada hangi dilin kullanılacağını bazen müşteri belirler bazen de yönetimin kararı uygulanır. Tamamen yurtiçinde kalacak ve Türk personel tarafından geliştirilecek yazılımın tüm belgelendirmesi ve kodlanması Türkçe yapılabilir. Öte yandan, yabancı kişilerin katılım sağladığı ortak yazılım geliştirme projelerinde ortak anlaşılabilen bir dil kullanılmalıdır.

Günümüzde kullanılan yaygın programlama dillerinin tamamı İngilizce anahtar sözcükler içermekte ve yalnızca İngiliz alfabesindeki harfleri desteklemektedir. Hatta PASCAL, ADA gibi bazı dillerin sözdizimi dahi İngilizce konuşma diline yaklaşmıştır. Derleyicilerin çoğu da yalnızca İngiliz alfabesini desteklemektedirler. Böyle yaygın programlama dilleri Türkçe ile kullanılmaya çalışıldığında, harf yetersizliğinden dolayı, bazı değişken ve yordam adları Türkçeden uzaklaşmaktadır.

Türkçenin bozulmasını engellemek amacıyla kaynak kodun İngilizce olarak yazılması daha tutarlı olmaktadır. Türkçe harfleri destekleyen dil veya derleyici kullanmak başka ortamlara taşınabilirliği büyük ölçüde azaltmaktadır. Bazı metin düzenleyiciler ve derleyiciler Türkçe harfleri desteklememektedirler. Hatta bazı işletim sistemleri Türkçe dosya isimlerinde bile sorun çıkarmaktadırlar. Öte yandan, pek çok yazılım mühendisliği destek gereci Türkçe dilini henüz desteklememektedir. Onların ürettiği yapılar, belgeler, diyagramlar ve şablonlar da Türkçe olmamaktadır. Hele bir de daha önce İngilizce kullanılarak geliştirilmiş olan kütüphanelerin ya da otomatik kod üreticilerin ağırlıklı kullanımı gerektiğinde, ortaya karma dille yazılmış kaynak kod çıkabilmekte ve kodun okunabilirliği en aza inmektedir.

Eğer kodlamada Türkçe kullanılacaksa, olabildiğince Türkçe karakter içermeyen sözcüklerin seçilmesine dikkat edilmelidir, anlam karmaşasına engel olunmalıdır. Kaynak kodun İngilizce, belgelendirmenin Türkçe yapılması seçilebilecek uygun bir şekil olarak değerlendirilebilir. Bu durumda karşılaşılabilecek bir başka sorun da özel isim ve kısaltma karmaşasıdır. Örneğin, belgelerde isterlerden tasarım sonuna kadar kullanılan ve Türkçe harf içeren bir kısaltma, kodlama sırasında bir başka anlam ifade etmesi olası bir isime dönüşmek zorundadır. Yine, koda yakınlığı sağlayabilmek için tasarımın da aynı dilde yapılması gereklidir. Bu amaçla gerekirse bir dönüşüm tablosu kullanılmalıdır.

Hangi aşamada hangi dilin kullanılması gerektiği yönetim tarafından alınması gerekli önemli bir karardır. Bu kararın verilmesi tamamen teknik amaçla olmalı, yalnızca anadilimizin zarar görmesini engellemeye çalışmak hedeflenmemelidir.

11.3.2. Kod Belgelemesi

Tasarım her ne kadar iyi belgelenmiş olsa da, kaynak kodun içinde yapılan belgeleme en kullanışlı ve en doğru olanıdır. Bu belgeleme işlemi içinde, tanımlayıcı isimlerinin seçilmesi, açıklamalar ve görsel yapı yer alır. Şimdi bu özellikleri içerecek şekilde kod belgeleme kurallarına değinelim:

- **Arayüz Tanımlaması**
  Arayüz Tanımlaması, federelerin ve Yürütme Altyapısı'nın birbirleri ile iletişimi sırasında kullanılacak standart yöntemleri belirler. Bu yöntemler toplam altı yönetim alanına bölünmüştür:
  - Federasyon yönetimi
  - Bildirim yönetimi
  - Nesne yönetimi
  - Ait olma yönetimi
  - Zaman yönetimi
  - Veri dağıtım yönetimi

