YAZILIM KRİZİ VE SSADM*

Hakan SARBANOĞLU

1. GİRİŞ

1980 li yıllar Türkiye'sinin en göze çarpan olgularından birisi de bilgisayarlı işlemlerin yaygınlaşmasıdır. Hemen hemen hergün basın ve yayın bültenlerinde " Bilgisayar alındı ", " Bilgisayar kuruldu ", " Bilgisayara yüklendi ", " Otomatize edildi ", " Bilgisayara geçildi " vb. haberlere rastlanmaktadır. Bu yaygınlaşmanın Kuzey Amerika ve Batı Avrupa ülkelerine göre biraz geciktiği de bilinen bir gerçektir. " Bilgisayar nedir? " " Bilgisayarcı kimdir? ", " Bilgisayarlaşma ne demektir? " sorularına karşılık alınan yanıtlar ise ne yazık ki ülkemizde bu konuda hala derin bir kavram kargaşasının yaşandığını göstermektedir /YÜCE 1986/. Bu sorulardan ilk ikisi değil ama üçüncüsü bu yazının odak noktasını oluşturmaktadır. " Bilgisayarlaşma nedir? ", " Nasıl bilgisayarlaşılır? " konusu belkide bir ülkenin geleceğine damgasını vuracak bu gelişme sürecinde en önemli noktayı oluşturmaktadır. Aslında ne " bir bilgisayar satın alınması ", ne " bilgilerin bilgisayara (?) kaydedilmesi ", ne de " Bilgisayar programı yazılması " bilgisayarlaşma demek değildir. Bu nedenle haber bültenlerinde veya basında haberlerin yukarıda örneklendiği biçimde değil de " Bilgisayar destekli yeni bir sistemin uygulanmasına geçildi " şeklinde verilmesi çok daha fazla sevindirici olurdu.

Burada sözü edilen sistem kavramı :

" Bir bütün oluşturacak biçimde birbiri ile ilişkili, birbirini etkileyen parçaların kümesi."

Veya ;

" Arzulanan amaçlar için bir takım görevleri tamamlamak üzere donanım, yazılım ve personelin karşılıklı ilişki içinde işlem yapacak organizasyonu " olarak tanımlanmaktadır.

Bu tanıma göre örneğin " Gülhane Askeri Tıp Akademisi Hasta Kayıt ve Takip Sistemi " dendiğinde; muayene ve tedavi için başvuran hastaların kayda alınmalarını, hastane içindeki bütün sevk ve işlemlerini, bunların sonuçlarını takip etmek ve çeşitli değerlendirmeler, istatistikler yapmak üzere kullanılan her türlü araç ve gereç, kayıt formları, işlem kuralları, talimatlar ve

---

(*) *Structured Systems Analysis and Design Method.*

bu alanda çalışmaları da kapsayan geniş bir kavram anlaşılır. Verilen bu örnekten de anlaşılacağı gibi hastahanede bu amaç için halen kurulu bulunan düzen ister bilgisayarlı isterse bilgisayarsız olsun yine bir sistemdir ve mevcut fiziksel sistem (bazen sadece mevcut sistem) olarak adlandırılır.

Mevcut fiziksel sistem belirli gereksinimleri karşılamaktadır ve eğer kurumda yeni gereksinimler ortaya çıkmışsa veya mevcut sistem aksıyorsa o zaman yeni bir sistem arayışı içine girilir. Bu arayış çoğunlukla kuruluşun kendi içinden doğar ve bu konuda çeşitli incelemeler, etüdler ortaya çıkmaya başlar. Yanlışlar da bu andan itibaren başlar. " Bir bilgisayar alıp bu işi ona yaptıralım " veya " Bir program yazıp bu işi bilgisayarda yaptıralım, düğmeye basınca bize herkesin bilgilerini versin " gibi sözler duyulmaya başlanır. Bunlar mutlaka hoş görülmesi gereken heyecanlardır. Ancak hiçbir zaman sistem değişikliğini öngören böylesi önemli süreçlerde karar almaya yeterli değildir. Bu aşamada en tehlikeli yaklaşım ise haklı olsalar dahi yöneticilerin her zaman alışılmış olan aceleciliğidir. Yıllar önce bilgisayarlaşmaya başlamış ülkelerde özellikle bu acelecilik yüzünden başarısızlıkla sonuçlanmış proje sayısının,başarıya ulaşanlardan çok daha fazla olduğu görülmüştür ve hala da görülmektedir. Tabii acelecilik tek neden değil, en kolay suçlanabilecek nedendir; ancak bugün " Bilgisayarlaşma " dan istenen sonuçların alınmayışının pekçok daha başka nedenleri de olduğu görülmektedir.

2. YAZILIM KRİZİ

Yazılım krizi terimi 1968 de Batı Almanya'da yapılan Yazılım Mühendisliği Konferansı'nda ilk olarak kullanıldığında o kadar dikkat çekmemişti /DIJKSTRA, 1968/. Ancak 1970'li yıllarda giderek daha sık takrarlanır oldu ve genel bir yazılım krizinin var olduğu herkesçe kabul edilir hale geldi. Basitçe açıklamak gerekirse yazılım krizi, yazılım sistemleri tesis etmenin başlangıçta tahmin edilenden çok daha güç gerçekleşmesi demektir. Yazılım sistemini gerçekleştirmek üzere uzun uzun programlar yazılmakta ancak sonuçta bir türlü bilgisayarlı sistemlerden olan beklentiler tam olarak karşılanamamaktadır. Bu durum özellikle büyük yazılım sistemleri için çok daha vahimdir. Yazılım sistemlerinin boyutları ele alındığında, büyük-küçük ayrımının da kaynağını oluşturan ilginç bir sonuçla karşılaşılır. bu sınırın altındakiler yani küçük yazılım sistemleri, tek bir kişi tarafından anlaşılabilen ve tasarımı,

gerçekleştirilmesi ile ilgili bütün detayları bir kişinin zihninde tutabilecek kadar az karmaşık olan sistemlerdir. Diğer taraftan büyük sistemler ise projenin bütün detaylarının tek bir kişi tarafından akılda tutulması ve izlenmesi imkânsız olan yazılım sistemleridir; dolayısıyla ekip çalışmasını, işbölümünü ve uzmanlaşmayı gerektirirler. İşin ilginç olan yönü, büyük sistemleri kurarken karşılaşılan sorunlar, küçük sistemlerin sorunlarının ölçeklendirilmesi demek değildir.

Bu konuda çokça kullanılan bir örnek vardır: /YOURDON, 1980/

8-10 m. genişlikli bir dere üzerinde kurulan bir köprü ile bir boğaz köprüsü arasında bir benzerlik kurulabilir ve " biri sadece diğerinin büyüğü, prensip aynı, birini inşa eden diğerini de inşa eder " iddiasında bulunulabilir. Oysa büyük köprü inşa etmenin bambaşka sorunlarının olduğu açıktır. İşte yazılım sistemleri de böyledir. Belli bir problemi çözmek üzere birkaç yüz satırlık bir program yazıp başarıya ulaşmış bir programcıdan bir büyük yazılım sistemini kurmasını (örneğin THY'nın rezervasyon ve bilet satış sistemlerini kurmasını veya bir savaş uçağının navigasyon sistemleri yazılımlarını hazırlamasını) beklemek büyük bir yanılgı olur. İşte uzun yıllar yaşanan ve hala da ortadan tamamen kalkmamış olan yazılım krizi büyük ümitlerle kurulan birçok yazılım sisteminin çökmesine neden olmuştur.

