**BLM 426 YAZILIM MÜHENDİSLİĞİ**

**BAHAR 2016**

**Yrd. Doç. Dr. Nesrin AYDIN ATASOY**

**6. HAFTA: YAZILIM TASARIMI**

Tasarım, herhangi bir mühendislik ürünü geliştirme sürecindeki ilk adım sayılabilir. Tasarımcının amacı, geliştirilecek bir ürünün ilk modelini veya gösterimini ortaya çıkarmaktır. Tasarım genel olarak deneyim ve bilgi birikimiyle desteklenen çeşitli kurallarla yapılır. İnşaat, makine, mekanik düzenek veya elektronik tasarımlar için çeşitli hesaplamalar ve yardımcı araçlar kullanılabilir. Bilgisayar yazılımının tasarımı ise daha yeni bir geçmişe sahiptir. Çeşitli geliştirme teknikleri, tanımlama ve tasarım yöntemleri bulunsa da yazılım mühendisliği hala bir “sanat” niteliğindedir.

Bir kodlayıcı, ne kadar iyi olursa olsun, bir tasarım yapıp onu yeterli şekilde yazılı hale getirmedikten sonra verimli bir geliştirme yapamaz. En küçük kod parçaları için dahi olsa mutlaka önce tasarım yapılmalı, ondan sonra kod yazımına geçilmelidir. Hiçbir bilgisayar programı doğrudan kod yazma ile başlanmamalıdır. Zira, yazılım tasarımı, bir binanın temeline benzetilebilir. Yeteri kadar sağlam olmayan bir temel üzerine plansızca inşa edilen katlardan oluşan bir binanın depreme dayanıklı olması da beklenemez. Ayrıca üzerine başka katlar çıkmak da mümkün olmaz. Yazılım tasarımı konusunda genel olarak yazılım tasarımı hakkında temel bilgiler verilecek, basamaklar, yöntemler, kurallar ve çeşitli özel durumlar anlatılacaktır.

# TASARIM AŞAMASI

Genelde bir bütün olarak düşünülmesine rağmen yazılım tasarım aşaması adımlar halinde gerçekleştirilir. En önemli adımlardan birisi veri tasarımıdır, çözümleme sırasında toplanan bilgilerin ve bilgi yapılarının yazılımda kullanılacak veri yapılarına dönüştürülmesini içerir. Daha sonra gelen mimari tasarımı, yazılım birimlerinin yapısal parçalarını, birbirleriyle ilişkilerini tanımlar. Yordamsal tasarım, yazılımı oluşturan yapısal birimler yordam ve fonksiyonlar haline dönüştürülür. Arayüz tasarımı da insan-makine etkileşiminin şeklini, alt sistemlerle olan ara yüzlerin ayrıntılarını içerir. Tüm bunlar bir belgede toplanır, değerlendirilir ve sonra da kodlama aşamasına geçilir. Tasarım, yazılımın testine kadar her şeyi etkilediğinden nitelik unsurunun öne çıktığı ilk aşama olma özelliğini taşımaktadır.

Yazılım gelişim çevrimi içinde yer alan ve tekrarlanarak yapılan tasarımla ilgili çeşitli işlemler vardır. Bunlar arasında projenin bir bütün olarak tasarlanması, yani sistem tasarımının yapılması, standart yazılım birimlerinin belirlenmesi ve bu uygulama için hazırlanması, yeni yazılım birimlerinin oluşturulması, bu birimlerin içyapılarının belirlenmesi, birimleri bir araya getirerek sistemin oluşturulması sayılabilir.

Tasarımın birinci amacı her zaman basitlik olmalıdır. Çünkü anlaşılır ve basit bir tasarım hem kodlamada hem de sonraki değişikliklerde kolaylık sağlar. İleride gereksinim duyulabilecek iyileştirme, genişleme, taşınabilirlik gibi bazı durumların önceden sezilememesi halinde, sistemin bu tür değişime açık tasarım ve gerçekleştirimin basit olması ile gerçekleşebilir. Sistemi öyle tasarlanmalıdır ki, bir dizi değişiklik yapılsa bile sistem tasarımı hala basit kalabilmelidir. Bunun için, değişiklik yapılması olası olan kısımlara özel dikkat gösterilmeli, veri yapılarında esneklik sağlanmalı, programlama dilinin sağladığı kolaylıklar göz önüne alınmalıdır. Yazılım tasarımı sırasında sürekli olarak gözetilmesi gereken temel ilkelerden en önemli gördüğümüz üçü şunlardır:

**Soyutlama (abstraction),** denetimi ve anlaşılabilirliği artırmak üzere en az ayrıntı ile işlem yapmaktır. Bu amaçla, yazılım isterlerini gruplayarak, birimler ve modüller oluşturulur, aralarındaki ilişkinin en aza indirgenmesine çalışılır. Birbirlerinden soyutlananan modüllerde bulunan yordam ve veriler üzerinde erişim kısıtlamaları sağlanarak bilgi güvenliği artırılır.

**Bilgi gizleme (information hiding),** modüllerin iç yapılarını diğerlerinden gizlemek, bu şekilde karmaşıklığı engellemek ve soyutlamayı artırmaktır.

**Kapsama (encapsulation),** tüm isterlerin eksiksiz olarak karşılanması amacıyla yordam ve verilerin denetim altına alınmasıdır.

Tasarım nitelikleri olarak adlandırılan aşağıda sıraladığımız bu özelliklerin yazılım tasarımı yapılırken sürekli göz önünde bulundurulması gereklidir:

* İsterlerin izlenebilirliği olmalıdır.
* Geliştirilen birimin kodunun ve testlerinin izlenebilirliği olmalıdır.
* Programlama dilinden olabildiğince bağımsız olmalıdır.
* İşlevselliği, başarımı ve güvenirliği yüksek bir ürün oluşturulmalıdır.
* Yürütme sırasında oluşabilecek hataların ilgili iş sürecini aksatmayacak şekilde kotarılması sağlanmalıdır.
* Öğrenmesi ve kullanımı kolay bir ürünü hedeflemelidir.
* Tekrar kullanılabilir olmalıdır.
* Bir ürün ailesine temel oluşturabilmelidir.
* Kolay anlaşılmalıdır.
* Gerektiğinde kolaylıkla değiştirilebilmelidir.
* Kurumsal tasarım standartlarına uygun olmalıdır.
* Diğer tasarımlarla birleştirilebilmesi mümkün olmalıdır.