11.2.4.4. Ticari Ara Katmanlar

Çeşitli yazılım firmalarının, araştırma kurumlarının ve üniversitelerin geliştirdiği çok sayıda ara katman ürünleri bulunmaktadır. Bunların bir kısmı belirli bir işletim sistemini destekleyerek başarımını en üst düzeye çıkarmaktadırlar. Bir kısmı da daha geniş kapsamlı olarak sunulmaktadır. Birçok ürün aşağıdaki ortak özellikleri desteklemektedir:

- Geniş çaplı iletişimin gerektiği durumlarda, İnternet'in küresel erişim gücünü kullanabilecek uygulama geliştirebilme yeteneği
- Yazılım geliştiricilerin yeniden eğitimine gerek duyulmadan, hızlı bir şekilde yeni geliştirme yapabilme ve ürünü pazara çabuk sunabilme
- Tüm işlevselliğin en az kod yazarak sağlanması
- İşlevsel olarak tüm katmanların halen kullanılan herhangi bir sistemle uyum içinde çalışabilme yeteneği
- Bilgi işlem uzmanlarına yazılım öğelerinin iletişimi ile uğraşmak yerine iş problemlerini çözme zamanı sağlaması
- Programlama dilinden ve donanımdan bağımsızlık
- İstenen dağıtık ve paralel uygulamayı kolaylıkla oluşturma, yönetme ve gerektiğinde değiştirebilme.

Ticari ara katmanlar çok çeşitli amaçlar taşıdıkları ve genellikle özel olarak geliştirildikleri için burada daha fazla ayrıntıya girmeyeceğiz.

11.3. Kodlama Biçimleri

Tasarım ne kadar iyi yapılırsa yapılsın onu hayata geçiren işlem kodlamadır. Günümüzde kodlama, yazılım geliştirme sürecinin hala büyük emek gerektiren, önemli, fakat bazıları için biraz can sıkıcı bir evresidir. Birçok dosya ile uğraşmak, binlerce satır kod yazmak, bunları doğru yazmak, derleyiciden başarıyla geçirmek, test etmek ve sonunda da doğrulamak büyük emek gerektirir. Bu emeğin iyi bir ürüne dönüşebilmesi için bazı biçimlere uyulması, bazı kuralara dikkat edilmesi gereklidir. Şimdi bunlardan bazılarına değinelim:

11.3.1. Kodlama Dili

Yazılım geliştirmede Türkçe ya da bir başka dil kullanımı çoğu zaman tartışma konusudur. Belgelendirme ve kodlamada hangi dilin kullanılacağını bazen müşteri belirler bazen de yönetimin kararı uygulanır. Tamamen yurtiçinde kalacak ve Türk personel tarafından geliştirilecek yazılımın tüm belgelendirmesi ve kodlanması Türkçe yapılabilir. Öte yandan, yabancı kişilerin katılım sağladığı ortak yazılım geliştirme projelerinde ortak anlaşılabilen bir dil kullanılmalıdır.

Günümüzde kullanılan yaygın programlama dillerinin tamamı İngilizce anahtar sözcükler içermekte ve yalnızca İngiliz alfabesindeki harfleri desteklemektedir. Hatta PASCAL, ADA gibi bazı dillerin sözdizimi dahi İngilizce konuşma diline yaklaşmıştır. Derleyicilerin çoğu da yalnızca İngiliz alfabesini desteklemektedirler. Böyle yaygın programlama dilleri Türkçe ile kullanılmaya çalışıldığında, harf yetersizliğinden dolayı, bazı değişken ve yordam adları Türkçeden uzaklaşmaktadır.