2.1. YAZILIM KRİZİNİN BULGULARI

Yazılım krizi çeşitli bulgularla ortaya çıkmaktadır. Bu bulgular şöyle sıralanmaktadır /BOOCH, 1983/.

a. Bilgisayara dayalı sistemler kullanıcı gereksinimlerini çoğunlukla karşılamamaktadır. (Cevap verme)

b. Yazılımlar sık sık başarısızlıkla sonuçlanmaktadır.(Güvenirlik)

c. Yazılımların maliyetleri önceden tahmin edilememekte veya tahmin edilen maliyetleri çok aşmaktadır. (Maliyet)

d. Yazılımların bakımı karmaşıktır. Maliyeti yüksektir ve hatalara yol açmaktadır. (Değiştirilebilirlik)

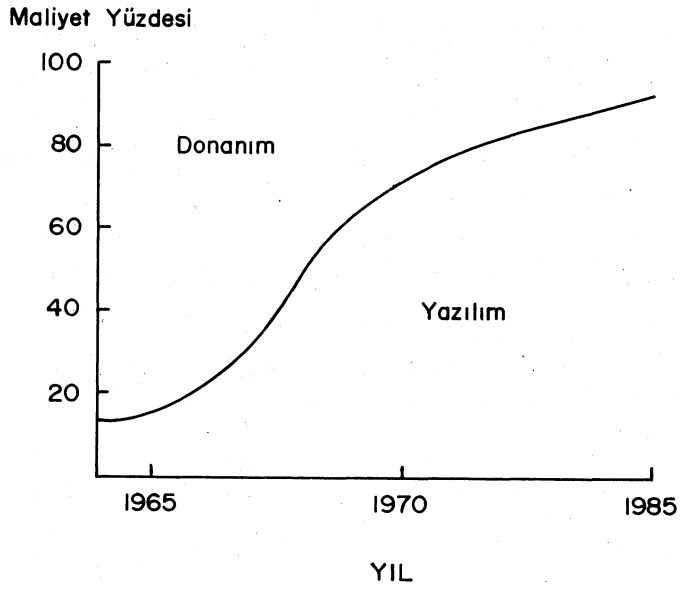
e. Hazırlanan yazılımlar genellikle geç tamamlanmakta ve söz verildiğinden daha az olanaklarla teslim edilmektedir. (Zamansızlık)

f. Programlar belli bir bilgisayara bağımlı olarak hazırlandığından başka bir tip bilgisayarda veya işletim sisteminde aynen çalıştırılması mümkün olmamaktadır. (Taşınabilirlik)

g. Yazılım geliştirme çabaları; işlem, zaman ve bellek gereksinimi gibi ilgilenilen kaynakların en uygun kullanılımı sonucunu vermemektedir.(Etkinlik

Yazılımlar aslında soyut ürünlerdir; tutulur, görülür tarafları yoktur. Bu nedenle yukarıda sayılan bulgular hakkında ancak çok az sayılabilir ölçüt vardır. En dikkat çekici ölçüt ise maliyettir.

Şekil-1 Amerikan Savunma Bakanlığınca yapılan donanım - yazılım harcamaları oranını göstermektedir. Harcamaların çok büyük bir oranının yazılıma kayması sayılan bulguların ne derece isabetli olduğunu gösteren açık bir göstergedir.



Şekil-1

2.2. YAZILIM KRİZİNİN NEDENLERİ

M.T.Devlin /DEVLIN, 1980/ yazılım krizinin nedenlerini şu maddeler altında toplamaktadır :

a. Kuruluşların yazılım geliştirme aşamalarını anlamada başarısız olmaları.

b. Yazılım mühendisliği konusunda eğitilmiş personel azlığı.

c. Günümüzde bilgisayarların " bir anda bir işlem yapma " prensibinin (Von Neuman Prensibinin) getirdiği kısıtlamalar.

d. Kuruluşların eski programlama dillerini ve alışkanlıklarını kullanmada ısrar etme eğilimi.

Bilgisayar dünyasında 1970 ve 1980'li yılların bu sorunlara çözümler bulmakla geçtiği söylenebilir. Aslında en önemli sorun insanın kendisinden kaynaklanan kısıtlamalardır. Çıplak elle bir çukur kazmak zordur, Urfa Tünelini kazmak ise imkânsızdır. Fakat Urfa Tüneli kazılmıştır ve bu işte pekçok araç, makine ve teknik kullanılmıştır. Aynı şey yazılım geliştirmede de geçirlidir. Yazılım mühendisleri karmaşık sorunların çözümü için birçok donanım (Bilgisayarlar, çevre birimleri), yazılım araçlarını ve tekniklerini (yapısal programlama tekniği, veri akış diagramları, nesneye yönelik tasarım vb.) kullanmak zorundadırlar.

Yukarıda sayılan sorunlardan ikincisi, yani personel sorunu hemen hemen her alanda geçerlidir. Ancak çok yeni bir disiplin olan yazılım mühendisliğinin sorunları oldukça ağırdır. Tanımı bile kamu oyunca belirsiz olan ve somut olmayan birtakım şeyler üreten yazılım mühendisliğinin henüz ortak bir eğitim standardı dahi yoktur. Nitelikli yazılım mühendisi talep ve arzı arasındaki uçurumun ise ülkemizde uzun süre etkisini göstereceği tahmin edilmektedir.

Sorunlardan üçüncüsü yani donanım sorunu aslında büyük bir kısıtlayacı değildir. Entegre devrelere dayalı 3 ncü kuşak bilgisayarların 1965 yılındaortaya çıkması ile donanımcılar yazılımcılara büyük bir işlem gücü sunmuşlardır. VLSI teknolojisine dayalı 4 ncü kuşak bilgisayarlar ise günümüzde bazı özel problemlerin dışında yeterli işlem gücünü vermektedirler. Bununla beraber Japonya ve Amerika başta olmak üzere 5 nci kuşak bilgisayarların yapımı konusunda çalışmalar sürmektedir.

Programlama dillerinde de büyük gelişmeler vardır. FORTRAN ve COBOL gibi eski dillerin yeni birtakım yazılım tasarım tekniklerini destekleyememesi üzerine özellikle C, PASCAL ve ADA gibi daha geniş yetenekleri olan diller geliştirilmiştir. Öte yandan 4 ncü kuşak programlama dilleri (4GL) ise programcıları problemin çözümünü bilgisayara aktarırken " nasıl yapması " gerektiğini kodlama derdinden kurtarmış ve sadece " ne yapması " gerektiğini kodlamayı yeterli kılmıştır.

Bu yazının bundan sonraki bölümlerinde asıl üzerinde durulacak olan; sorunlardan ilki yani yazılım geliştirme sorunu ve bu soruna karşılık geliştirilen yöntemlerden bir tanesidir. (SSADM).

3. SİSTEM GELİŞTİRME TEKNİKLERİ

Bilgisayar destekli bir sistem geliştirme, iki değişik durumda söz konusu olur; mevcut sistem vardır veya yoktur. Mevcut sistem manuel (bilgisayar desteksiz) ise bu durumda sistem geliştirme faaliyeti yeni sistem bilgisayar destekli olacağından "otomatize etmek" olarak adlandırılır. Halen uygulamada olan bir sistemin otomatize yeni bir sistemle değiştirilmesi genellikle "daha çok, daha hızlı, daha ucuz, daha doğru vb.) isteklerden kaynaklanır. Mevcut sistem yoksa, başka bir deyişle daha önce yapılmayan bir iş otomatize yeni bir sistemle yapılacaksa sistem geliştirme faaliyetinin karakteri değişik bir biçim alır.

Mevcut bir sistemin otomatize edilmesi veya yepyeni bir sistemin hazırlanması için izlenebilecek çok çeşitli yaklaşımlar vardır. Bu yöntemlerin genel gidişleri çoğunlukla aynıdır ve şu safhalardan geçer:

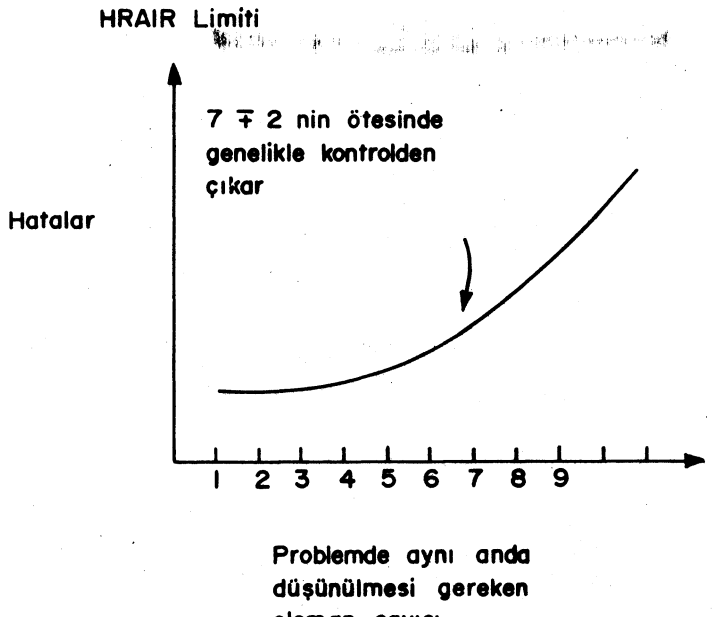
a. Fizibilite etüdü (Gerekli ise)
b. Sistem analiz
c. Sistem tasarımı
d. Sistem gerçekleştirme
e. Uygulama ve bakım

Bu safhalar sistem (veya yazılım sistemi) yaşam süreci olarak anılır.