Yazılım projelerinde tasarım, **projenin büyüklüğüne göre, yazılım tasarım uzmanları tarafından yapılır. Personel yeterliliğine, deneyime ve projenin karmaşıklığına bağlı olarak sistem çözümleyicisi veya proje yöneticisi hatta kodlayıcılar tarafından da yapıldığı görülür**. Bazen de, büyük bir sistem bileşenlere ayrılır ve her bir bileşenin çözümlenmesi, tasarımı ve gerçekleştirilmesi ayrı gruplar tarafından yapılır.

# YAZILIM TASARIM SÜRECİ

Yazılım geliştirme sürecinin aşamalarından ilki olan **isterler çözümlenmesi daha kuramsal iken, tasarım, kodlama ve test daha tekniktir**. Tasarım aşaması bir tür süreç şeklindedir. Bu süreç sonunda ortaya çıkan tasarım, kodlamanın ve testin temeli olduğu için mutlaka yeterli zaman ayrılması gereklidir.

Tasarımın niteliğini değerlendirebilmek için iyi tasarım kıstaslarının belirlenmesi gereklidir. Bu kıstaslar genellikle **Yazılım Geliştirme Planı’nda** kullanılan standartlarla birlikte belirtilir.

Yazılım tasarımı sürecinde ve tanımlamalarda rehber olarak aşağıdaki standartlar kullanılabilir:

IEEE 1016.1-1993, IEEE Guide to Software Design Descriptions

IEEE 1016-1998, IEEE Recommended Practice for Software Design Descrip­tions

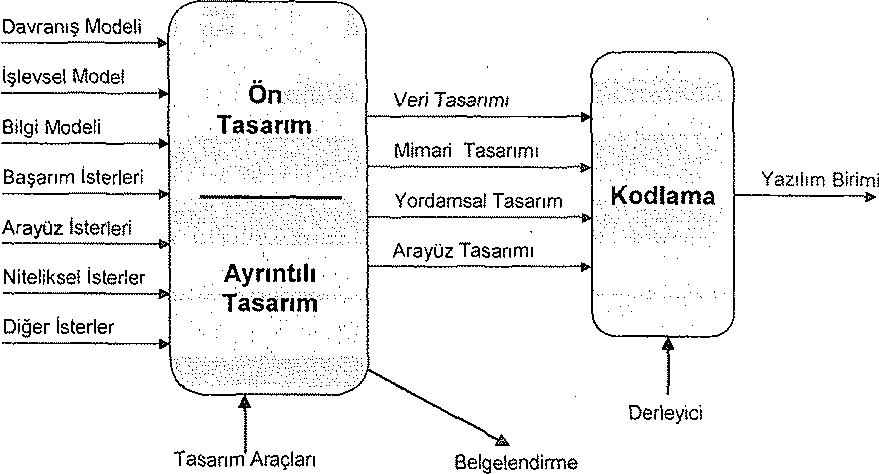
IEEE/EIA 12207.1, Guide for Information Technology - Software life Cycle Processes - Life Cycle Data

Tasarım için çeşitli yazılım geliştirme yardımcı araçları kullanılabileceği gibi, çeşitli tasarım yöntemleri de kullanılabilir. **Sonuçta ortaya elektronik ya da basılı ortamda çeşitli belgeler çıkar.** Tasarım sırasında isterler yazılım geliştirmede kullanılacak ifadelere dönüştürülürler. **Teknik olarak, süreç başında, bu ifadeler ve gösterim tarzı ile yazılımın genel görünüşü oluşturulurken, süreç sonunda tasarım kaynak koda yakın bir hale gelir.** Yönetsel olarak bu süreç iki aşamada ele alınır:

**Ön Tasarım (Preliminary design):** İsterlerin veri ve mimari tasarımına dönüştürülmesidir.

**Ayrıntılı Tasarım (Detail design):** Veri ve mimari tasarımının ayrıntılı veri yapıları ile algoritmik gösterime dönüştürülmesidir.

Yazılım tasarımı, isterler çözümlemesi sonunda elde edilen bilgilerle gerçekleştirilir ve kodlamaya esas olacak **veri tasarımı, mimari tasarım, yordamsal tasarım ve arayüz tasarımı** yapılır.Bunlar şekil 1’ de görülmektedir. Bu arada gerekli belgeler de üretilir.



**Şekil 1.** Yönetsel süreç modeli.

## Veri Tasarımı

Veri yapıları ve modelleri, **birbirleriyle mantıksal olarak ilişkili verileri yönetilebilir şekilde bir arada tutmaya yararlar.** Bu yapılar ve modeller, veriler arasındaki hiyerarşik ilişkileri ve erişim yöntemlerini belirler. Yapıların düzenlenmesi ve karmaşıklık derecesi tamamen tasarımcı tarafından belirlenir. Tasarımcı, veriye erişim yöntemi, hız, etkinlik, büyüklük, işlev bakımından çözümlemesini yaparak en uygun veri tiplerini ve yapılarını belirler. Veri yapıları ve modelleri bu kitabın konusu dışında tutulduğu için yalnızca tanımlama olarak kısaca değinelim:

**Sayısal öğeler,** belirli bir temel tipten olup programlama dili ve donanıma göre değişiklik gösterebilirler. Örneğin C dilindeki tamsayı, bazı donanımlar için 32 bit, bazıları için 64 bit ile gösterilmektedir.

