8. Hft
2. GrP
Bslnyz

1.3. Dillerin Özellikleri

Programlama dillerinin karşılaştırılmasında çeşitli özelliklerin ve yeteneklerin varlığı esas alınır. Dillerin özelliklerini genellemek ve bir kısmını da daha alt gruplar için kullanmak mümkündür. Genel özellikler tüm diller için karşılaştırma yapılabilmesine olanak tanırken, bazı özellikler alt gruplardaki dillerin karşılaştırılmasında kullanılır.

1.3.1. Genel Özellikler

Programlama dillerinin genel özelliklerini şu şekilde özetleyebiliriz:

- **Tasarımdan koda geçiş kolaylığı**
  Kuramsal olarak, iyi yapılmış bir ayrıntılı tasarımı koda çevirmek kolay bir işlemdir. Ancak bunun için programlama dilinin tasarımda öngörülen yapıları desteklemesi gereklidir. Bu yapılar arasında, veri tanımlamaları, giriş/çıkış düzenekleri, zaman belirtimi, hata kotarımı, aritmetik, mantık ve bit işlemleri, nesneye yönelim özellikleri, soyut veri tipleri, modülerlik sayılabilir.

- **Amaca uygunluk**
  Uygulama alanının ve geliştirilecek yazılımın işlevsel ve başarım özelliklerine göre programlama dili seçimi yapılmalıdır. Bu seçim, tasarım sırasında da etkisini ortaya koyar. Zira, dilin desteklemediği bir yapıyı tasarlamak ve kodlamaya çalışmak yerine, amaca uygun yapılara sahip bir dili seçerek onun özelliklerini tasarım sırasında kullanmak daha yararlı olacaktır. Örnek olarak, C++ dilini seçip tasarımı nesneye yönelik olarak yapmayı, ADA dilini seçip `task` özelliğini kullanarak paralel programlama yapmayı verebiliriz.

- **Dilin etkinliği**
  Yaygın programlama dilleri İngilizce üzerine kurulmuş olup konuşma diline yaklaştıkça hem yazılması hem de okunması kolaylaşır, düşüncelerin makineye anlatılması daha etkin olur. Bu da kodlamayı yapan insanların verimliliğini artırır. Yapıları zayıf, okuması, yazması ve anlaması güç bir dil ile bilgisayara istenileni yaptırmak son derece güç olacağından emek ve zaman kaybı kaçınılmaz olur.

- **Derleyici etkinliği**
  Aynı kaynak kod kullanılmasına rağmen her derleyici aynı nitelikte kod üretimi yapamaz. Programlama dilinden üretilen makine kodunun etkinliği yürütme

sırasında ortaya çıkar. Son zamanlarda bilgisayar donanımları çok gelişmiş olsa da makine kodunun hızlı çalışması, mimariyi iyi kullanması, bellek kullanımını etkin yapması özellikle gerçek zamanlı sistemler için arzu edilen özelliklerdendir. Derleyiciler kaynak kodu çözümlemeden önce ne kadar iyi denetlerler ve üretilen makine kodunu ne kadar iyileştirirlerse o kadar başarılı sayılırlar.

- **Taşınabilirlik**

  Bir programlama dili ile yazılan kaynak kodun değişik derleyicilerle, işletim sisteminin yeni sürümleriyle ve hatta başka makineler üzerinde sorunsuzca çalışması arzu edilir. Ancak ne yazık ki, birçok derleyici üreticisi genel başarımı artırmak uğruna bazı özel yapılar, bildirimler ve hatta anahtar sözcükler kullanabilmektedirler. Öte yandan, işletim sistemine özgü alt düzey çağrıların kaynak kod ile çok iç içe kullanılması ile taşınabilirlik özelliği azalabilir. Gerçekten taşınabilir bir dil ile yazılmış kaynak kodların, paket yazılımların başka yazılımları geliştirirken tekrar kullanımı mümkün olabilir. Bu da ancak belirli standartlara sahip dillerin kullanılmasıyla gerçekleşebilir. Bu standartlar genellikle ISO (International Standardization Organization), ANSI (American National Standardization Institute) ve OMG (Object Management Group) tarafından sağlanmaktadır.