Psikolog George Miller 1954'te yayınladığı ünlü araştırmasında /MILLER, 1956/ bir insanın bir anda işleyebileceği varlık sayısını kabaca yedi olarak açıklamaktadır. Yazılım sistemi geliştirme de, bir problem çözme faaliyeti olduğundan uygulanacak yazılım geliştirme tekniği için bu sınır çok önemli bir faktördür. Bu nedenle yazılım mühendisliğinin prensiplerinden birisi de sistemleri öylesine ayrıştırmaktır ki, her çözüm düzeyinde aynı anda uğraşılan varlık sayısı bu sınırı aşmasın. (Şekil-2)

Sistem birtakım alt modüllere ayrıştırıldığında bu modüller arasındaki tutarlılık önemli bir kriter olarak ortaya çıkar. Yukarıda sıralanan sistem geliştirme safhalarından ilk üçü (fizibilite, analiz, tasarım) için bugün yüzlerce yöntemin bulunduğu bilinmektedir. Özellikle tasarım yönünden birbirinden farklılaşan bu yöntemlerden"tutarlılık kriterini" sağlayanlar şu dört ana grupta toplanabilir: /BOOCH, 1983/

a. Tümden gelimci yapısal tasarım
b. Veri yapısı tasarımı
c. Parnas ayrıştırma yöntemi
d. Nesneye yönelik tasarım



Şekil-2

Bu tasarım yöntemlerini çok kısaca tanımlamak gerekirse:

Tümden gelimci yapısal tasarım, fonksiyonel bir yöntemdir. Bu yöntemde sistem ana işlem gruplarına ayrıştırılır. Ayrıştırma herbir işlem için bir modül elde edilene kadar sürdürülür. Bu tip tasarım yöntemi işleme yönelik ve yapısı "sıralı" olan sistemlere uygundur ve çok yaygındır /YOURDON,1979/.

Veri yapısı tasarımında ise işlemlerden çok veri yapıları tanımlanır ve işlemler (program birimleri) bu veri yapılarına göre kurulur. Daha çok COBOL tipi programlama dilleri ile uygulanır /JACKSON, 1972/.

Parnas ayrıştırma kriteri ise, sistemin her modülünün bir tasarım gizlemesine dayanır /PARNAS, 1972/.

Nesneye yönelik tasarım ise çok yeni bir yöntemdir. Bu yöntemde nesneler özellikleri ve işlemleri ile ayrı ayrı tanımlanır. Klasik yöntemlerde veriler işlem modüllerine gönderilirken nesneye yönelik sistemlerde her nesnenin kendi işlemleri ayrıca tanımlanır (metodlar) /COX, 1987/.

Yöntem ne olursa olsun bilgisayar destekli bir sistem geliştirmek üzere yapılan sistem analiz ve tasarım çalışmasının sonucunda; hangi kütüklerin hazırlanacağı, bu kütüklerde hangi verilerin ne şekilde kaydedileceği, hangi programların hazırlanacağı, tek tek bu programların yapacağı işler, yeni

sistemi kullanacak olan personelin tek tek fonksiyonları, yapılacak işlemlerle ilgili talimatlar, gerekli bilgisayar ve çevre birimleri, hatta kullanılacak ekran görüntülerinin şekli bile belirlenir.

Ancak bütün bunlar henüz kâğıt üzerindedir. Bu tasarımı gerçekleştirmek üzere donanımın sağlanması, programların ve kütüklerin hazırlanması ve testi gerektirmektedir. Belli bir projeye yönelik olarak donanımın sistem analiz ve tasarım çalışmalarından sonra temini en uygun zaman olmaktadır. Zira kullanıcının gerçek ihtiyaçları ve yeni kurulacak sistemin özellikleri ancak sistem analiz ve tasarım çalışmalarının sonucunda ortaya çıkmaktadır.

Uzun ve zahmetli bir faaliyet olan sistem analiz ve tasarım çalışmalarının bitiminde sadece kâğıt üzerinde sonuçların elde edilmesi, bu çalışmaların gereksiz olduğu ve elde edilen sonuçların zaten biliniyor olduğu iddiasını doğurabilir. Ancak bilinen gerçeklere dayalı olarak amaca yönelik en uygun sistemi ortaya çıkarmayı hedefleyen bu çalışmaları gözardı etmek büyük yanlışlıklara yol açabilmektedir.

Bundan sonraki bölümde sistem analiz ve tasarım çalışmalarına büyük bir önem verilen İngiltere'de yaygın olarak uygulanan bir yöntem açıklanacaktır.

4. YAPISAL SİSTEM ANALİZ VE TASARIM YÖNTEMİ (SSADM)

İleri düzeyde bilgisayarlaşmış olan İngiltere'de hükümetin bilgisayar politikasına yön vermek ve öte yandan tüm resmi kuruluşların bilgi işlem uygulamaları ile ilgili düzenlemeler yapmak üzere merkezi bir kuruluş bulunmaktadır. CCTA (Central Computer Technology Agency : Merkezi Bilgisayar Teknolojisi Kuruluşu) isimli bu kuruluş devlet sektöründeki bilgi işlem çalışmalarına yön vermenin yanısıra çeşitli araştırma ve eğitim faaliyetlerinde de bulunmaktadır.

İngiltere'de her kuruluşun bilgi işlem sistemi geliştirmek üzere değişik yollar izlemesinin birçok sakıncaları görülmüş ve bu arada bazı projelerin başarısızlıkla sonuçlandığının da görülmesi üzerine standart ve en uygun bir sistem analiz ve tasarım yönteminin geliştirilmesi fikri ortaya atılmıştır. Bunun üzerine CCTA öncelikle böyle bir yöntemden nelerin beklendiğini ayrıntılı biçimde tesbit etmiş ve bu tesbitlerini çeşitli üniversite ve kuruluşlarının görüşleri ile olgunlaştırmıştır. Böyle bir yöntem geliştirmek üzere bir yarışma açılmış ve katılan 57 teklif içinden seçilen birkaçı denenerek

daha da geliştirilmiştir. Sonuçta teklif olarak hazırlanan yöntemlerden biri en uygun olarak seçilmiş ve bu yönteme SSADM (Structured System Analysis and Design Method : Yapısal Sistem Analiz ve Tasarım Yöntemi) adı verilerek, tüm resmi kuruluşlarda bilgisayar destekli bir sistem geliştirilirken bu yöntemin izlenmesi öngörülmüştür. Bunu sağlamak üzere birçok resmi ve özel eğitim kuruluşunda SSADM kursları açılmış ve yöntem öğretilerek yaygınlaştırılmıştır. Bu gün Silâhlı Kuvvetler dahil olmak üzere ilgili bütün bilgi işlem personeli bu eğitimlere gönderilmektedir. 1983'ten beri uygulanan yöntem o denli başarılı olmuştur ki özel sektörce de kullanılmaya başlanmıştır. Hatta artık bilgi işlem elemanı işe almak üzere verilen ilânlarda SSADM kursu almış olma şartı aranır olmuştur.

4.1. SSADM'İN SİSTEM YAŞAM SÜRECİNDEKİ YERİ

SSADM sistem geliştirmenin programlamadan önceki safhalarına hitap eder (Şeki-3) ve iki ana safhası vardır. İlk safha nelerin yapılması gerektiğinin belirlendiği analiz safhasıdır, ikincisi ise bunların nasıl yapılacağının belirlendiği tasarım safhasıdır. Pekçok projede ilk olarak, yine SSADM'in uygulanabileceği bir olurluluk etüdü çalışması yapılabilir.

Her kuruluşta bilgisayar destekli sistemlerin kendi işlem ve yapılarına nasıl uyarlanacağı hakkında belli bir görüş oluşmuştur. İşte SSADM ihtiyaçların belirlendiği böyle bir aşamada başlayabilir. SSADM'e başlayabilmek için ihtiyaçların ifade edildiği bir belgenin hazırlanması yeterlidir.