**Diziler** birden fazla aynı tür öğenin ardışık olarak sıralanmasıyla oluşur. Tek boyutlu olanlar bazen *vektör* adımı alırlar. Dizilerin çok sayıda boyuta sahip olduğu durumlarda *matris* oluşur.

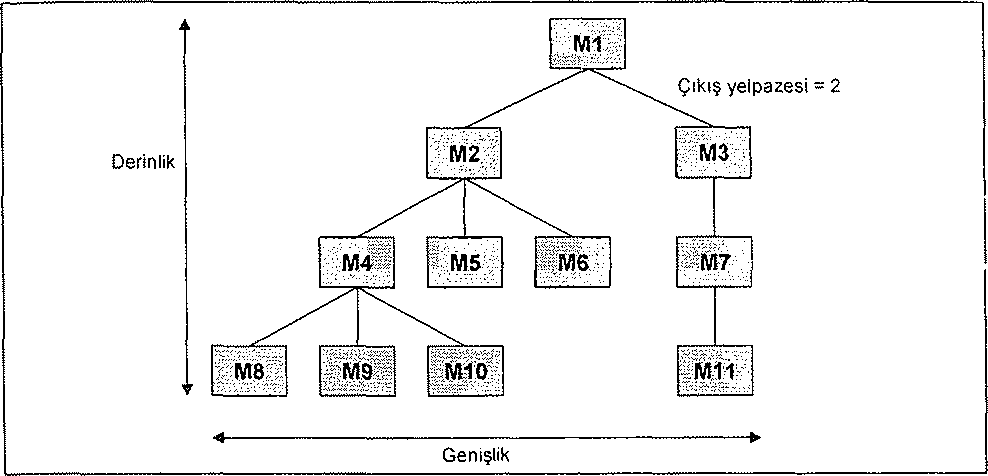
**Dinamik veri yapıları**, programın çalışması sırasında gereksinim duyuldukça bellekte oluşturulması ve yönetilmesi esasına dayanır. Bağlı listeler (linked list), ağaç (tree) ve eşlem (map) yapıları, buna örnektir.

**Veri yapılarının ve modellerinin** ne şekilde kullanılacağı da ayrı soyutlama kavram­larının kullanımını gerektirebilir. Örneğin, bir bağlı liste, ya "İlk giren ilk çıkar" (fırst- in-fırst-out) ilkesine dayalı bir yığın (stack), ya da rastgele erişimli bir depo olabilir. Tasarım sırasında benimsenen soyutlama derecesine göre, veri yapısının iç tasarımı belirtilmeden işlevselliği ön planda tutulabilir. Yani, tasarımcı, yalnızca bir yığın kul­lanmak istediğini belirtebilir, yığının gerçekleştirimini kodlayıcıya bırakabilir.

Veri yapısı ile veri modeli içiçe geçmiş iki ayrı kavramdır. Birisi verinin bellekte tutul­ması veya saklanmasıyla ilgilenirken diğeri veriler arasındaki ilişki ve bağıntılar konusuyla ilgilenir. Veriler üzerinde işlem yapacak olan algoritmalar da bu veri mo­dellerine göre tasarlanırlar.

## Mimari Tasarımı

**Uygulama yazılımı, bir problemin çözümünü çeşitli parçalara bölerek sağlayabilir.** Bu parçaların yazılımdaki karşılığımodüllerdir*.* **Modüllerin hiyerarşik ilişkilerini gös­teren yapıya uygulama yazılımı mimarisi denir.** Şekil 2’de modüllerden oluşan böyle bir yapılanma gösterilmektedir:



**Şekil 2.** Modül yapısı.

Yazılım içindeki modüller birer nesne olabileceği gibi, tasarım veya gerçekleştirim yönteminin özelliğine göre, birer program, birer paket, birer nesne veya birer yordam olabilirler. Yapının **çıkış yelpazesi, genişliği ve derinliği modüler yapı hakkındaki önemli** ölçütlerdir. Yazılım mimarisi seçiminde dikkate alınması gerekli nok­taları şu şekilde özetleyebiliriz:

### **Uygulama alanının özellikleri**

Yazılımın kullanılacağı alanın gereksinimlerine göre yazılım birimlerini fizik­sel olarak belirli donanım öğeleri üzerinde çalıştırmak gerekebilir. Sistemin merkezi ya da dağıtık olması, açık sistem olması, belirli bir amaçla kullanmak üzere tahsis edilmesi ya da gömülü sistem olması mimari seçimine etki eder.

### **Uygulama yazılımının karmaşıklık derecesi**

Basit uygulamalar, tek program içinde, her türlü arayüz ve bilgi işlemeyi kap­sayacak şekilde geliştirilebilirler. Daha karmaşık uygulamalarda, hem geliştir­me hem de yürütme bakımından yazılımı öğelere, öğeleri de birimlere bölmek daha kolay şekilde geliştirme, test ve bakım olanağı sağlar.

### **Kullanıcı arayüzü kısıtlamaları**

Bilgi işleme birimleri ile kullanıcı ara yüzünün farklı mimariye sahip işlem­cilerde çalışması gereken durumlar olabilir. Yüksek nitelikte grafik görüntü verebilen bilgisayarlar her amaç için uygun olmadıklarından bir ayrım yapmak gerekebilir. O takdirde, uygun bir iletişim altyapısı kullanılması zorunludur.

### **Taşınabilirlik**

Geliştirilen yazılımın sonradan başka işletim sistemi veya donanım ile kullanılmak üzere farklı ortamlara taşınması gerekiyorsa, katmanlı bir yaklaşımla, asıl yazılımı olası taşıma işinden etkilenmeyecek şekilde tasarlamak gerekir. Bu amaçla, yazılım mimarisi içine uygun katmanlar yerleştirilebileceği gibi ileti­şimin zorluklarını gidermek üzere bir ara katman mantığı da kullanılabilir.