- **Geliştirme araçları**

  Bir programlama dilinin en özelliklerinden biri geliştirme sırasında kodlayıcıya sunduğu hizmetlerle de ölçülür. Geliştirme araçlarının varlığı aslında derleyici üreticisi olan firmaya bağlıdır. Desteği az bir dilin etkinliğinin yüksek olmasını beklemek doğru olmaz. O nedenle, programlama dili seçiminde, geliştirme ortamı (tümleşik veya komut şeklinde), hata bulma ve ayıklama gereçleri, kodu yazarken renk kodlarıyla yardım veren metin düzenleyiciler, kodu biçimlendiren gereçler, geniş kütüphaneler, örnekler, çapraz derleyiciler, ters mühendislik gereçleri dikkate alınmalıdır.

- **Bakım**

  Yazılım bakımının kolay olmasındaki en büyük etkenlerden biri, esnek bir tasarımın yanında, kaynak kodun niteliğidir. Kod, okunabilir, anlaşılabilir ve yeni tasarıma göre rahatça uyarlanabilir olmalıdır. Anlaşılabilirliği artıran dilin önemli özellikleri arasında da veri tiplerinin tanımı ve değişken bildirimi, tanımlayıcıların ve değişken isimlerinin uzunlukları, etiketleme özellikleri, okunabilir konuşma diline yakın şekilde düzenleme sayılabilir.

- **Tip kontrolü**

  Programlamanın temelinde veri işleme yatar. Veriler de bilgisayar donanımının kullandığı temel tipler (karakter, tamsayı ve kayan nokta sayısı) haline dönüştürülerek kullanılırlar. Bunun yanında, kodlama sırasında kolaylık sağlamak ve güvenilirliği artırmak üzere alt tipler yaratılır; kodlamada bu tiplerden üretilen sabitler ve değişkenler kullanılır. Programlama dilinin tip kontrolü yapma yeteneği güvenilir yazılım geliştirme için son derece gerekli bir özelliktir. Çünkü, insanlar hata yapmaya eğilimlidirler. Elle yazılan kodlar tip

kontrolü zayıf bir derleyicinin denetiminden geçip bazı durumlarda çalışsa bile genelde çökmeye mahkumdurlar. Örneğin, C dilinde `char` ile `int` tipindeki değişkenleri birbirinin yerine kullanmak mümkündür. Oysa, birincisine yapılan dört sekizli için en büyük değer olan 255'ten büyük bir sayı atanması durumunda, derleyicinin herhangi bir hata vermemesine rağmen yürütme sırasında hata meydana gelir. Ancak, kuvvetli tip kontrolüne sahip bir dil ve derleyici ile bu tür esneklik ortadan kaldırılır ve derleme anında yapılan kontroller nedeniyle yürütme anında hatalı çalışma olasılığı çok azalır.

- **Denetim yapıları**
  Tüm programlama dilleri kodlayıcıya ardışık komutlardan, koşul testine göre dallanmalardan ve tekrarlardan oluşan mantıksal yapılar sunarlar. Klasik dillerin bazılarında hata yakalama düzenekleri, paralel programlama yapıları, zaman denetimli yapılar bulunmaktadır. Bunların yanında, dağıtık programlamayı ve gerçek zamanlılığı kendi yapıları ile destekleyen diller de bulunmaktadır.

11.1.3.2. Nesneye Yönelik Dillerin Özellikleri

Alt gruplar için tanımlanabilen özelliklerden nesneye yönelik diller için kullanılan ayırt edici özellikler şunlardır:

- *Modülerlik*: Yazılım, kendi üzerinde işlemler yapılmasına izin veren basit ve tutarlı arayüzlere sahip ayrı modüller halinde geliştirilir.
- *Veri soyutlama* (abstraction): Nesneler hem verileri hem de onlar üzerindeki işlemleri kapsayan soyut veri tipleri şeklinde tanımlanırlar.
- *Otomatik bellek yönetimi*: Sistem özkaynaklarını kullanan nesneler işleri bitince onları otomatik olarak serbest bırakırlar.
- *Sınıflar*: Sınıf, kendisinden nesnelerin yaratılan, fakat diğer basit tiplerden olmayan özel yapıda bir tiptir.
- *Kalıtım* (inheritance): Bir sınıf, temel sınıfın veri ve işlemlerinin bir kısmını ya da tamamını devralarak türetilir. Bu türetimi çoklu ve tekrarlanan bir şekilde yapmak mümkündür.
- *Çokşekillilik* (polymorphism): Bir nesne, yürütme anında, çeşitli sınıflardan yaratılmış başka nesneleri işaret edebilir, yürütme anında dinamik olarak başka nesnelere bağlanabilir.