SSADM sonuçta detaylı olarak :

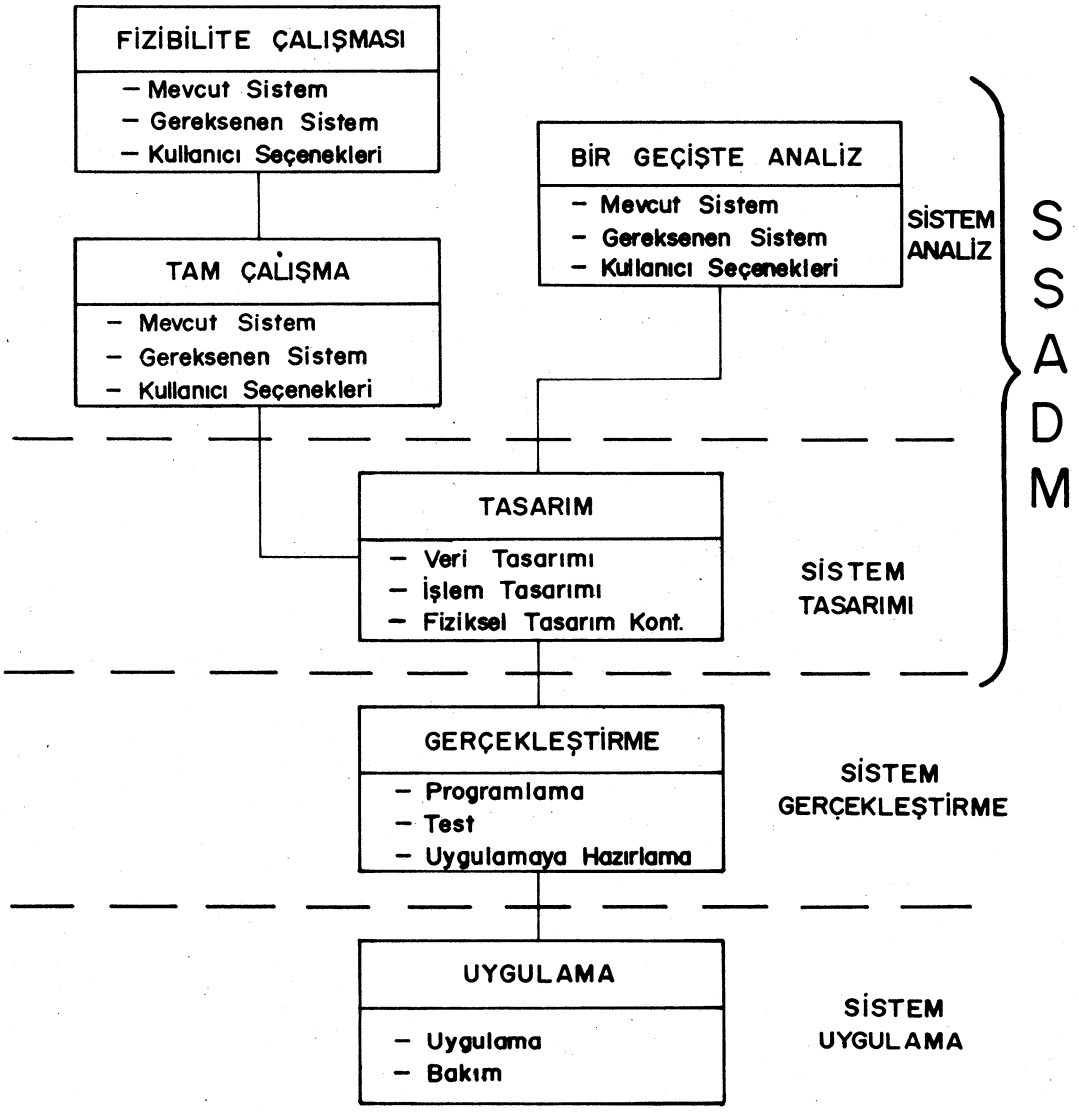
- Hedeflenen donanım ve yazılım ortamı için program özelliklerini ve kütük özelliklerini veya veri tabanı şemalarını,

- Kullanıcı ve işlem talimatlarını üretir. SSADM'in tamamlanmasından hemen sonra programlama çalışması başlayabilir.

4.2. SSADM'İN GENEL ÖZELLİKLERİ

SSADM bilgisayara dayalı bilgi sistemlerinin analiz ve tasarımı için standart bir yaklaşımdır ve yapısal, işlemsel ve dökümantasyon standartlarının bütünleştirilmesinden oluşmuştur.

SSADM diğer klâsik sistem geliştirme yöntemlerinden oldukça farklıdır. Bu nedenle bu paragrafta SSADM'i şekillendiren ana prensiplere kısaca değinmekte yarar vardır.



Şekil-3 : Bilgisayar destekli sistem geliştirme projesi safhaları.

a. Sistem yapıları, veri yapıları ile belirlenir. Fonksiyonlara yönelik değil veriye yönelik bir yöntemdir.

b. Mantıksal ve fiziksel kavramları birbirinden ayırır.

c. Üç temel sistem görüntüsünün; veri akışlarının veri yapılarının ve varlık yaşam öykülerinin paralel geliştirilmesi ile maliyet-yarar açısından uygun bir çözüme erişene kadar iteratiftir.

d. Mantıksal tasarımın fiziksel tasarıma dönüştürülmesi reçetesidir.

e. Ayrıntılı performans, tahmin ve optimizasyon çalışmaları fiziksel tasarımdan gerçekleştirmeye (programlamaya) geçmeden önce yapılır.

f. Kullanıcı, geliştirme sürecinin bütün aşamalarına katılır.

g. Genel yaklaşım tümden gelimcidir.

h. Düzenli ve biçimsel gözden geçirmeler vardır. Bu kontrollarda kalite, tamamlık ve uygulanabilirlik aranır.

4.3. SSADM'İN YAPISAL STANDARTLARI

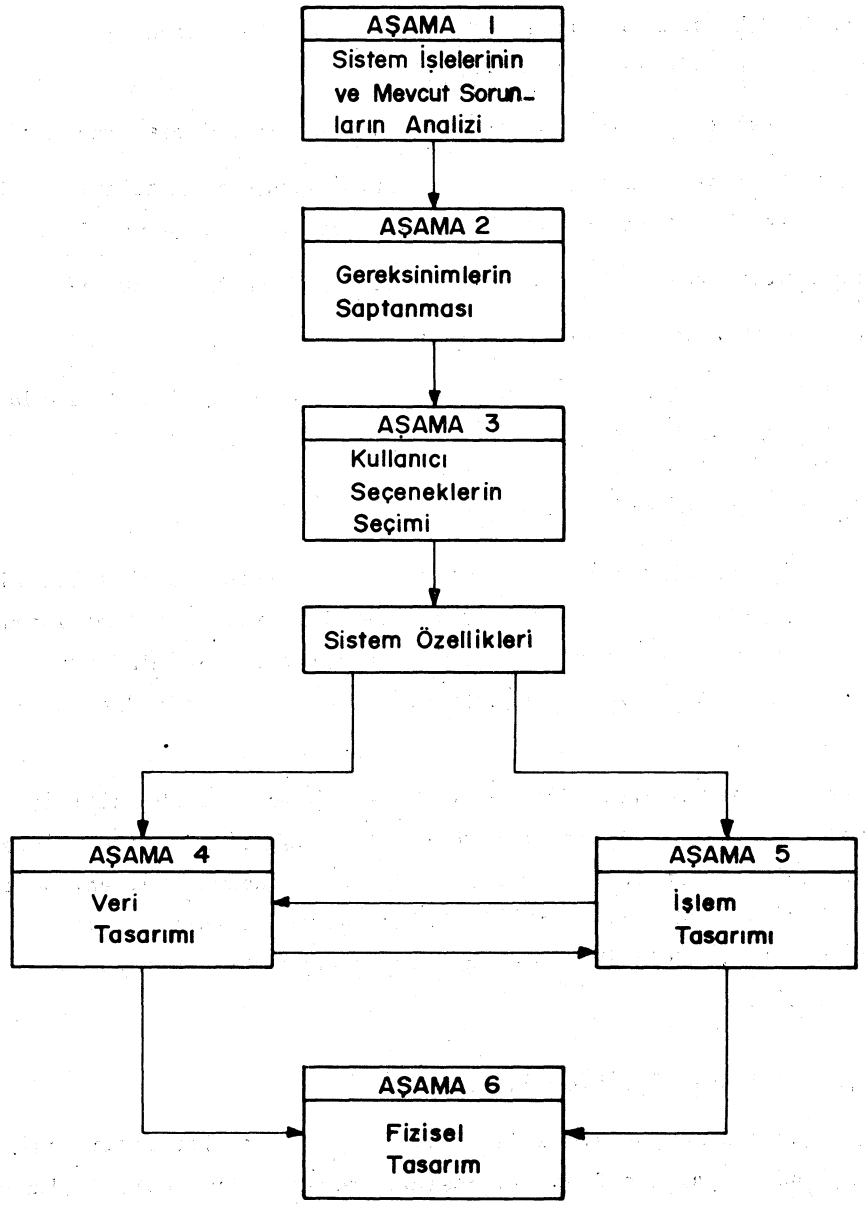
Yapısal standartlar geliştirme projesinin yapısını; açıkça belirlenmiş, sınırlı bir kapsamı olan, aralarındaki ilişkiler ayrıntılı tanımlanmış ve yine açıkça tanımlanmış somut ürünleri olan görevler biçiminde sıralamaktadır. Başka bir deyişle sistem geliştirme projesi sırasında "Ne yapılacağını" açıklar.