## Yordamsal Tasarım

**Yordamlar (procedure, function) bilgi işlemeyi gerçekleştirmek üzere yazılım modülünün iç yapısında bulunurlar. Bir yordam, veri yapıları, döngüler, karşılaştırma­lar, dallanmalar yardımıyla tüm bilgi işleme özelliklerini tanımlamalıdır.** Yordamlar arasında da belirli bir yapı gözetilmesi zorunludur. Bir yordam içinde başka yordam­ların çağrılması, yazılım mühendisliği kuralları bakımından gereklidir. Bir modülün tüm işlevlerinin tek bir yordamla gerçekleştirilmesi olası olmadığı gibi, yordam içinde yordam çağırmada aşırıya kaçılmamalıdır.

Veri ve program yapılarının tasarımı tamamlandıktan sonra yordamsal tasarım başlar. **Yordamsal tasarım, modüllerin içyapılarındaki algoritmik ayrıntıların tamamlanmasıdır. Tasarım, konuşma diline yakın bir anlatımla yapılabileceği gibi çeşitli şekilsel gösterimlerle de yapılabilir.** Şimdi yordamsal tasarımın nasıl yapılacağım görelim:

### **Yapısal Programlama Gösterimi**

Yazılım tarihinin en eski tasarım yöntemlerinden biri işlevleri metinsel bir şekilde anlatmaktır. Bu anlatım için genellikle ingilizce kullanılmaktadır. *Program Tasarım Dili* (Programming Design Language) adı verilen bir dil de tasarım için tanımlan­mıştır. *Sözde kod* (Pseudo-code) adı verilen bu yöntemle, gerçek programlama dili yapıları­na benzer şekilde, ancak daha serbest bir sözdizimiyle her yapı ve her yordam tanım­lanır. Genellikle ardışık deyimler, koşullu dallanma ve döngüler kullanılır. Basit bir yordamın sözde kod ile tasarımı :

**PROCEDURE Periodic\_Processing**

**FOR EACH entry IN sensor\_list DO**

**Read heat sensor data into current\_temperature**

**IF current\_temperature > MAX\_TEMP THEN**

**CALL Alarm WİTH sensor\_ıd**

**ELSE**

**CALL Store\_Data WİTH sensor\_ıd, value**

**END İF**

**END DO**

**END**

Program tasarım dilleri genellikle ADA veya PASCAL gibi yüksek düzey bir dili andırırlar. Özel bir yazılım paketi gerektirmeksizin normal bir metin yazıcı kulla­nılabilir. Tasarım dili ile yazılmış metin dosyalarını grafiksel bir tasarım yöntemine dönüştürebilen araçlar da bulunmaktadır. Örneğin, bir yazılım paketi, bir tasarım kodunu tarayıp akış diyagramı üretebilir.

### **Grafiksel Gösterim**

Bazen bir resim birçok satırdan oluşan bir anlatımın yerine geçebilir. Bu gerçekten hareketle, çeşitli grafiksel gösterim yöntemleri bulunmuş, bu yöntemleri kullanan yazılım tasarım araçları geliştirilmiştir. Ancak, bir şeklin eksik ya da yanlış çizilmesi, okuyucunun gösterim simgelerini iyi bilmemesi sonucu tasarımı yanlış anlaması hatalı kodlamaya neden olabilir. Bu nedenle grafiksel gösterimlerin iyi öğrenilmesi ve iyi anlaşılması gereklidir. Şimdi, grafik tabanlı gösterim şekillerine biraz değinelim:

### **Yapısal çözümleme ve tasarım**

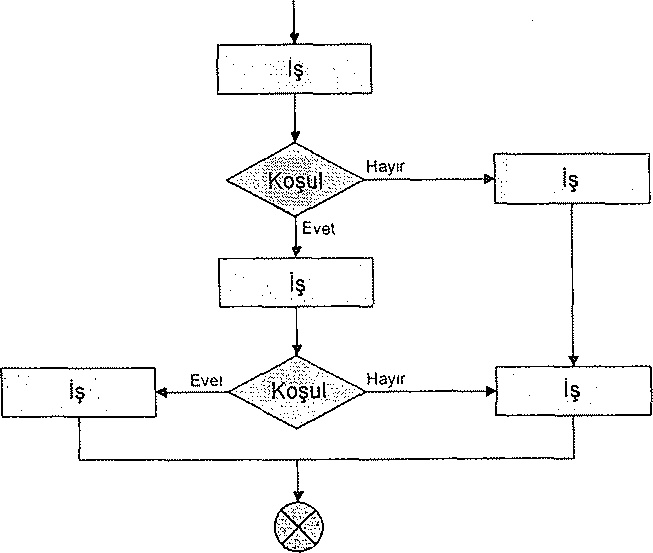
Yapısal çözümleme ve tasarım yapmak için veri akış diyagramları ve durum geçiş diyagramları kullanılır.

### **UML**

"Unifıed Modelling Language" (UML), nesneye yönelik çözümleme ve tasarımın hem metinsel hem de grafiksel olarak yapılabilmesine yardımcı olan uluslararası çevrelerce kabul edilmiş, standart ve yaygın bir tanımlama dilidir. Piyasada bu dili destekleyen çeşitli yazılım araçları ve tümleşik ortamlar bulun­maktadır.

### **Akış diyagramları**

En eski ve en yaygın program tasarım yöntemlerinden biriakış diyagramları(flow chart) kullanmaktır. Günümüzde bir program çok sayıda modülden veya yordamdan oluştuğundan, çok sayıda akış diyagramı kullanmak gerekebilir. Uzun veya karmaşık yordamlarda iç içe diyagramlar kullanmak da mümkün­dür. Hazırlanan diyagramlar arası geçişin sağlanması ve tutarlılığın korunması bir miktar zorluk getirir. Şekil 3’ te bir akış diyagramı örneği gösterilmektedir:



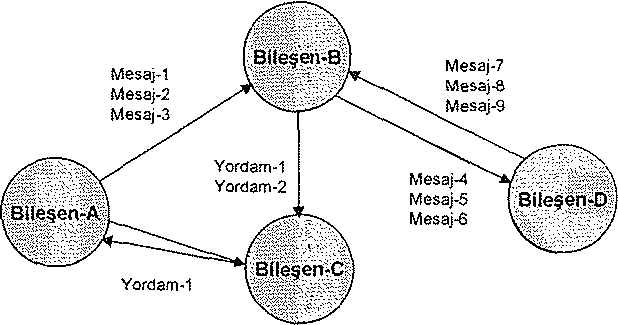
**Şekil 3.** Örnek akış diyagramı.