Bütün koşulları karşılayan dillere *nesneye yönelik*, yalnızca ilk dördünü karşılayan dillere ise *nesneye dayalı* adı verilmektedir. Nesneye yönelik dillere örnek SMALLTALK, C++, JAVA, nesneye dayalı dillere örnek olarak da C, PASCAL ve ADA verilebilir.

11.1.3.3. Gerçek Zamanlı Dillerin Özellikleri

Zamansallığın kısıtlama derecesine göre gerçek zamanlı sistemlerin geliştirilmesinde herhangi bir programlama dilinin kullanımı mümkün olabilir. Ancak, görev kritik olan

ve sıkı gerçek zamanlılık gerektiren sistemler için uygun özellikler taşıyan programlama dillerinin kullanılması daha yararlıdır. Bu özellikler yetenekli ve esnek dillerle karşılanabilse de kodlayıcının son derece dikkatli olması ve ne yaptığını çok iyi bilmesi gereklidir. Şimdi bu özellikleri kısaca görelim:

- *Kuvvetli tip kontrolü* (strong typing) yazılımı geçersiz verilerden korumak için derece gerekli bir özelliktir. Derleme ve yürütme anında yapılan tip kontrolü ile tasarım sırasında beklenen veri büyüklükleri, tipleri ve sınırları denetim altında tutulur. Geçersiz verilerin işleme girmesinin istenmemesi durumunda hata üretilir ve bu hataların uygun şekilde düzeltilmesi beklenir. Özellikle derleme anında yapılan tip kontrolü ile yürütme sırasında ortaya çıkabilecek pek çok hatanın önceden tespit edilerek önlem alınması sağlanabilir.
- *Dinamik bellek yönetimi* kısıtlı belleğe sahip sistemlerde, bilgi girişine göre bellek ataması ve temizlemesi yapmak üzere gereklidir. Dil tarafından etkin bir bellek yönetimi ve atık toplama desteklenmelidir. Eğer, bellek yönetimi için gerekli işlemlere yürütme anında işlemci zamanı ayrılması istenmiyorsa, ilklendirme sırasında yeterli miktarda bellek ayrılarak daha etkin çalışma sağlanabilir.
- *Parametre geçirme teknikleri* yeteri kadar hızlı yapılmalıdır. Örneğin, değer ile geçirme (pass-by-value) tekniğinde, yordam çağrısı sırasında nesne veya değişkenin tümü yığına kopyalanır. Bazı dillerde bu işlem kodlayıcıya saydam olmasına rağmen başarımı arttırır. Başarım bakımından etkileyici rol oynayan ve çok defa çağrılan bu tür yordamlarda adres ya da işaretçi geçirme teknikleri veya evrensel değişken kullanımı tercih edilmelidir.
- *Hata yakalama ve kotarma* (exception handling) yazılımın bütünlüğünü bozmadan yürütme sırasında meydana gelebilecek hataların yakalanması ve durumun düzeltilmesi işlemidir. Bazı yazılımlarda veri hatası sonucu toplam çökme pek önemli olmazken, sürekli çalışması gerekli yazılımlarda oluşan hataları yakalamalı ve tamamen çökmeden gerekli önlemi almalıdır. Bu düzenek yardımı ile yazılımlar hataya dayanıklı olarak geliştirilebilirler.
- *Soyut veri türleri* yardımıyla veriler ve işlevler beraberce kapsanarak bir değişken üzerindeki işlemin diğer değişkeni etkilememesi sağlanır.
- *Modüler yapı*yı destekleme özelliği ile geliştirme sırasında kolaylık sağlanırken daha önce geliştirilmiş ve denenmiş modüllerin tekrar kullanımı mümkün olur.
- *Zaman belirtimi* olanağı ile daha sağlam zaman yapılarını (kesin olarak belirli bir süre bekleme, belirli bir saatte uyanma gibi) kodlama sırasında oluşturmak mümkündür.
- *Kolay okunabilirlik,* yazılan kodun insan anlayışına en yakın şekilde gösterilebilmesidir.
- *taşınabilirlik* ve *genişleme yeteneği* de diğer önemli özellikler arasındadır.