Türkçenin bozulmasını engellemek amacıyla kaynak kodun İngilizce olarak yazılması daha tutarlı olmaktadır. Türkçe harfleri destekleyen dil veya derleyici kullanmak başka ortamlara taşınabilirliği büyük ölçüde azaltmaktadır. Bazı metin düzenleyiciler ve derleyiciler Türkçe harfleri desteklememektedirler. Hatta bazı işletim sistemleri Türkçe dosya isimlerinde bile sorun çıkarmaktadırlar. Öte yandan, pek çok yazılım mühendisliği destek gereci Türkçe dilini henüz desteklememektedir. Onların ürettiği yapılar, belgeler, diyagramlar ve şablonlar da Türkçe olmamaktadır. Hele bir de daha önce İngilizce kullanılarak geliştirilmiş olan kütüphanelerin ya da otomatik kod üreticilerin ağırlıklı kullanımı gerektiğinde, ortaya karma dille yazılmış kaynak kod çıkabilmekte ve kodun okunabilirliği en aza inmektedir.

Eğer kodlamada Türkçe kullanılacaksa, olabildiğince Türkçe karakter içermeyen sözcüklerin seçilmesine dikkat edilmelidir, anlam karmaşasına engel olunmalıdır. Kaynak kodun İngilizce, belgelendirmenin Türkçe yapılması seçilebilecek uygun bir şekil olarak değerlendirilebilir. Bu durumda karşılaşılabilecek bir başka sorun da özel isim ve kısaltma karmaşasıdır. Örneğin, belgelerde isterlerden tasarım sonuna kadar kullanılan ve Türkçe harf içeren bir kısaltma, kodlama sırasında bir başka anlam ifade etmesi olası bir isime dönüşmek zorundadır. Yine, koda yakınlığı sağlayabilmek için tasarımın da aynı dilde yapılması gereklidir. Bu amaçla gerekirse bir dönüşüm tablosu kullanılmalıdır.

Hangi aşamada hangi dilin kullanılması gerektiği yönetim tarafından alınması gerekli önemli bir karardır. Bu kararın verilmesi tamamen teknik amaçla olmalı, yalnızca anadilimizin zarar görmesini engellemeye çalışmak hedeflenmemelidir.

11.3.2. Kod Belgelemesi

Tasarım her ne kadar iyi belgelenmiş olsa da, kaynak kodun içinde yapılan belgeleme en kullanışlı ve en doğru olanıdır. Bu belgeleme işlemi içinde, tanımlayıcı isimlerinin seçilmesi, açıklamalar ve görsel yapı yer alır. Şimdi bu özellikleri içerecek şekilde kod belgeleme kurallarına değinelim:

11.3.2.1. İsimlendirme

Bir yazılım kaynak kodu içinde kullanılan veri tipi, yordam, değişken ve sabitlerin isimleri (tanımlayıcılar) belirli bir düzen içinde verilmelidir. Bu düzeni sağlamak için şu noktalara dikkat edilmelidir:

- Karmaşıklığa yol açabilecek veya başkası tarafından okunduğunda anlaşılmayı güçleştirecek isimler (dosya, yordam, değişken adı vb.) kullanılmamalıdır.
- Evrensel bir değişken veya sınıf ismi aynı zamanda yerel olarak kullanılmamalıdır.
- Proje boyunca aynı isimlendirme yöntemi kullanılmalıdır.
- Bazı dillerde büyük ve küçük harf ayrımı olduğu unutulmamalı, buna göre bir isimlendirme yöntemi seçilmelidir (örneğin, küçük ve büyük harf ayrımı yapan diller için, sınıf ve yordam isimlerinde baş harfi büyük sözcükler, değişkenler ve nesneler için tamamı küçük harften oluşan sözcükler kullanılabilir).
- Anlaşılabilirliği arttırmak için uzun ve birden fazla sözcükten oluşan isimler kullanmaktan kaçınılmamalı, ancak bunda da aşırıya kaçılmamalıdır. Aşırı uzun sözcükler hem okumayı zorlaştırır hem de yazmayı. Bazı derleyicilerin dikkate aldıkları karakter sayısı da kısıtlı olabilir.
- Yazılım küçük tutmak amacıyla kısa isimler kullanımı doğru değildir. İsimlerin uzun veya kısa olması çalışmayı ve boyutu etkilemez.
- Değişken ve nesne isimlerinde birden fazla sözcük kullanılacaksa aralarına alt çizgi işareti (_) konmalı veya her sözcüğün baş harfi büyük yazılarak yan yana getirilmelidir. Bir başka yöntem düşünülüyorsa tüm proje içinde tutarlılık korunmalıdır.
- İsimlerin baş harfi rakam olmamalıdır. Ancak, benzer özellikte ve bir sıra takip eden değişkenleri göstermek için rakam içeren isimler kullanılmalıdır.
- Bir dilde anahtar sözcük veya onun yakın bir benzeri isim olarak kullanılmamalıdır. Zira, bazı derleyiciler, kullanım yerine göre bunu ayırt edebilirlerken bazıları edememekte ve kodun taşınması gerektiğinde çok büyük sorunlar ortaya çıkmaktadır.
- Program başında tanımlanan sabitlerin veya makroların isimlendirilmesinde büyük harfler kullanılması daha uygundur.
- Küçük öbekler ve döngü değişkenleri dışında tek harfli isimler veya tanımlayıcılar kullanılmamalıdır. Özellikle zorunlu evrensel değişkenlerde çok kısa isimlerden kesin olarak kaçınılmalıdır.
- Dosyalara verilecek isimler işletim sistemine bağımlı olduğu için her zaman anlaşılır isimler vermek mümkün olmayabilir. Bu durumda mantıklı kısaltmalar yapılmalı ya da sıra numaralarına sahip dosya isimleri kullanılmalıdır. İçerik açısından benzeşen dosyaların baş harflerini aynı tutmak kalabalık bir çalışma dizininde işletim kolaylığı sağlayacaktır.
- Günümüzde kullanılan programlama dilleri ve derleyiciler İngilizce alfabeye göre yapıldıkları için, diğer dillere özgü bazı harfler desteklenmemektedir. Bu

nedenle, İngilizce dışında bir dil kullanmak, İngilizce alfabe dışında kalan harfleri kullanmak gerektirdiği için, o dilde anlamsız, hatta başka anlamlara çekilebilen garip sözcükler ortaya çıkmasına neden olabilmektedir. Bu durumu göz önünde bulundurarak, okunabilirliği artırmak amacıyla, program kodunda mümkünse yalnızca İngilizce kullanmaya gayret edilmelidir.

- İsimlendirmede Türkçe ve İngilizce sözcükleri karıştırarak kullanmaması, kodun tamamında projenin başında seçilen dilin kullanılması önerilir!

11.3.2.2. Açıklamalar

Program kaynak kodunun çalışmasını en iyi açıklayan kodun kendisinin yanında kullanılan *açıklama satırları*dır (comment). Bu şekilde doğal bir dil ile o anda ne yapıldığı, o yordamın ne işe yaradığı anlatılır. Daha sonradan kod okunurken yalnızca bu açıklamalarla yetinmek bile mümkün olabilir. Bazı modern yazılım geliştirme araçları tasarımda oluşturulan açıklamaları koda yansıtabilmektedirler. Bazı tersine mühendislik araçları da kodu tarayıp çeşitli belgeler ve açıklamalar üretebilmektedirler. Ancak yine de kodlayıcının kendi koyduğu açıklamalar en geçerli olanlardır. Şimdi de açıklama yazmak için bazı kurallar görelim:

- Her türlü yazılımın mutlaka bir iç belgelendirmesi olmalıdır. Bunun yollarından biri de açıklama satırlarıdır.
- Kodun sonradan bakımı yapılması gerektiğinde en büyük yardımcı son halini gösteren açıklama satırlarıdır.
- Açıklama satırlarının değiştirilmesi, sonradan bakımı biraz zor olabileceği için aşırıya kaçılmamalıdır.
- Her modülün başında standart yapıda açıklama bulunmalıdır. Bu yapı şu bilgileri içerebilir:
  - Modülün veya yordamın adı, saklama düzenindeki numarası
  - Modülün amacının anlatımı
  - Arayüz tanımlaması (çağrı şekli, giriş ve çıkış parametreleri)
  - Önemli değişkenler ve kısıtlamalar
  - Geliştirme tarihçesi (yazarın adı, geliştirme tarihi, değişiklik yapanların isimleri, tarihleri ve açıklamalar)
- Kod içinde yapılan işi tanımlayıcı açıklamalar da şu bilgileri içerebilir:
  - Her satır yerine bir öbek için açıklama yazılmalıdır.
  - Boş satırlar ve başlangıçları hizalayarak açıklamaların kolay okunması sağlanmalıdır.
  - Açıklamalar kodun yaptığı işle uyumlu olmalı, yanlış anlaşılmaya neden olmamalıdır.
  - Uygulama alanındaki kodlayıcılar tarafından bilinen veya bilindiği kabul edilen olgular için açıklama kullanılmamalıdır.
  - Bazı kod yazma araçları programlama diline göre açıklama satırlarını farklı yazı tipi ve renkte gösterebilirler. Bu özelliğe sahip araçlar kullanmak genel verimi artırır.

- Çok sıkı denetlenen nitelik güvence yöntemleri uygulandığı takdirde, açıklama satırları arasında, hangi yazılım isterinin karşılandığı yazılır.

• Açıklama satırlarında kullanılacak dil, İngilizce olabileceği gibi Türkçe de olabilir (bazı yerlerde Türkçe harflerin kullanılamayacağı dikkate alınarak).

11. 3.2.3. Hata Ayıklama

Hata ayıklama ilk kodlama sırasında kodlayıcının en çok yaptığı iştir. Yazılan her kod parçası derleyiciden geçirilerek hatasız bir şekilde nesne koduna dönüştürülmek zorundadır. Sonra da birleştirilip bağlanarak yürütülebilir kod elde edilir. Bundan sonra, birçok programlama dili yürütme anında yazılımın içini görme şansı vermez. Özellikle hata ayıklama gereksinimi ortaya çıktığında geliştiricinin en klasik yardımcısı standart çıkış ekranına kod içinden iletiler yazdırmaktır (`cout`, `printf`, `put_line` gibi komutlarla). Bazı geliştirme ortamları bu amaçla hata ayıklayıcı yazılımlar sunarlar. Ancak, bazen bunları kullanmak mümkün olmaz. Örneğin, paralel çalışan, zaman ve olay bağlılığı gerektiren yazılım birimleri, çok görevcikli yazılımlar için hata ayıklama araçları kullanılamaz. Böyle durumlarda en iyi yardımcı, kod içindeki bazı noktalara gelindiğinde ekrana bir ileti vermek ve o anki değişkenlerin değerlerini ekrana veya bir dosyaya yazdırmaktır. Kod içine bu tür komut satırları eklemek yazılım tamamlanıp kullanıma verildiğinde büyük bir başarım düşüklüğüne yol açar. Çünkü, ekrana çıkış yapmak işlemciyi gerçekten çok meşgul eder. Bu satırları kod içinden ayıklamak ise ileride gereksinim duyulduğunda tekrar hazırlamayı gerektirir.

Ekran çıktılarını her zaman değil de yalnızca istendiği zaman ve istenen miktarlarda kullanabilmek için mutlaka bir düzenek kullanılmalıdır. Bazı programlama dilleri ve derleyiciler koşullu derleme özelliği ile bu tür kod parçalarının tamamen gizlenmesini sağlayabilirler (C ve C++ için `#ifdef - #endif` yapısı gibi). Şekil-7.5, C++ için iki adet düzenek örneği göstermektedir:

```
#define DEBUGPRINT
...
#ifdef DEBUGPRINT
   cout << "X = " << xval << endl;
#endif
-------------------------------------------------
#ifdef DEBUGPRINT
#define PRINT1(A)      cout << A << endl
#define PRINT2(A,B)    cout << A << B << endl
#define PRINT3(A,B,C)  cout << A << B << C << endl
#else
#define PRINT1(A)      /*PRINT1(A)*/
#define PRINT2(A,B)    /*PRINT2(A,B)*/
#define PRINT3(A,B,C)  /*PRINT3(A,B,C)*/
#endif
```

Şekil-7.5. Koşullu derleme örneği.