## Arayüz Tasarımı

**Modüler bir şekilde geliştirilen yazılımın çeşitli arayüzleri bulunur. Bunların bir kısmı içsel arayüzler bir kısmı da dışsal arayüzlerdir.** İçsel arayüzler genellikle yazılımın kendi iç öğeleri, bileşenleri ve birimleri arasındadır. Yazılımın dış dünya ile arayüzü ise başka sistemlerle olabileceği gibi etkileşimli sistemler için kullanıcıyla da olabilir. Tasarım, arayüzün bu özelliğine göre değişiklik gösterir.

### **Bileşen Arayüz Tasarımı**

**Büyük yazılımlar birkaç ana öğeden, her bir öğe birkaç bileşenden ya da birimden oluşabilir. Bu bileşenler arasında mutlaka tanımlı bir arayüz vardır. Bileşenler ya da birimler birer yürütülebilir program olabilecekleri gibi, bir program grubu da olabilir­ler.** Daha detaylı yapılarda ise bileşenler ayrı birer görevcik (thread), hatta birer yordam grubu olabilirler. Bağımsız birer süreç halinde geliştirilen bileşenler Şekil 4’ te görüldüğü gibi birbirleriyle işletim sisteminin sağladığı çeşitli alt düzey iletişim düzenekleriyle haberleşirler. Örneğin, Unix tabanlı işletim sistemleri için bağlantılar (socket), ileti kuyrukları (message queue), paylaşılır bellek parçaları (shared memory) ve semaforlar kullanılır. Bu düzenekler yardımıyla arayüzü oluşturan iletilerin veya uzaktan yordam çağrılarının (remote procedure cali) gerçekleştirimi yapılır.



**Şekil 4.** Örnek akış diyagramı.

Bileşenler arası arayüz tasarlarken dikkat edilmesi gerekenleri şöyle özetleyebiliriz:

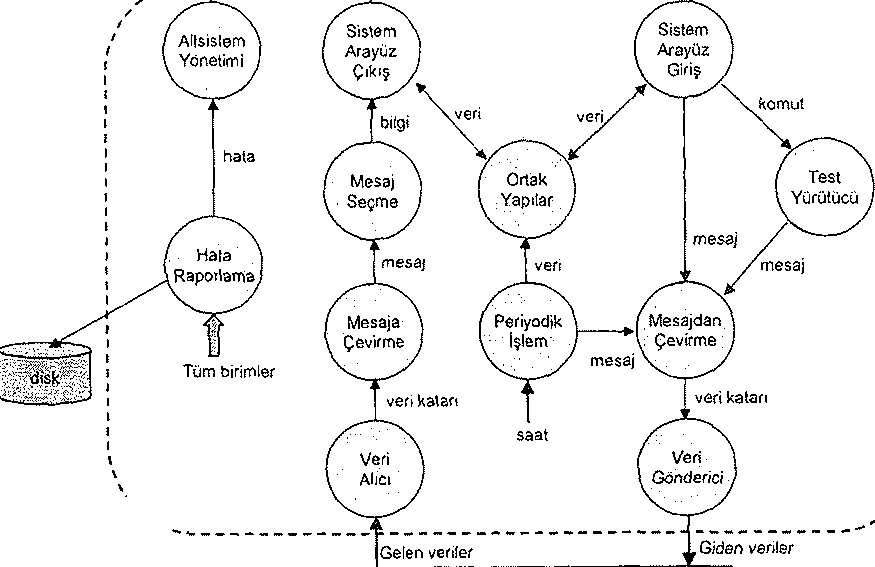
* Arayüz anlaşılır yapıya sahip ileti ya da yordamlardan oluşmalıdır.
* İleti tabanlı arayüzlerde başarım için ileti boyutu uygun şekilde ayarlanmalı, çok kısa ve çok uzun iletiler kullanılmamalıdır.
* Büyük miktarda veri aktarımı için ileti yerine ortak bir veri deposu ve aktarılacak verinin adresi kullanılmalıdır.
* Arayüzler belirli bir veri tipine bağımlı olmamalıdır.

### **Kullanıcı Arayüz Yazılımı Tasarımı**

Bilgisayar sistemlerinin hemen hemen hepsi insanların denetiminde çalışırlar. Bu nedenle de kullanımı kolay, etkili ve açık bir arayüze sahip olmalıdırlar. Sistemler genellikle standart bir bilgisayarın arayüzü ile denetlenirler. Yani, bir klavye ile girdi sağlanır ve ekranda çıktı görülür. Bunlar dışında, basılı çıktılar, çeşitli göstergeler çıktı için kullanılabileceği gibi çeşitli tuş takımları, işaretçi ve ayar düğmeleri de girdi için kullanılabilir. Önemli olan, uygulama alanında en uygun kullanım olanağım sağlaya­cak yöntemi bulmak, uygulamak ve en iyi etkileşimi sağlamaktır.

### **Sistem-Altsistem Arayüz Yazılımı Tasarımı**

Çoğu sistem birkaç altsistemi tümleştirerek daha büyük sistemler elde etmek üzere tasarlanır. Tümleştirme için altsistem arayüz yazılımları kullanılır. Bu yazılımlar de­netledikleri alt sistemlerin gerektirdiği iletişim protokollerini destekleyerek veri alış­verişi sağlarlar.