11.1.4. Dil Seçimi

Belirli bir proje için programlama dilinin seçimi genel başarı için büyük önem taşımaktadır. Yeni ve bağımsız olarak başlayan, bu şekilde de devam eden projelerde dil seçimi genellikle gereksinimlere göre belirlenir. Ancak, bazı durumlarda müşteri tarafından belirli bir dilin kullanımı zorunlu tutulabilir; ya da başka teknik nedenlerden dolayı belirli bir dilin kullanılması gerekebilir. Örneğin, önceki yazılımlarla uyumluluk için COBOL dili zorunlu tutulabilir. Halen, birçok savunma amaçlı projede ADA dili zorunlu tutulmaktadır.

Dil seçimi yapılırken aşağıda belirtilen ilkelerin dikkate alınmasında yarar vardır:

- Genel uygulama alanının gereksinimleri göz önünde tutulmalıdır. Özellikle görev kritik ve askeri alanlarda sistem güvenilirliğini zedelemeyecek diller kullanılmalıdır.
- Yazılımın kullanılacağı bilgisayar ortamı, donanımın ve işletim sisteminin özellikleri dikkate alınmalıdır.
- Seçilecek dilin yetenekleri algoritmik ve işlemsel karmaşıklık derecesini karşılayabilir olmalıdır.
- Başarım gereksinimleri, dilin diğer özelliklerinden daha fazla önem taşıyabilir. Çok yetenekli bir dil olmasına rağmen yeterli hıza sahip olmayan bir dil sistemin de başarısız olmasına yol açabilir.
- Veri yapılarının özellikleri ve karmaşıklık dereceleri seçilecek dilin yetenekleri arasında olmalıdır. Aksi takdirde yeterli yazılım niteliği ve başarım elde edilemez.
- Yazılım geliştirme personelinin bilgi düzeyi seçilecek dil için önemli bir etkendir. Yeni bir dili öğrenip deneyim kazanıncaya kadar geçecek sürede iyi bilinen bir dilde geliştirme yapmak daha iyi sonuç verebilir.
- İyi bir derleyicinin ve geliştirme ortamının varlığı verimliliği arttırabilmek için dikkate alınmalıdır.
- Seçilecek dil, yazılımın bakım aşamasında son derece önemli olan okunabilirlik ve anlaşılabilirlik özelliklerini desteklemelidir.

11.1.5. Dillerin Uygulama Alanları

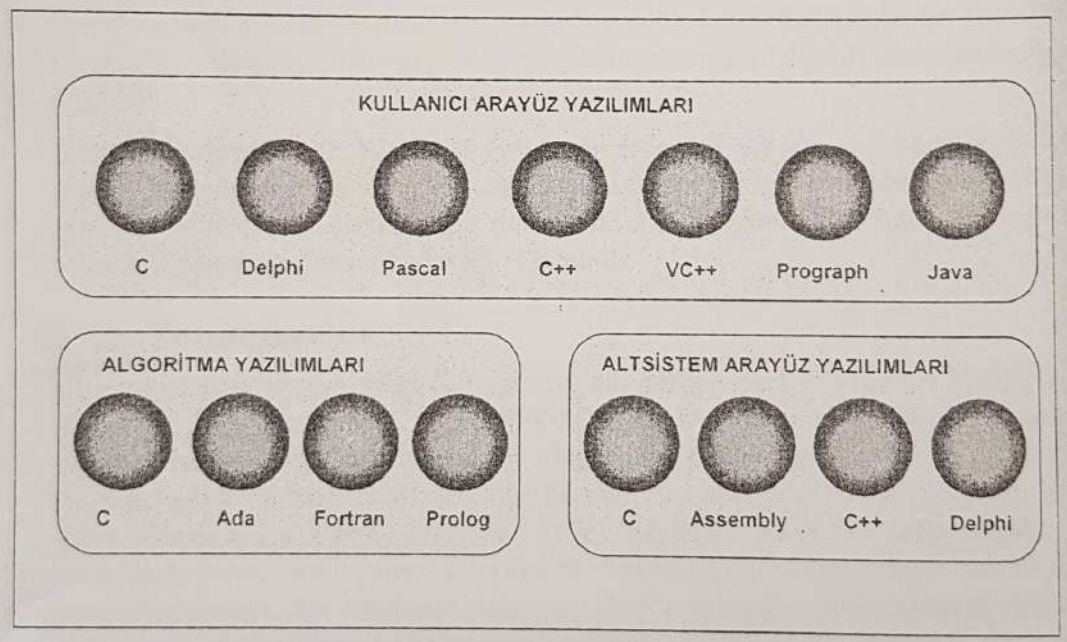
Uzun yıllardır en güçlü ve en yaygın diller arasında C dili gösterilmektedir. Sistem yazılımlarında (işletim sistemi, ara katman, aygıt sürücü gibi) tercih edilen C'nin nesneye yönelik uyarlaması olan C++ da onun kullanıldığı pek çok alanda yaygın olarak kullanılmaktadır.