Böyle düzenekler, toplam geliştirme sürecini kısaltıcı ve sonradan bakım sırasında hata bulmada zaman azaltıcı etki sağlarlar. Bu tür deyimleri kod içine bolca serpiştirmek ise tam tersi etki yaparak okunabilirliği iyice düşürür. O nedenle dikkatli olmalı ve yalnızca amacı karşılayacak kadar az miktarda iletiler verilmelidir. Özellikle gerçek zamanlı sistemlerde, olayın yakalandığı anda böyle raporlamalar yaptırmak son derece risklidir.

Hata ayıklama konusuna Ayrıt 8.4 te yeniden ele alınacaktır.

11.3.3. Veri Bildirimi

Veri yapıların düzeni ve karmaşıklığı tasarım aşamasında, veri bildiriminin biçimi ise kodlama sırasında belirlenir. Yazılım biriminde kullanılan verileri anlaşılır ve bakımı kolay olacak şekilde düzenleyebilmek için aşağıdaki kuralları dikkate almak yararlı olacaktır:

- Aynı yazılım paketi içinde ortak olarak kullanılan veri tipleri uygun bir dizinde tutularak tekrarların önüne geçilmelidir. Örneğin, C ve C++ dilleriyle geliştirilen yazılımda kullanılan ortak veri tipleri (`typedef`) başlık dosyalarına yerleştirilmeli ve bu dosyalar, derleme sırasında taranan bir dizinin içine konmalıdır.
- Bir tek program içinde ortak olarak kullanılan veri tipleri aynı dizinde bulunan ve beden dosyaları tarafından içerilen başlık dosyalarına konmalıdır.
- Soyut veri tipleri tanımlanması gerektiğinde, Bilgi Gizleme ve Yerelleştirme ilkelerine bağlı kalınması için bir dosya içinde kullanılan veri tipleri o dosyanın başında, dilin izin verdiği sözdizime göre, bir sıraya göre yerleştirilmelidir.
- Veri yapıları, veri tiplerinin uygun şekilde birleştirilmesi ve bildirimi ile ortaya çıkar. Evrensel veri yapıları ve değişkenler, görülebilirlik kurallarına uygun olarak belirli bir dosyada toplanmalıdırlar.
- Veri tiplerinden değişken bildirimi yapılırken, tek sırada değil de alt alta ve alfabetik bir sıra izlenmesi aranan bir değişken isminin kolay bulunmasını sağlar.

11.3.4. Deyim Yapıları

Yazılım biriminin gerçekleştirilmesi ve mantıksal akışın sağlanması kodlama sırasında yazılan deyimlerle, yani ifadelerle gerçekleşir. Deyimleri yazarken dikkat edilecek bazı kurallar okunması ve anlaşılması rahat bir kaynak kod oluşmasını sağlar. Şimdi biraz bu kurallara değinelim:

- Her deyim bir satırda yer almalı, gereksiz karmaşıklıklar içermemelidir.
- Çevrim ya da koşul testleri ile dallanma gibi programlama diline bağlı yapılar uygun satırbaşı girintileri ile hizalanarak kolay okunabilir hale getirilmelidir.
- Karmaşık koşul testleri (`if` yapıları) kullanmaktan kaçınılmalıdır.
- İç içe girmiş fazla sayıda döngü ve koşullu dallanmalar kullanılmamalıdır.
- Koşullarda gereksiz ve tekrarlanan testler yapılmamalıdır.

9. Hafta
1. Grup
Son

- Mantıksal ve aritmetik ifadeleri daha açık ve anlaşılır hale getirmek için parantezler kullanılmalıdır.
- Deyim içinde uygun şekilde boşluk karakteri kullanılarak okunabilirlik artırılmalıdır.
- Deyimler, yazan kişiden başkasının da okuyup anlayabilmesi için olabildiğince basit, ancak işlevini yapabilecek şekilde yazılmalıdır.