

Bölüm 7

Tahmin

Bu bölümde öğrenilecek kavramlar:

Tahminin önemi	Tahmin Nedir?
<i>İyi bir tahmin yapma</i>	<i>Tahmin Modelleri</i>
<i>COCOMO</i>	<i>REST</i>
<i>Uzman tahminleri</i>	<i>SLIM</i>

Tahmin, projenin hedeflerine göre öncelikle proje büyülüğu daha sonra harcanacak emek, süre ve maliyeti proje başlangıcında öngörmektir. Emek ve süre gibi değişkenlerin sadece toplam büyüklükleri değil, aşamalara nasıl dağılacığı da tahmin edilebilir. Hedeflerin ne kadarına ulaşabileceği diğer bir tahmin konusudur. Yazılım projelerindeki geliştirme adımlarının ne kadar zaman alacağını önceden bileyebilmek zor, ancak mümkün değildir. Bunun için çözümü anlamlı parçalara ayırma, sınıflama, önceki proje tecrübelerini inceleme, uzmanlardan fikir alma ve matematiksel algoritmalarla desteklenen çeşitli yöntemler kullanılır.

Tahmin için önceki projelerde edinilen tecrübeler en önemli araçtır. Önceki projelerin süresi, görevlendirilmiş kişi sayısı, harcanan kaynaklar ve üretilen yazılımlar yeni projeler için yol göstericidir. Tecrübelerden istifade edebilmek için öncelikle tecrübeleri toplamak, değerlendirmek ve saklamak gereklidir.

Tahmin konusunda birçok modelleme yöntemi geliştirilmiştir. Bunların bazıları belli bir yazılım kısmını hedeflerken bazıları da tüm süreçte kullanılabilir. Proje büyülüğu ve geliştirme aşaması model seçiminde etkilidir. Tahmin modelinin karmaşık ve çok fazla değişkene sahip olması, doğru sonuçlar üreteceğini garantilemeyiz. Bazen önceki proje tecrübeleri, matematiksel modeller daha doğru sonuç verir. Mümkünse en az üç yöntem kullanılmalı ve alınan sonuçlar değerlendirilerek nihai tahmin yapılmalıdır.

Tahmin, belirsizlik içerir. Projenin ilk aşamalarında belirsizlik daha fazladır ve yapılan tahminlerin başarısını etkiler. Proje ilerledikçe tahminin doğruluğu artar. İlk başta her şeyi doğru tahmin etmeye çalışmak yerine, başta yapılan tahminler ihtiyaçların netleşmesiyle birlikte güncellenebilir.

Tahmin ve proje planını izlemek arasında yakın ilişki vardır. Sağlıklı tahmin sağlıklı planlamayı ve sağlıklı planlama da sağlıklı ilerlemeyi getirir. Yapılan çalışmalar plandaki tahminlerden büyük ölçüde saparsa, plan önemini yitirir. Proje ekibi üyeleri kendi bildikleri yönlerde yalnız başlarına ilerlemeye başlar.

7.1. Tahminin Özellikleri

Tahmin projeyi henüz gerçekleştirmeden sırasıyla proje büyüklüğü, harcanacak emek ve zaman gibi değerleri öngörmektir. Gelecekteki olaylar tam bilinmeyeceğinden, her tahmin belirsizlik içerir. Yazılım projesinin kendi yapısı itibarıyla içeriği karmaşalık ve değişkenlik da bu belirsizliğe ilave olur. İyi tahmin, belirsizlik ve riski azaltır.

Projenin büyümesi, bilimsel çalışmalar ve kurumun tamamen farklı bir sahaya yatırım yapılması projedeki belirsizliği artırtır. Belirsizliğin artması tahmin aralığını artırmalıdır. Ancak müşteri ve üst yönetim baskısı buna izin vermeyebilir. Üst yönetim kendi hedeflerini taahhüt etmek ister. Proje ekibi ve özellikle de yöneticiyi yaptıkları gerçekçi tahminleri savunulmalıdır. Tahmin plana uymaya zorlanmamalı, plan tahmine uymalıdır. Projenin süresi azaltılmak isteniyorsa tahmin çıktılarıyla oynamak yerine ihtiyaç, kısıt ve alternatif yöntemler üzerinde yoğunlaşmak gereklidir [McConnell-2006.1]. Üst yönetim yazılım konusunda, yazılım ekibi de iş amaçları konusunda bilgi sahibi olursa tahmin ve hedefler arasında ortak noktalar bulunabilir.