**Şekil 5.** Örnek akış diyagramı.

Arayüz yazılımları tümleştirilen altsistemin özelliğine göre çok çeşitli yapıda olabi­lirler. Yazılımın modüllerinin herbiri ayrı ayrı yürütülebilir şekilde, ayrık süreçler halinde geliştirilebileceği gibi, zaman kısıtlamaları elverdiği takdirde, paralel çalışan görevciklerden oluşan, bir tek süreç halinde de geliştirilebilir. Altsistemden gelen veriler Veri Alıcı modül ile arayüz donanımından okunur. Bir veri katarı şeklinde olan bu ham veri, Arayüz İsterleri Belirtimi'nde ya da Teknik Anlaşma'da anlatılan yönteme göre ve sınır kontrolü yapılarak tüm alanları anlaşılabilir iletiler haline dönüştürülür. İletinin adına göre bir dallanma yapılarak içindeki veriler işlenmek üzere alınır ve bilgi haline getirilir. Bu bilgiler ana sistemin kullandığı arayüze sistem girdisi ya da bir komutun yanıtı olarak gönderilir, aynı zamanda sistem içinde bulunan ortak veri yapıları da tazelenir.

**Ana sistemden gelen çıktılar ya da komutlar arayüz giriş biriminde işlenerek ortak yapıların da yardımıyla iletiler oluşturulur.** Bu iletiler protokole göre veri katarına çevrilerek arayüz donanımına yazılır. Ana sistemden gelen komutlara göre bir test çevrimi başlatmak ya da durum bilgisi almak mümkündür.

**Bazı altsistemler düzenli aralıklarla haberleşmek isteyebilirler.** Bu iletileri üretmek, zaman aşımlarını denetlemek ve atık toplamak üzere bilgisayar saatini kullanan bir periyodik işlem birimi kullanılabilir. Bu işlevler yanında, altsistem arayüz yazılımı tüm birimlerden gelen hata iletilerini toplayıp raporlama ve sonradan çözümleme amaçlı olarak kaydetme yeteneğine sahip olmalıdır. Her altsistem, ana sisteme kendi durumu hakkında sürekli rapor (heart-beat) vermelidir.

# TASARIM YÖNTEMLERİ

**İyi bir tasarım için belirli bir yöntemi seçip kurallarıyla uygulamak gereklidir. Hangi tasarım yöntemi veya aracı seçilirse seçilsin iyi kullanıldığı takdirde pek çok yarar sağlar. Ancak, eksik ya da hatalı kullanım geliştirilecek yazılımın da hatalı olmasına neden olur. Çünkü kodlama tasarıma dayanarak yapılır.** Projede resmi tasarım tek­niklerinin benimsenmesi bazı etkenlerden dolayı engelleniyor olabilir. Bunlar ara­sında, proje sürelerinin yeterli olmaması, tasarım araçlarına yeterince yatırım yapıla­maması, gerekli eğitimlerin alınamaması, üst yönetimden yeterli desteğin sağlanama­ması, yazılım geliştirme personelinin isteksizliği gibi nedenler sayılabilir. Yine de belirli bir yöntem seçilerek kurallarına göre uygulanması için çaba harcanmalıdır.

Yazılım tasarımında kullanılabilecek pek çok yöntem bulunmaktadır. Bu yöntemleri şu şekilde listeleyebiliriz:

* Böl ve yönet (divide-and-conquer)
* Tümevarım (bottom-up)
* Tümdengelim (top-down)
* Aşamalı ayrıntılandırma (stepwise refınement)
* Buluşsal yöntemler (heuristic methods)
* Deneme yanılma yaklaşımı (iterative approach)
* Artımlı yaklaşım (incremental approach)
* İşleve yönelik tasarım (function-orierıted design)
* Yapısal tasarım (structural design)
* Veri akışına yönelik tasarım (dataflow-oriented design)
* Nesneye yönelik tasarım (object-oriented design)
* Veriye yönelik tasarım (data-oriented design)

Tasarım oldukça geniş bir konu olduğu için, biz burada en yaygın olarak kabul edilen yöntemlerden birkaçının özelliklerine değinmek istiyoruz. Bunlar,**yapısal tasarım(structural design**),**veri akışına yönelik tasarım (data-flovv-oriented design),nesneye yönelik (object-oriented) tasarım ve veriye yönelik (data-oriented)tasarımdır.**

## Veri Akışına Yönelik Tasarım

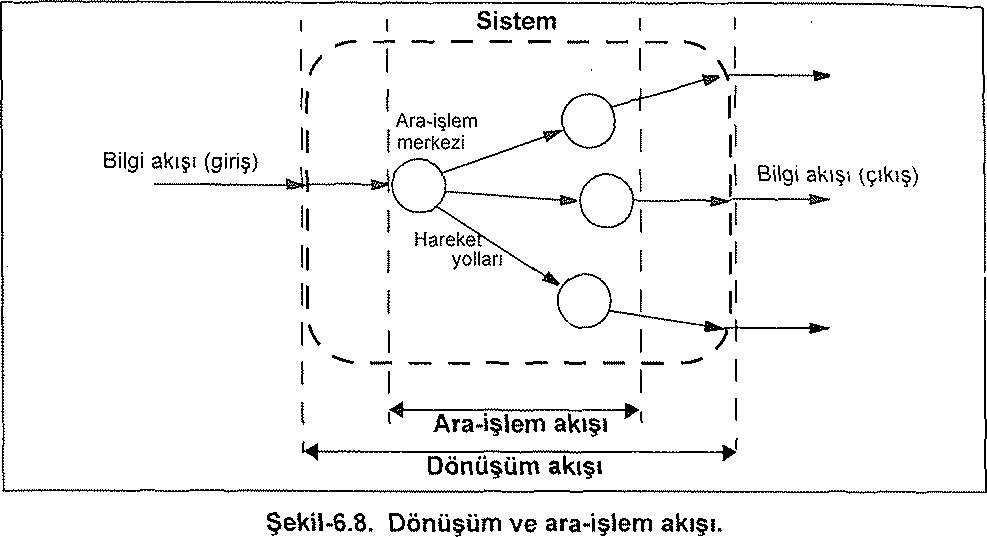
Yazılım isterleri çözümlemesinin bir parçasının bilginin çözümlenmesi olduğuna değinmiştik. Bilgisayar tabanlı bir sistemde, bilgi, belirli bir şekilde sisteme girer, çeşitli aşamalarda değişikliğe uğrar ve sistemden çıkar. **Yazılım tasarımında bu bilgi akışı dikkate alınarakveri akış diyagramları (data flow diagram) kullanılır**. Bu diyag­ramlara daha önce yapısal çözümleme kısmında değinmiştik. **Veri akışına yönelik (data flow-oriented) tasarımda, verilerin değişim şekilleri program yapısına uyarlanır.** Bu yönteme yapısal çözümlemeye benzediği için kimi zaman yapısal tasarım (structured design) adı da verilir. Bu yöntem modüler yaklaşım, yukarıdan aşağı tasarım modeli ve yapısal programlama ile birleştirilerek oluşturulmuştur.