Analiz ve tasarım olmak üzere iki safhası bulunan SSADM altı aşamadan oluşmaktadır (Şekil-4). Aşamalar birbiri ile ilişkili adımlar halinde tanımlanmıştır. Herbir adım için bu adımın hedefleri, tanımı, kullanılan teknikler, girdileri ve ürünleri ayrıntılı belirlenerek bu adımı oluşturan görevler sıralanmıştır. Böylelikle oldukça uzun ve ayrıntılı bir görev listesi elde edilmiştir.

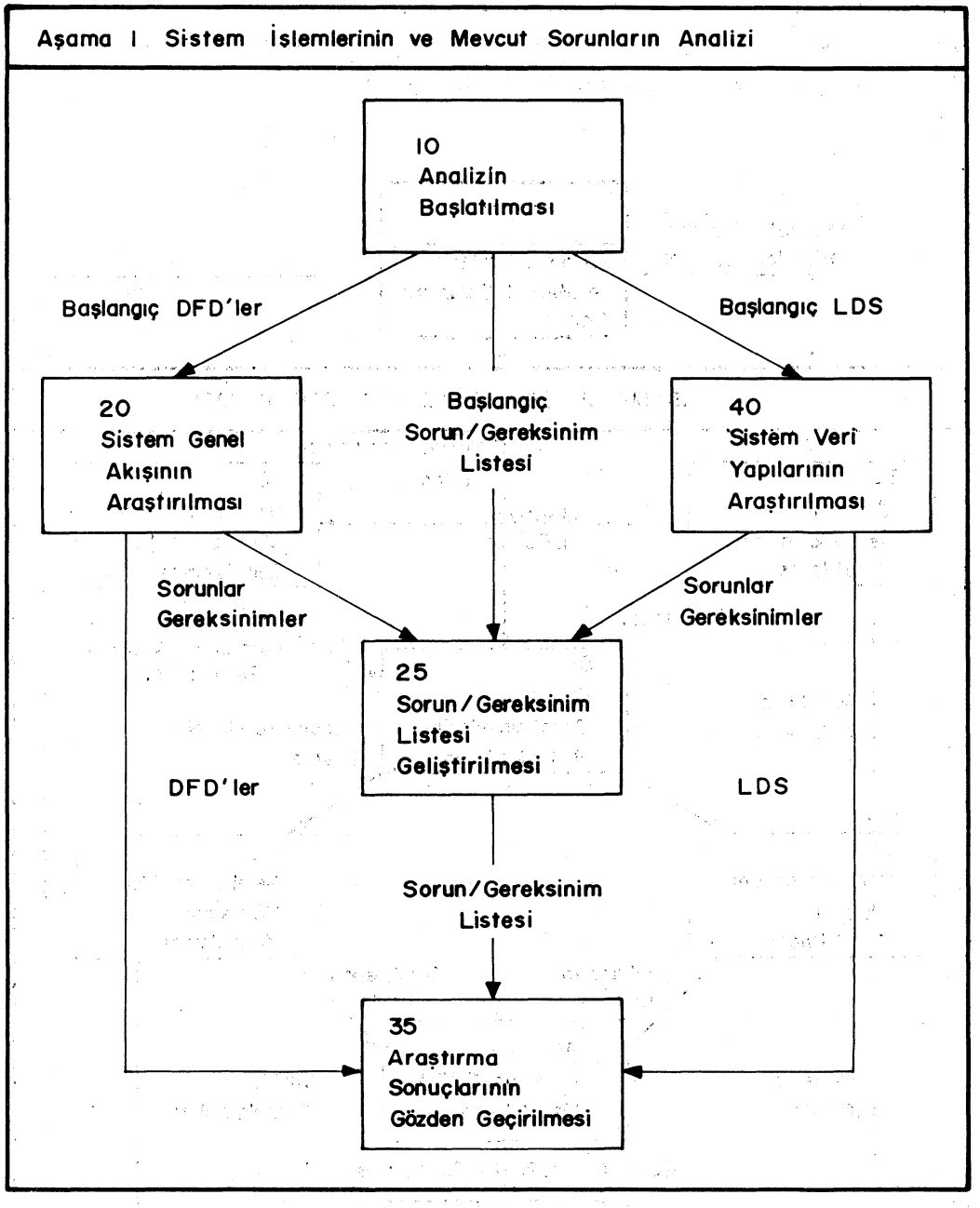
4.3.1. AŞAMA-1 : MEVCUT SİSTEM ANALİZİ

Bu ilk aşamada mevcut sistemdeki işlemler ve sorunlar analiz edilir. Bu aşamada ilk adım analiz safhasının başlatılmasıdır (Şekil-5). Başlangıç dökümanları incelenir, sistemin başlangıç kapsamı belirlenir, müteakip adımların çatısını oluşturmak üzere üst düzey analizler yapılır ve analiz safhası planlanır. Daha sonra mevcut fiziksel sistemin sorunları ve kısıtlamaları ile

birlikte bir tanımlaması yapılır. Bütün varılan sonuçlar sistemin sahibi kullanıcı ile gözden geçirilir.



Şekil-4 : Aşama diagramı.

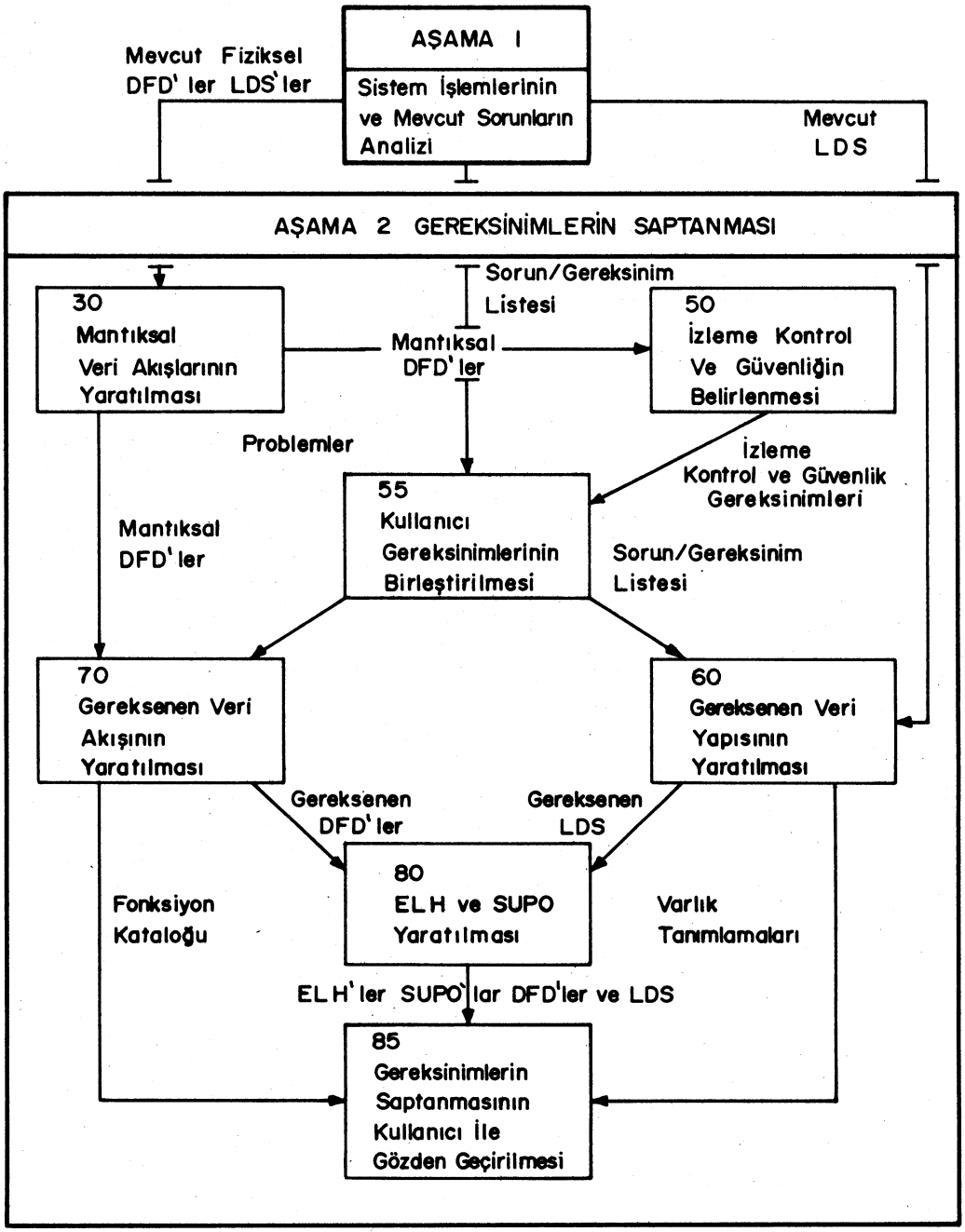


Şekil-5 : SSADM 1 nci aşama diagramı.