7.1.1. Tahmin Mantiği

Tahmin öncelikle doğrudan sayılacak büyükliklere, daha sonra dolaylı hesaplamala-
ra ve en son olarak da kişisel yargılara dayanmalıdır [McConnell 2006.1]. Projede ilk anda kaç adet analiz maddesi olduğu *sayılacak* bir büyükliktür. Bir analiz maddesi için kaç satır kod yazılacağı önceki projelerden *hesaplanarak* bulunabilir. Eğer önceki projelerde bu tür bir bilgi toplanmamışsa veya tamamen yeni bir alanda yazılım geliştiriliyorsa *kişisel yargılarla* tahmin yapılır.

Tahmin yapmak ölçmekten ayrılmaz [Brady-1994]. Öncelikle sayılacak bir şeyler bulmaya çalışılmalıdır. Geliştirme, bunu sağlayacak şekilde yapılmalıdır. Örneğin analiz belgesi serbest metin yerine maddeler şeklinde yazılabilir. Böylece analiz belgesi sayılabilir hale gelir. Üst yönetimin isteğiyle anı tahminler yapılmaması ve az da olsa inceleme yapılması, sayılacak bir şeyler bulabilmeyi ve tahminin doğruluğunu artmasını sağlar [McConnell-2006.1].

Tahmin yaparken birçok etken dikkate alınmalıdır. Projenin yapısı ve karmaşalık derecesi çok önemlidir. Kullanılan yazılım aracının üretkenliği de hesaba katılmalıdır. Buna ilave olarak mimari tasarım da proje zaman ve maliyetini etkileyecektir. Aynı proje, seçilen mimari yaklaşımına göre çok farklı süre ve maliyetlerde geliştirilebilir [Jones-2009]. Proje ekibi tecrübe de tahminleri etkiler. Çünkü için işi kimin yapacağını bilmek, işin süresini hakkında fikir sahibi olmayı sağlar.

7.1.2. Tahminlerin Sistemin Küçük Parçaları Üzerinden Yapılması

Tahminin projenin *iş bölümleme yapısı* üzerinden gerçekleştirilmesi hataları azaltır. Tahmin hatalarının bazıları negatif, bazıları ise pozitif yönlüdür. Negatif ve pozitiflerin birbirini götürmesi neticesinde toplam hata daha küçük olur. Geliştirilecek sistemin parçalara bölünmesi, sistemin daha iyi anlaşılmasını sağlayarak tahmin doğruluğunu artırmasına farklı bir açıdan da fayda sağlar.

Tahminlerin toplamda doğru olması için ayrıntılı tahminlerin beklenen değerinin doğruya yakın olması gereklidir. Bazı araştırmalara göre bu değer %50 oranında tutarlı olmalı ve kısa-uzun tahmin adetleri yakın olmalıdır [McConnell-2006.1]. Ayrıntılıların tamamen tutarsız olması genel toplamda önemli hatalara yol açar.