Eğitim dili olarak yaratılmış PASCAL ile birlikte, görsel programlama dilleri olan Visual BASIC, Visual C++, DELPHI dibi diller özellikle PC dünyasında yaygın olarak kullanılmaktadırlar.

ADA (83 ve 95), C, MODULA-2 gerçek zamanlı sistemlerde, COBOL, LINK iş dünyası yazılımlarında, FORTRAN bazı mühendislik uygulamalarında, PROLOG, LISP

yapay zeka uygulamaları ve uzman sistemlerde daha fazla tercih edilmektedirler. Nesneye yönelik çözümleme ve tasarım kullanılması durumunda, SMALLTALK, Objective-C, C++ ve JAVA ön plana çıkmaktadır. Günümüzde gittikçe yaygınlaşan JAVA özellikle İnternet uygulamalarında ve eğitim merkezlerinde kullanılmaktadır. ADA dili halen A.B.D. ve İngiltere tarafından savaş sistemlerinin kritik kısımlarının geliştirilmesinde zorunlu tutulmaktadır.

Bu deneyimlere göre, bazen bir bilgisayarlı sistemin tamamında tek bir dil kullanmak mümkün olmayabilir. Örneğin, bir kontrol sistemi yazılımında, C ile yazılmış aygıt sürücüler ADA veya C++ diliyle denetlenirken, grafiksel kullanıcı arayüzü için C veya C++ kullanılabilmektedir. Şekil-7.3 çeşitli dillerin beraberce kullanıldığı karma bir yapıyı göstermektedir:



Şekil-7.3. Karma dillerin kullanımı

Çeşitli amaçlar için özel olarak geliştirilmiş fakat fazla yaygın olmayan yeni diller bazı geliştiriciler için fazla tercih edilmeyebilir. Zira, yeni dillerin belgelendirmeleri, kütüphaneleri eskilerinkine oranla daha zayıf olabilmekte veya farklı platformları desteklemekte güçlük çekilmekte ya da öğrenmede güçlüklerle karşılaşılmaktadır. Özellikle o alanda deneyim sahibi kodlayıcı bulabilmek veya var olanları eğitebilmek oldukça güç olabilmektedir.

Yazılımın geliştirilip doğru olarak çalıştırılabilmesi için iyi bir şekilde test edilmesi, ondan sonra da bakımının yapılabilir olması için, en başta, kolay okunur ve anlaşılır olması, iyi belgelendirilme yapılması gereklidir. Bu nedenle, geliştirmedeki teknik kolaylıklar yanında, dil seçiminin bakıma olan etkisi de unutulmamalıdır.