Veri akışına yönelik yöntemin çok çeşitli uygulama alanları vardır. Aslında, her türlü yazılım bir veri akış diyagramıyla gösterilebilir. Ancak, veri akışına yönelik yaklaşım, özellikle hiyerarşik veri yapılarının bulunmadığı ve bilgilerin ardışık olarak işlendiği yazılımlar için daha kullanışlıdır. Karmaşık sayısal çözümleme yazılımları, mühendislikle ilgili çeşitli yazılımlar ve kontrol sistemleri yazılımları örnek olarak verilebilir. Bu yöntemin biraz daha genişletilerek gerçek zamanlı ve kesme kontrollü uygulamalarda kullanılması sağlanmıştır, veritabanı sistemleri, uzman sistemler, nesneye yönelik arayüzlerin bulunduğu sistemlerde diğer yöntemlerin kullanılması daha uygun olur.

## Akış Türleri

Yazılım isterlerini veri akışına yönelik olarak tasarıma aktarabilmek için bilgi akışının program yapılarına dönüştürülmesi gereklidir. Bu amaçla bilgi akışının, sınırların, işleme şeklinin ve yapıların tanımlanması gereklidir.

Tanımlama için kullanılacak veri akış diyagramlarında gösterim şekli olarak Şekil 6’da verildiği gibi iki tür bilgi akışı yer alır:



**Şekil 6.** Dönüşüm ve ara işlem akışı.

## Dönüşüm (transform) akışı:

Her sisteme dış dünyadan bir giriş vardır. Girişler sistem içinde işlenir ve dış dünyaya çıkış şeklinde gönderilir. Her giriş bu merkezde ardışık bir sıra ile işlenir ve bir dönüşüm akışını oluşturur.

## Ara-işlem (transaction) akışı:

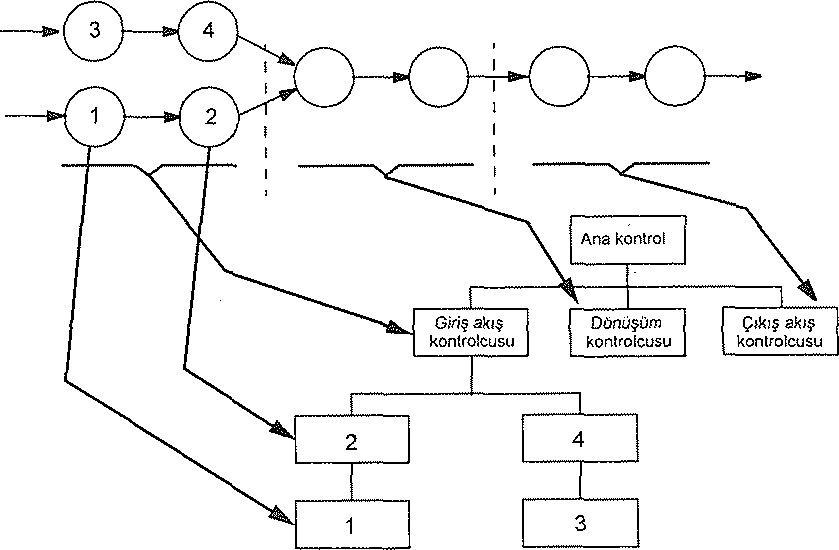
Giriş şeklinde gelen bir veri akışı, bir veri öğesine göre, çeşitli akış yollarından birine yönlendirilerek bir başka veri akışını tetikleyebilir. Bu şekilde bir ara- işlem akışı oluşur. Her akışta ara-işlem değerlendirilerek elde edilen değere göre bir hareket yolu seçilir.

## Tasarım Aşamaları

Veri akışına yönelik tasarım, akış diyagramının incelenmesiyle başlar. Önce akış tür­leri belirlenir ve akışın sınırları çizilir. Dönüşüm ve ara-işlem merkezleri belirlenir. Sı­nırların yerine göre dönüşümler, yani daire ile gösterilen süreçler, birer modül olarak program yapısı ile örtüşür hale getirilirler. Tanımlama ve örtüşmenin yapılışı dönüşüm ve ara-işlem çözümlemeleri ile gerçekleşir. Süreçlerle yapıların hassas bir şekilde örtüşmesi, yürütme denetiminin yukarıdan aşağıya doğru süreçlere dağıtılmasıyla yapılır. Bu aşamada modülerlik özelliğinin korunmasına dikkat edilir.