7.1.3. Tahminin İhtimal Aralığı İfade Etmesi

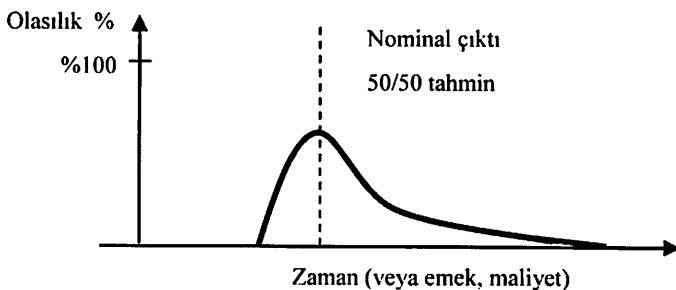
Projede kısıt, tahmin ve hedef farklı kavramlardır [McConnell-2006.1]. *Kısıt*, müşteri ve üst yönetim talebini ifade eder. *Tahmin*, proje büyüklüğünden yola çıkarak yazılım ekibi tarafından belirlenir. *Hedef*, müşteriye taahhüt edilen süre ve özellik gibi kalemlerdir. *Tahmin*, *hedefi* değil, *hedefe* ulaşma ihtimal aralığını ifade eder.

Proje ve görevlerin bitiş tarihiyle ilgili tahminler kesinlikten ziyade ihtimal aralığı ifade etmelidir. Tahminde aralık yerine tek bir nokta kullanılması, aslında bir *tahmin* değil *hedef* ifadesidir. Yazılım ekibi yaptığı tahminlere göre proje hedeflerini gerçekleştirmeyi taahhüt eder. Örneğin müşteri projenin 9 ayda bitmesini ister. Yazılım ekibi proje % 70 ihtimalle 9 ay, % 95 ihtimalle 12 ayda biter şeklinde bir tahminde bulunur. Müşteriye farklı tahminler değil, projeyi 12 ayda bitirmek gibi gerçekleşme olasılığı daha yüksek olan tek bir değer taahhüt edilir.

Müşteri ve üst yönetime tek bir hedef değeri sunulması genellikle tercih edilir. Aksi halde farklı ihtimaller müşteri tarafından anlaşılamayabilir ve bu da müşteri üzerinde yazılım ekibinin yaptığı işten emin olmadığı gibi bir izlenime yol açabilir. Ancak hedef değeriyle birlikte doğrudan hedefle ilgili belirsizliklerden de belli ölçüde bahsedilmesi uygun olacaktır.

Tahmin matematiksel olarak olasılık dağılım fonksiyonlarıyla ifade edilir; bkz. Şekil 7.1. *Normal dağılım* veya *beta dağılım* gibi farklı olasılık dağılımlarıyla çeşitli modeller oluşturulmuştur. Proje belli bir tarihten önce neticelenemez. Bu yüzden belli bir tarihten önce bitme ihtimali sıfır olarak verilmiştir. Projenin bitmemesi de ihtimal dâhilindedir.

Nominal çıktı projenin başarılı veya başarısız bitme ihtimallerinin toplamının eşit olduğu (%50) sınırlıdır. Nominal çıktıının solundaki alan projenin tahmin edilenden daha başarılı, sağındaki alan ise daha kötü neticelenme ihtimalini gösterir.



Şekil 7.1. Tahminin ihtimal fonksiyonu [McConnell-2006.1]

*Belirsizlik, müşteri ve üst yönetime açıkça anlatılmalıdır. İhtimal ifadeleri belirsizliğin anlaşılmasına yardım eder. IEEE-CS/ACM'nin yayımlamış olduğu *Yazılım Mühendisliğinin Ahlaki Kodlarında* belirsizliğin tahminlere yansıtılması ahlaki bir görev olarak gösterilmiştir.*

Madde 3.09: Yazılım mühendisleri çalışıkları veya çalışacakları projenin maliyet, süre, kalite ve çıktılarıyla ilgili gerçekçi sayısal tahminleri garanti eder ve (paydaşlara) bu tahminlerdeki belirsizlik değerlendirmelerini sağlar.

7.1.4. İyimserlik ve Kötümserlik

Projeler genellikle tahmin edilen süreden uzun, nadiren de kısa sürer. Bu durumda zaman açısından ilk yapılacak tahminin iyimser (kısa süreli) veya kötümser (uzun süreli) mi olması gerektiği tartışma konusudur. Bu tartışmada dikkate alınması gereken birçok faktör vardır [McConnell-2006.1].