4.3.2. AŞAMA-2 : GEREKSİNİMLERİN SAPTANMASI

Bu aşamada birçok faaliyet gerçekleştirilir (Şekil-6). Mevcut fiziksel sistemi ifade etmek üzere hazırlanmış olan veri akış diagramları mantıksallaştırılır; başka bir deyişle sistemin özü ortaya konur. Sistemin güvenlik,

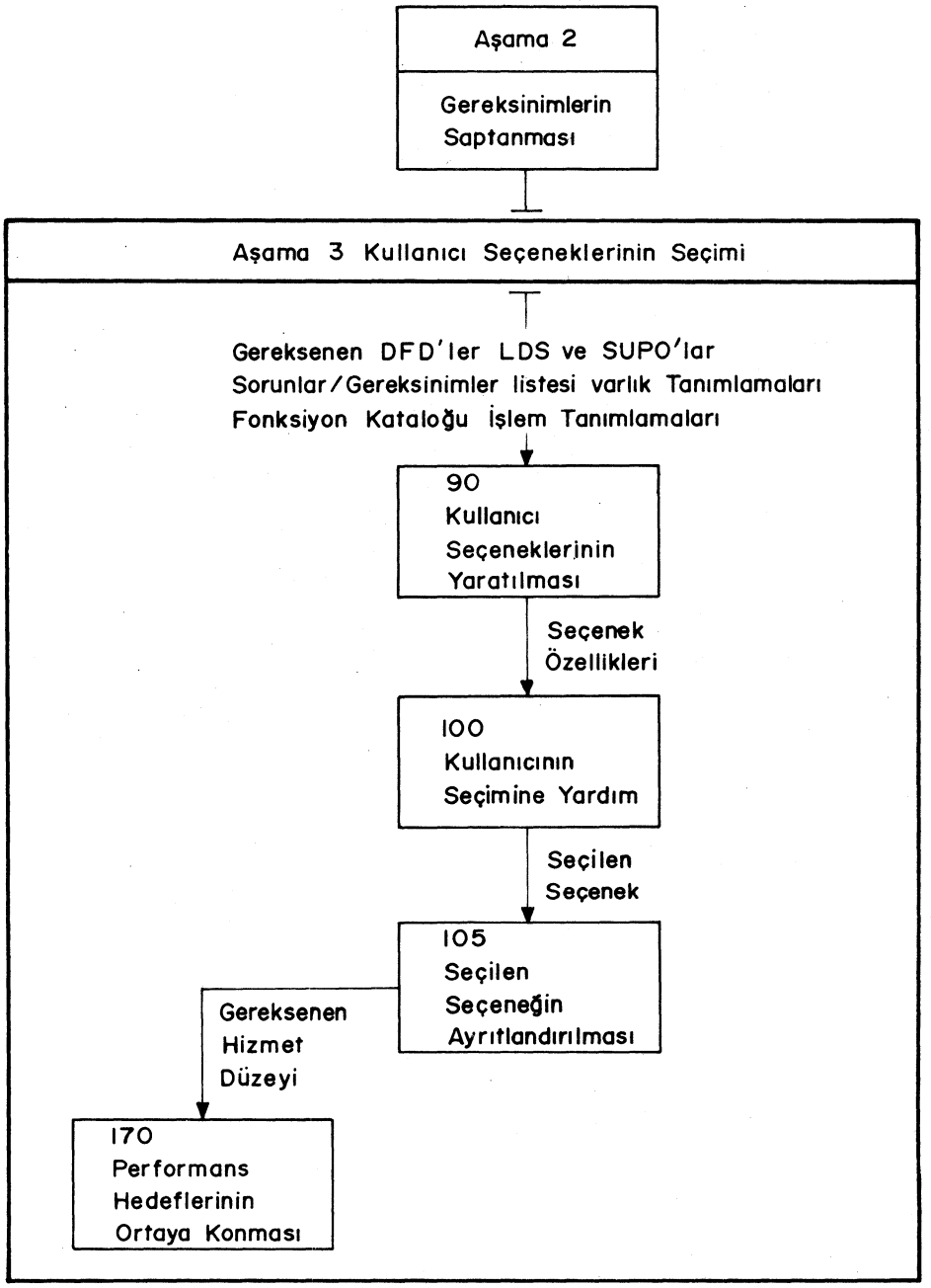
kontrol ve izleme özellikleri incelenir. Baştan beri tutulan sorunlar ve gereksinimler listesi geliştirilir ve gereksenen sistemin bir modeli üretilir. Sonuçlar yine kullanıcı ile gözden geçirilir.



Şekil-6 : SSADM 2 nci aşama diagramı.

4.3.3. AŞAMA-3 : KULLANICI SEÇENEKLERİNİN SEÇİMİ

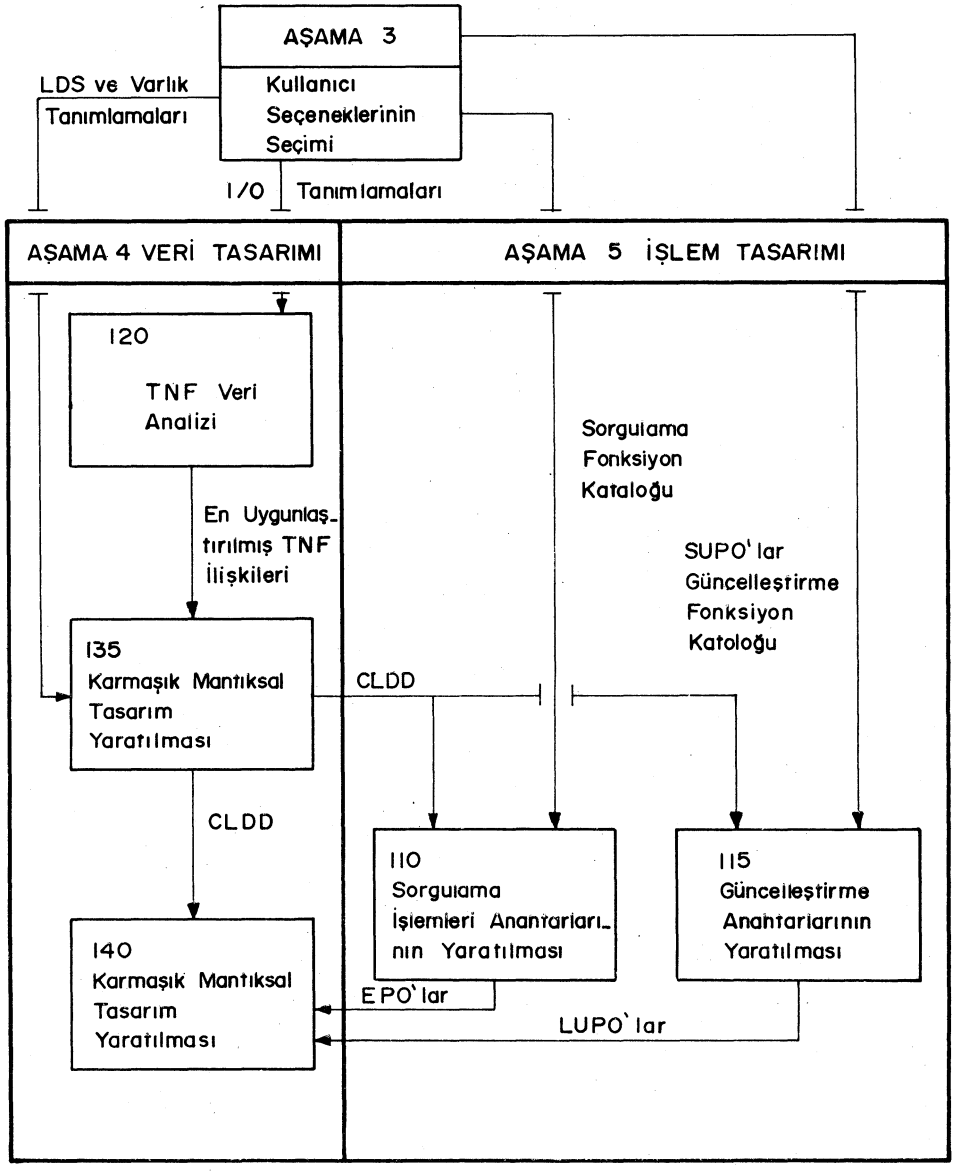
İkinci aşamada ortaya konan sistem gereksinimlerinin karşılanması için birçok seçenek söz konusu olabilir (Şekil-7). Buna göre bu aşamada seçenekler yeterince ayrıntıları ile ortaya konur ve kullanıcının bunlar içinde birini seçmesine yardımcı olunur. Daha sonra seçilen opsiyon detaylandırılarak "sistem özellikleri" hazırlanır.