## Dönüşüm Çözümlemesi

Dönüşüm çözümlemesi, program yapısını oluşturmak üzere, isterler çözümlemesi sırasında yapılan işin tekrar gözden geçirilmesiyle başlar. Şekil 7’de görülen dönüşüm çözümlemesi ana basamaklarını şöyle sıralayabiliriz:



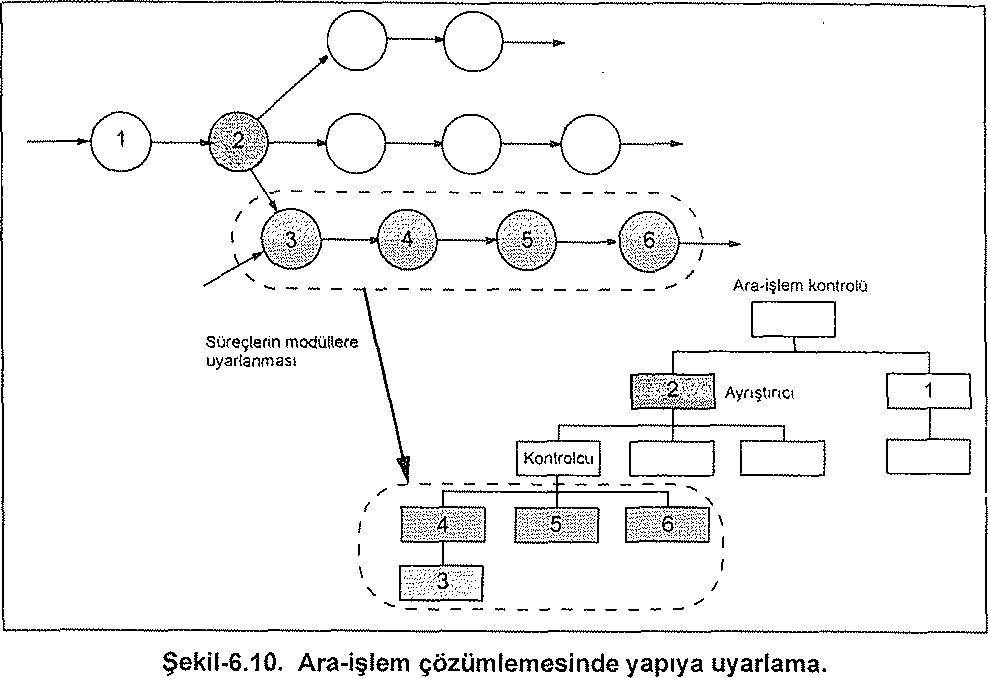
**Şekil 7.** Dönüşüm çözümlemesi basamakları.

Yazılım İsterleri Belirtimi'nde anlatılan sistem özellikleri ile Düzey 0 veri akış diyagramı kullanılarak bilgi akışları, yapı ve arayüzler incelenir.

Veri akış diyagramı daha ayrıntılı hale getirilerek Düzey 1 ve Düzey 2 veri akış diyagramları hazırlanır. Bunlar içinde dönüşüm akış özellikleri olanlar aranır. Bu maksatla, girişten çıkışa doğru giden akışlar ele alınır, ara-işlem dönüşümü olup olmadığına bakılır.

## Ara İşlem Çözümlemesi

Ara-işlem çözümlemesinde izlenen yol aşağı yukarı dönüşüm çözümlemesinde olduğu gibidir. Aralarındaki temel fark veri akış diyagramının Şekilde göste­rildiği şekilde program yapışma uydurulmasıdır. Ana basamaklar aşağıda olduğu gibidir:



**Şekil 8.** Ara işlem çözümlemesinde yapıya uyarlama.

Düzey 0 veri akış diyagramı, ya da diğer adıyla temel sistem modeli incelenir.

Veri akış diyagramları gözden geçirilir ve gerekiyorsa düzeltme yapılır.

Veri akış diyagramının neresinde dönüşüm, neresinde ara-işlem akışı olduğu araştırılır.

Bir giriş ve birden fazla çıkış olan süreçler ara-işlem merkezi olarak belirlenir ve her çıkış, yani her eylem yolu için akış özellikleri tanımlanır. Giriş yolu, yani ara-işlem akışı ile çıkış yolları için sınırlar çizilir.

Ara-işlem akışı, dağıtıcı (dispatcher) bir program yapısına uydurulur. Girişe göre yapılacak bir dallanmada kullanılacak modüller tanımlanmış olur.

Genel tasarım ilkeleri göz önüne alınarak program yapısı iyileştirilir. Modül­lerin bazıları birleştirilerek veya daha küçük başka parçalara ayrılarak program yapısı iyileştirilir.

## Modüler Tasarım

Veri akışına yönelik tasarımla oluşturulan program yapısı üzerinde bazı düzenlemeler yapılarak etkin birmodüler, yanibirimsel tasarım elde edilebilir. Sistem parçalara ayrıldıkça, her parçanın karmaşıklık derecesi azalacak, dolayısıyla da sistemin toplam karmaşıklığı düşecektir. Modül sayısının çok az olması yeterli soyutlama ve ayrıştırma sağlamaz; dolayısıyla da gerçekleştirimde yarar sağlamaz. Modül sayısının çok fazla olması da hem işgücünü hem de arayüzleri arttırır, tümleştirme güçlükleri oluşturur.

Modüllerin hiyerarşik yapısının gösterildiği sistem yapısında oluşan derinlik veya genişlik çok fazlaysa azaltılmalıdır. Bunun için, denetim ve karar verme yapıları yukarıya çekilmeli, yalnızca çağrılarak kullanılan yapılar alt düzeylerde tutulmalıdır.

## Tasarım Anlatımı

İyi bir mimari tasarım için dönüşüm ya da ara-işlem çözümlemesi yanında iyi bir anlatım da gereklidir. Bu amaçla yazılım tasarımının belgelendirmesi yapılır. Genel olarak Tasarım Tanımlaması (Design Description) adım taşıyan belge içinde, her modül için yapılan işin metinsel anlatımı, arayüzün tanımı, yerel ve evrensel veri yapılarının tanımı, bellek, işlemci ve zamansallık gibi kısıtlamaların belirtimi bulunur. Karar verme düzenekleri ve giriş/çıkışlar belirtilir.

Her yazılım geliştirme işlemi gibi bu ön tasarımın da bir gözden geçirmesi yapılır, bulunan eksiklikler giderilir, gerekirse iyileştirme yapılır. Bundan sonra da takip eden aşama olan ayrıntılı tasarıma geçilir.

**KAYNAKLAR**

1. Yazılım Mühendisliği; M. Erhan Sarıdoğan.