11.1.6. Yeni Diller

Günümüzde kullanılan bilgisayar sistemlerinin bilgi işleme ve zaman gereksinimleri tek işlemcili mimarilerin sınırlarını aştıkça ilave gerçek zamanlılık yetenekleriyle donatılmış dağıtık bilgi işleme yöntemleri daha çekici hale gelmeye başlamıştır. Gerçek zamanlı uygulamalar ve dağıtık bilgi işleme gibi birçok alanda güvenilirlik, doğruluk, sağlamlık, tasarım, geliştirme, test ve bakım kolaylığı göz önüne alındığında geleneksel programlama dillerinin yetersiz kaldığı görülmektedir. Nesneye yönelim kavramı yazılım modülleri için yüksek düzeyli bir soyutlama sağlayarak yazılımların tasarımını ve geliştirmesini kolaylaştırmaktadır. Fakat, klasik nesneye yönelik programlama dilleri ne dağıtık programlamayı ne de gerçek zamanlılığı doğrudan destekleyememekte, bu gereksinimleri çeşitli platformlar için özel olarak hazırlanmış kütüphanelerle karşılamaktadırlar. Çeşitli araştırmalarla halen üzerinde çalışılan dağıtık programlamayı kolaylaştıran, gerçek zamanlılığı destekleyen yeni diller geliştirilmektedir. Bunların yanında, İnternet uygulamalarını daha kolay geliştirmeyi amaçlayan, özellikle ASCII metin üzerinde çok kuvvetli işler yapabilen özel diller de geliştirilmekte, hatta HTML, XML gibi diller oldukça yaygınlaşmaktadır. Bir de belirli yazılım firmalarının kendi mimarilerini kullanıcılar arasında daha da yaygınlaştırmak amacıyla geliştirdikleri yüksek düzey diller vardır. Bu tür diller kullanımı düşünüldüğünde, mutlaka taşınabilirliğin göz önüne alınması gereklidir.

11.2. Kod Çevrim İşlemi (11.2)

Programlama dillerinin en önemli özelliklerinden biri de insan ile makine arasında ne tür bir geçiş sağladığıdır. İnsanlar normal karakterlerle gösterilen yazı ve sayıları anlayabilirler. Ancak bir bilgisayar yalnızca 0 ve 1 değerlerini alabilen ikili düzenden, yani "bit"lerden anlamaktadır. Dolayısıyla arada iletişimi sağlayacak bir çevirmene gereksinim vardır [12].

Normal olarak bir program ASCII biçimindedir. ASCII, insanların kullandıkları simge, harf ve rakamlar yerine onların sayısal karşılıklarını kullanır. Bu kodlar 0 ile 255 arasındadır. Herbir karakter bir sekizli (byte) ile gösterilir. Günümüzün gereksinimleri değerlendirildiğinde bu kadar karakterin yetmediği görülmüş ve iki sekizli ile gösterilen yeni ve daha evrensel bir kod sistemi de oluşturulmuştur. Halen birçok sistem, standart ASCII kullanmaktadır.

11.2.1. Derleyiciler

Normal metin halinde yazılmış program içeren dosyalara *kaynak kod* adı verilmektedir. *Derleyici* (compiler) bu dosyada bulunan metni oluşturan ASCII karakterlerini, dilin sözdizim kurallarına uygun olarak okur, hata yoksa bunları bilgisayarın anlayabileceği komutlardan oluşan makine diline (*Assembly*) çevirir. Bu komutları da bilgisayarın işlemcisinin tanıyabilmesi için *nesne koduna* (object code) çevirir.

Bir projenin kaynak kodu birden çok dosyadan oluşabilir veya çeşitli kütüphanelere ait kod parçalarını da kullanıyor olabilir. Bu durumda herbir dosya bilgisayar donanımı

ve işletim sistemine uygun bir derleyici ile ayrı ayrı derlenir ve meydana gelen nesne kodları ile durağan ya da dinamik kütüphaneler *bağlayıcı* (linker) kullanarak bağlanır. Bu işlem sonunda *yürütülebilir kod* (executable) elde edilir.

Derleyici bütün kaynak kodu okur ve dilin sözdizim kurallarını uygular. İyi bir derleyici, kaynak kodda bir hata varsa tam yerini belirleyerek kodlayıcıyı açık bir şekilde uyarır. Ancak, derleyici kaynak kodundan olabildiğince anlamlı bir sonuç çıkarmak için uğraştığından, bir hatanın asıl yapıldığı yerden daha ileride ortaya çıkması ve ilgisiz başka hataların da görülmesi olasıdır. Kaynak kod bir kere başarıyla derlendikten ve bağlandıktan sonra derleyici ile bir ilişkisi kalmaz; çalışması için sistemde bunların bulunması da gerekmez. Ancak, derleyicinin ve bağlayıcının yazılımın kullanılacağı işlemci mimarisine uygun olması gerektiği unutulmamalıdır.

Derleyicinin olumlu bir yanı da kod güvenliğini arttırmasıdır. Kaynak kodun saklandığı ASCII dosyası kolayca açılır ve okunabilir, hatta değiştirilebilir. Fakat, derlenmiş ve yürütülebilir hale getirilmiş bir yazılımı açmak ve değiştirmek o kadar kolay değildir.