Şekil-7 : SSADM 3 ncü aşama diagramı.

4.3.4. AŞAMA-4 VE AŞAMA-5 : VERİ TASARIMI VE İŞLEM TASARIMI

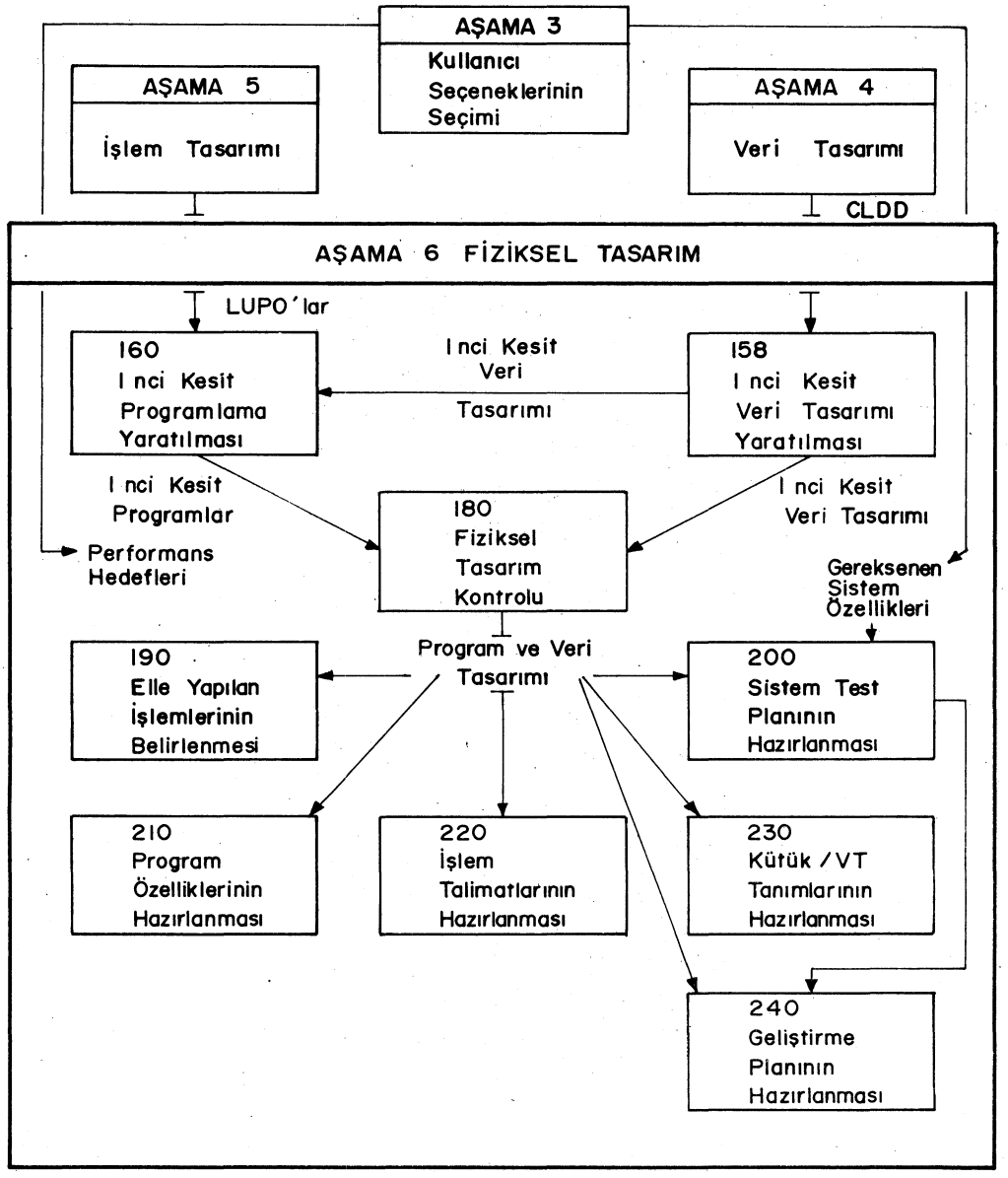
SSADM her ne kadar veriye yönelik bir sistem geliştirme yöntemiyse de işlem tasarımına da gereken önemi verir (Şekil-8). Bu iki aşama paralel yürütülürler. Veri tasarımında daha önce türetilmiş olan mantıksal veri yapısına ek olarak üçüncü normal form analizi yapılır. Sonra bu ikisi "karmaşık mantıksal veri tasarımı" olarak bütünleştirilirler. İşlem tasarımında ise daha önceden hazırlanmış olan sistemin fonksiyonlarına dayalı olarak sorgulama ve güncelleştirme işlemlerinin tasarımı yapılır.



Şekil-8 : SSADM 4 ve 5 nci aşama diagramı.

4.3.5. AŞAMA-6 : FİZİKSEL TASIRIM

Bu aşamada 4 ve 5 nci aşamalarda üretilmiş olan mantıksal tasırım, daha sonraki gerçekleştirme safhası için yeterli ayrıntıda özelliklere yani fiziksel tasarıma dönüştürülür (Şekil-9). Bu çerçevede program özellikleri detaylı olarak belirlenir, tasarım kontrolu ve test planları hazırlanır, işlem talimatları yazılır, elle yapılacak işlemler belirlenir ve son olarak geliştirme planı hazırlanır.



Şekil-9 : SSADM 6 ncı aşama diagramı.

4.3.6. GÖREV LİSTESİ

SSADM'in analiz ve tasarım safhalarını oluşturan altı aşama yukarıdaki paragraflarda çok kısa olarak açıklanmıştır. Bu aşamalar kendi içlerinde Şekil-5, 6, 7, 8, ve 9 da görülen adımlarla ayrılmaktadır. Her adıma tek anlamlı bir numara verilmekte ve daha sonra her adım da bir görev listesi şeklinde ayrıntılandırılmaktadır. Bu görevlerinde numaralandırılması ile uzun bir görev listesi elde edilmiştir (Şekil-10). Sistem analizi ve tasarımı boyunca "ne yapılacağını" detaylı gösteren bu görev listesi bağlayıcı değildir; sistemin özelliklerine göre ufak değişiklikler yapmak suretiyle uygulanır. Böylece yapısal bir standart ortaya çıkmış olur.

```
AŞAMA-1
   ADIM 10
         Görev 10.10
         Görev 10.20
         Görev 10.30
         .
         .
         Görev 10.90
   ADIM 20
         Görev 20.10
         Görev 20.20
         .
         .
         Görev 20.70
   ADIM 25
         Görev 25.10
         Görev 25.20
         .
         .
         .
AŞAMA-2
     .
     .
     .
AŞAMA-6
   ADIM 160
     .
     .
     .
   ADIM 240
```