Derleyicilerle birlikte kullanılan önemli terimlerden biri *derleme anı* diğeri de *yürütme anı*dır. Derleme anı, kaynak kodun derlenmesi işlemi sırasında meydana gelen olaylar için, yürütme anı da yürütülebilir durumdaki kodun işlemci üzerinde koşturulması sırasında meydana gelen olaylar için kullanılır.

11.2.2. Yorumlayıcılar

Bazı yazılımlar *yorumlayıcı* (interpreter) adı verilen özel altyapılarla birlikte kullanılırlar. Standart BASIC, LISP, PROLOG ve JAVA birer örnektir. Bir yazılım çalıştırılmadan önce yorumlayıcının bilgisayara yüklenmesi ve onun içinden de kaynak kodun yüklenerek çalıştırılması gerekir. Yorumlayıcı kaynak kodun satırlarını birer birer okuyarak işlemci komutları haline dönüştürür ve bu komutları uygular. Bu nedenle derlenmiş koda göre daha yavaştır. Yorumlayıcının kodun tamamını okumamasından dolayı, bir yazılımın tamamen doğru çalıştığını anlayabilmek için her parçasının çalıştırılarak yorumlayıcıdan geçmesi sağlanmalıdır. Özellikle büyük projelerde yorumlayıcı ile geliştirilen yazılımlarda hata ayıklaması oldukça güçtür. Günümüzde, bu tür dilleri belirli işlemciler için doğrudan çalıştırmaya yarayabilecek derleyiciler de geliştirilmiştir. Bu şekilde, klasik diller gibi nesne kodu da üretilebilmektedir.

11.2.3. Geliştirme Ortamı

Yazılım geliştirmek için ilk sahip olunması gereken, uygun bir geliştirme ortamıdır. *Ana sistem* olarak da adlandırılan bu sistemin yanında geliştirilen yazılımın test edilmesi için bir *test sistemi* ve gerçek sistem yani *hedef sistem* bulunmalıdır.

Ana sistem tüm geliştirme etkinliklerinin yürütüldüğü ve bazen de testlerin yapıldığı bir ya da daha fazla bilgisayardan oluşan bir sistemdir. Profesyonel anlamda bir ana sistemde aşağıdaki öğeler bulunmalıdır:

- **Sunucu**
  Ana sistem bir ağ şeklinde ise yeterli özelliklere sahip bir bilgisayar dosya sunucusu olarak kullanılmalıdır. Sunucu üzerinde herbir proje için yeteri miktarda alan ayrılmalı, farklı projelerin kaynak kod ve yazılım geliştirme dosyaları birbirlerine karıştırılmamalıdır.
- **Çalışma alanları**
  Herbir geliştirici için uygun büyüklükte ayrı birer çalışma alanı ayrılmalı, ortak ve paylaşılır alanlar tanımlanmalı, erişim hakları düzenlenmelidir. Kişisel çalışma alanları herbir proje için ayrı ayrı tutulmalıdır. Eğer ortak geliştirme yapılıyorsa, kişisel dizinler yerine proje dizinleri kullanılmalıdır.
- **Proje alanları**
  Her proje için ayrı ayrı geliştirme ve üretim dizinleri tutulmalıdır. Geliştirme dizinleri geliştiricilerin kaynak kod dosyalarını sürekli olarak değiştirdikleri, derlemelerin yapıldığı dizinlerdir. Genellikle ana sistemde testler yapılabiliyorsa böyle bir dizin kullanılır. Üretim dizinleri yalnızca sistem yöneticisinin erişebildiği, geliştirilmesi tamamlanmış ve test sisteminde sınanmaya hazır hale gelmiş yazılım birimlerinin bulunduğu dizinlerdir. Test sistemi veya hedef sistem ayrı yapıdalarsa bu dizinlere farklı derleyicilerle üretilen kodlar konur.
- **Metin düzenleyici**
  Geliştirme ortamında en az iki tür metin düzenleyici kullanılmalıdır. Bunlardan biri kod yazımı için çeşitli özelliklere sahip metin tabanlı kaynak kod düzenleyicisi, diğeri de belgelendirmede kullanılacak ofis türü uygulamalardır.
- **Derleyici**
  Geliştirme için en temel araç derleyicidir. Seçilen programlama diline, bilgisayar donanımı ve işletim sistemine göre derleyiciler farklıdırlar. Eğer derlenen yazılım aynı ana sistem üzerinde test edilecek ve yine aynı türde bir hedef sistemde kullanılacaksa tek bir derleyici yeterli olacaktır. Hedef sistem farklı ise, ayrı bir derleyici daha kullanılmalı, ana sistem testleri tamamlanan yazılım bu derleyiciden (çapraz derleyici) geçirilerek hedef sisteme yüklenmelidir. Ana sistemde test yapılmayacaksa, yalnızca hedef sistem için uygun derleyici kullanılmalıdır. İşletim sistemine göre farklılık gösterse de çoğu zaman derleyici yazılım paketleri bağlayıcıları (linker) da içerirler.
- **Kütüphaneler**
  Yazılım geliştirme ortamında tekrar kullanımı destekleyerek zaman tasarrufu sağlayan önemli bir etken kütüphane kullanımıdır. Daha önce geliştirilen ve tekrar kullanımı mümkün olan çeşitli modüller, kod parçaları, derleyici ile birlikte gelen standart kütüphaneler geliştiricilerin rahatça ulaşabileceği ortak bir alanda tutulmalıdır. Herbir proje için en çok kullanılan ortak bileşenleri tutmak üzere az bir adet kütüphane oluşturulmalı, standart bir şekilde derlenerek düzenleşim yönetimi uygulanmalı ve bakımı yapılmalıdır.