Şekil-10 : SSADM Görev Listesi

4.4. SSADM'İN İŞLEMSEL STANDARTLARI

Sistem geliştirme yöntemlerinin pekçoğunda sistem geliştirme boyunca neler yapılması gerektiği sıralanırken bu görevlerin nasıl yapılacağı açıklanmamıştır. SSADM'in üstünlüklerinden birisi de budur. SSADM'in işlemsel standartları birtakım teknik ve araçlardan oluşmaktadır ve kanıtlanmış bu tekniklerin ne zaman ve nasıl kullanılacağını da ortaya koymaktadır. Böylece görevlerin nasıl yapılacağı konusunda da bir standart ortaya çıkmaktadır. SSADM'de kullanılan başlıca teknikler şunlardır :

a. LDST (Logical Data Structuring Technique) - Mantıksal Veri Yapılandırma Tekniği : Bu teknik varlık veya veri modellerinin türetilmesinde kullanılır.

b. DFD (Data Flow Diagrams) - Veri Akış Diagramları : Bu teknik bilgilerin sistem içindeki ve sistem ile dış dünya arasındaki akışını göstermek üzere kullanılır.

c. ELH (Entity Life Histories) - Varlık Yaşam Öyküleri : Bu teknik, sistem sürecinin nasıl işlediğini, olayların sistem tarafından nasıl ele alındığını gösteren bir modelin hazırlanması için bir yöntemdir.

d. PO (Process Outlines) - İşlem Anahatları : Bu yöntemle ELH'lar işlemlerin sistem tarafından nasıl gerçekleştirileceğini gösteren tanımlamalara dönüştürülürler.

e. TNF (Third Normal Form) - Üçüncü Normal Form : Verinin tanımlanması ve normalizasyonu için kullanılan bir tekniktir.

f. 1CDD (1st Cut Data Design) - İlk Kesit Veri Tasarımı : Mantıksal veri modellerinin fiziksel kütüklere veya veri tabanlarına dönüştürüldüğü kurallar bütünüdür.

g. 1CP (1st Cut Programs) - İlk Kesit Programlar : Mantıksal işlemlerin sorgulamalara ve program akış diagramlarına dönüştürüldüğü kurallar bütünüdür.

h. PDC (Physical Design Control) - Fiziksel Tasarım Kontrol : Tasarımların kabul edilebilir bir performans ölçütüne göre düzenlenmesi için bir işlemdir.

SSADM'in eğitim dökümanlarında uzun olarak açıklanan bu tekniklerin ayrıntılarına burada değinilmeyecektir. Tekniklerden ilk ikisi pekçok sistem

analiz veya veri tabanı kitaplarında bulunabilir; ancak üçüncüsü oldukça yeni bir yaklaşımdır. Bu üç teknik SSADM'in anahtar teknikleri olarak adlandırılmaktadır.

Başlıcaları sayılan bu tekniklerden başka SSADM ile bir sistem geliştirmek üzere bazı yardımcı tekniklere daha gerek duyulmaktadır. Bunlar :

a. Kalite Kontrol Tekniği

b. Proje Yönetim Tekniği

c. Formal Dökümantasyon

olarak sayılabilir.

4.5. SSADM'İN DÖKÜMANTASYON STANDARTLARI

Dökümantasyondan amaç sistem geliştirme faaliyetinin herbir adımının ayrıntılı olarak kaydedilmesidir. SSADM'de hangi adımda hangi ürünler için nasıl bir form/klişe kullanılacağı örneklerle açıklanmıştır.

5. SONUÇ

Bilgisayar destekli bir sistem geliştirmek üzere iyi bir sistem analiz ve tasarım uygulanmadıkça, gerçekleştirilen sistemlerde birçok aksamalar gözlenmektedir. Böylesi sistemlerin programlamaları sırasında özellikle hatadan ayıklama işlemi uzun süre almakta, test olanaklarının zayıflığından ötürü sistemin güvenirliği azalmakta ve sistem beklenen performansı verememektedir. Öte yandan sistemde yapılması arzulanan küçük değişiklikler veya eklentiler çok zor gerçekleştirilmektedir.

Sistem analiz ve tasarımı için iyi bir sistem geliştirme yöntemi şu yararları sağlamaktadır :

a. İyi bir sistem geliştirme yöntemi kaynak israfı riskini ve dolayısıyla para sarfını azaltmaktadır. Kanıtlanmış bir yöntem bir projede yapılacak işlerin önceden ve en etkin biçimde tanımlanması, dolayısıyla geliştirme ekibi için her zaman takip edilecek bir yolun bulunması demektir.

b. İyi bir sistem geliştirme yöntemi çeşitli şekillerde sistem geliştirme ekibinin verimliliğini artırmaktadır :

(1) Geliştiriciye her proje için "tekerleği yeniden icat etmeye gerek bırakmayan" standart bir çerçeve sağlamaktadır.

(2) Geliştirme faaliyetinin herbir görevi için doğru araçları sağlamaktadır.

(3) İşlemlerin etkin şekilde gözden geçirilmesini zorlamakta, böylece hata ve tutarsızlıklar erkenden belirlenmektedir.

(4) Geliştirme dökümantasyonunu miktar ve içeriğini azaltarak verimliliğe yardımcı olmaktadır.

c. İyi bir yöntem geliştiriciyi yeterince dökümante edilen, esnek bir sistem üretmek üzere zorlamakta; sonuçta geliştirilen sistemin kalitesini artırmaktadır. Ayrıca analizcinin kullanıcı ihtiyaçlarını hassas olarak belirlemesine ve geliştirilen sistemin gerçekten kullanıcının istediği sistem olduğunu kontrol etmeye imkân tanımaktadır.

d. Her aşamanın sonunda yönetimin elinde hangi görevlerin bitirilmesi gerektiğini gösteren bir kontrol listesi bulundurarak proje gelişiminin izlenmesi için yönetime açık bir pencere bırakmaktadır.

e. Sistem geliştirme ile ilgili kişiler arasında;

(1) Kullanıcı ile analizci arasında,

(2) Analizci ile programcı arasında,

(3) Programcı ile veritabanı yöneticisi arasında,

(4) Veri yöneticisi ile veritabanı yöneticisi arasında,

(5) Ekip ile yönetim arasında,

(6) Uygulama ve geliştirme ekipleri arasında

bir iletişim sağlamaktadır.

f. Plânlamayı, izlemeyi, düzeltmeyi ve yeniden plânlamayı kolaylaştırmaktadır.

Klâsik sistem analiz ve tasarım yöntemlerine göre pekçok anlamlı avantajı bulunan SSADM sayılan bu yararları gerçekleştirmek üzere oluşturulmuş bir yöntemdir. 1983 yılından beri İngiltere'nin devlet sektöründeki tüm kuruluşlarında ve birçok özel sektör bilgi işlem merkezlerinde bilgisayara dayalı bilgi sistemleri için başarı ile uygulanmaktadır.

KAYNAKLAR

/1/ BOEHM B.W. 1983 : Datamation
"Software and It's Impect"

/2/ BOOCH G. 1983 : Addison - Wesley
"Software Engineering With ADA"

/3/ COX B.J. 1987 : Addison - Wesley
"Object Oriented Programing : An Evolutionary Approach"

/4/ DEVLIN M.T. 1980 : "Introducing Ada : Problems and Potentials"

/5/ DIJKSTRA 1968 : "The Humble Programmer"

/6/ EVANSON-GODDART, L. 1987 : Government Computing Vol.1 No.1
"SSADM - Friend or Foe ?"

/7/ JACKSON M. 1975 : NY. Academic Press
"Principles of Program Design"

/8/ Learmonth And Burchett Management System Ltd. 1985 : "SSADM Training Manual"

/9/ MILLER G.A. 1956 : The Psychological Review Vol.63 No.2
"The Magical Number Seven, Plus or Minus Two"

/10/ MARTIN J. And LEBEN J. 1986 : Prentice Hall
"Fourth - Generation Languages"

/11/ PARNAS D. 1972 : ACM Vol.15 No.12
"On the Criteria to be used in Decomposing a system into Modules"

/12/ SOMMERVILLE I. 1985 : Addison - Wesley
"Software Engineering"

/13/ SENN J.A. 1984 : Mc Graw Hill
"Analysis and Design of Information Systems"

/14/ SHOOMAN M.L. 1987 : Mc Graw Hill
"Software Engineering : Design, Reliability and Management"

/15/ YÜCE Uğur 1986 : 3 ncü Türkiye Bilgisayar Kongresi
"Bilgi işlem Ne idi, Ne oldu, Ne olacak ?"

/16/ YOURDON E. 1980 : "Software Design : Methods and Techniques"

/17/ YOURDON E. 1979 : Prentice Hall
"Structured Design"