- **Düzenleşim yönetim sistemi**
  İdeal bir geliştirme sisteminde mutlaka bir düzenleşim yönetim sistemi bulunmalıdır. Eğer özel bir düzenleşim aracı yoksa, işletim sisteminin sağladığı çeşitli kısıtlayıcılık özellikleri ve toplu iş yürütme dosyaları (script) kullanılarak basit bir sistem oluşturulmalıdır.
- **Yükleme araçları**
  Geliştirilen yazılım birimlerini test sistemi veya hedef sisteme yüklemek üzere, üretim dizinlerinde bulunan yazılım birimlerini ağ üzerinden aktarmaya veya herhangi bir taşıyıcı ortam üzerine yazmaya yarayan çeşitli araçlar kullanılır. Bunlar basit birer toplu iş dosyası olabileceği gibi bir veritabanı yönetim sistemine bağlı olarak çalışan özel araçlar da olabilir.
- **Yazılım mühendisliği araçları**
  Günümüzde pek çok yazılım geliştirme aracı bulunmaktadır. Bunlar yardımıyla, beli bir yazılım mühendisliği yöntembilimi kullanılarak çözümleme ve tasarım yapılabilir, hatta otomatik olarak kaynak kod üretilebilir.
- **Destekleyici araçlar**
  Bazı geliştirme ortamları tümleşik bir yapıda olup işlevleri yerine getirebilirler. Ayrık yapıdaki sistemlerde, temel geliştirme öğelerine ek olarak, hata ayıklayıcılar (debugger), arındırıcılar (purifier), test programları, metin düzenleyiciler, resim düzenleyiciler, ofis uygulamaları bulunabilir.

Geliştirilen yazılımın ilk testleri mümkünse ana sistem üzerinde yapılmalıdır. Bu şekilde hedef sistem meşgul etmeden küçük hataların bulunması sağlanabilir. Hedef sistemin bir benzeri olarak bir test sistemi kullanılabiliyorsa bu da yazılım geliştirme işleminin daha sağlıklı yürütülmesine yardımcı olur. Test sistemi hem donanım hem de yüklü işletim sistemi ve yazılımlar bakımından hedef sistem ile aynı ya da yakın benzerlikte olmalıdır. Hedef sistem ise tüm yazılımın gerçekten üzerinde koşacağı ana sistemdir. Bazen ana, test ve hedef sistemin üçü aynı geliştirme yerinde oluşturulabilir. Bazen de hedef sistemi kullanmak son aşamaya kadar mümkün olmayabilir. Örneğin, bir hidroelektrik santralı denetleyecek hedef sistem, geliştirme yerinden çok uzaklarda olabilir. O takdirde, onun bir benzeri olan test sistemi kullanılmak zorundadır.

8. Hafta
2. Grup
SON