Tahmin süresinin uzun tutulması ekibin gevşek davranışına ve 3 ayda bitecek bir projeyi 6 ayda tamamlamasına sebep olabilir. Bazı ekipler projenin son anına kadar oyalanır ve bütünlleşme dâhil birçok işi ağır baskı altına girecekleri son ana bırakırlar. Sürenin kısa tutulması ayrıca müşteri beklenileri zamanında karşılanmadığında, projeyi toparlamak için ilave süre de sağlayacaktır.

Tahmin süresi kısa tutulmasının olumsuz etkileri de vardır. Bu durumda proje planı ile gerçekleşen işler arasında uçurum oluşabilir. Plandan sapmanın belli sınır aşması halinde proje tamamen kontrolden çıkar. Kısa süreli tahminler, analiz ve planlama gibi etkisi ilk anda görülemeyen, ancak sonrası için çok faydalı faaliyetlerin üstünköprü yapılmasına yol açabilir. Ayrıca gecikmek ve sözünü yerine getirememek, mahcubiyet ve itibar kaybıdır. Gecikme sebebini müşteri ve yöneticiye izah etmek ilave çaba gerektirir.

Öncelik projenin planlanana göre gerçekleştirilmemesidir. Ancak tahminden sapmanın yol açtığı kontrolsüzlük yüzünden, proje genellikle birkaç gün değil aylar mertebesinde gecikir. Bu yüzden kötümser tahminler iyimser tahminlere göre tercih edilebilir!

7.1.5. Tahminin Zorlukları

Sağlıklı tahmin için öncelikle yapılacaklar adım adım belirlenmelidir. Yazılım proje geliştirme süreci ise birçok belirsizlik ve değişim içerir. Yapılacak işlerin de kısmen tahmine dayandığı söylenebilir. Bu durumda tahmin, yapılacaklar ve bunlara harcana- cak süre, maliyet gibi özelliklerden oluşan iki bilinmeyen tek bir eşitlik şecline gelir. Sonuçta yazılım projesinde tahmin zorlukları oluşur [Jones-2009, Fairley-2009]:

- Yapılacak işlerin ilk anda tam olarak belirlenememesi
- İhtiyaçlardaki değişimin yapılan tahminleri geçersiz kılması
- Önceki projelerle ilgili büyülüklük, emek ve maliyet bilgilerinin saklanmaması veya eksik ve yanlış saklanması
- Projelerde elde edinilen tecrübelerin tahminlere yansıtılmaması
- Tahmin için gerekli temel bilgilerin olmaması
- Tahminde kullanılan yöntem veya ürünün projeye uygun olmaması
- Proje kalitesini artırmak için gerekli belgeleme ve test gibi işlemlerin tahmine dahil edilmemesi
- Yazılım geliştirme ekibinin tecrübe ve becerisinin dikkate alınmaması
- Kontrol edilemeyen dış etkenler: Teknolojik değişimler, donanım temininin geçikmesi vb.
- Yeni teknoloji kullanılması veya yeni alana yapılan yatırımlar
- Bilimsel ve daha önce hiç yapılmamış çalışmalar
- Aşırı bekentiler

7.1.6. Gözden Kaçan İşler

Gözden kaçan işler tahmin doğruluğunu düşürür. Kişiler planladıkları işlerle ilgili tahminleri doğruya yakındır. Ancak genellikle gerekli işlerin %20-30'u unutulur. Bu da tahminde %20-30'luk hatanın sebebidir [Genuchten-1991]. Unutulan işler:

- | | |
|---|--|
| <ul style="list-style-type: none"> • Kurulum için harcanacak zaman, • Kullanıcı belgelerinin hazırlanması • Veri aktarımıları • Önceki projelere destek vermek için ayrılan zaman | <ul style="list-style-type: none"> • Gözden geçirme • Yeni elemanların intibakı • Test verileri oluşturma • Kişiilerin eğitim, tatil ve seyahat süreleri |
|---|--|

7.2. Proje Büyüklüğünü İfade Etmek

Yazılımın somut ve dokunulabilir bir ürün olmaması, bu alanda fiziksel büyülükle- ri ölçerken kullanılan boy, kilo gibi genel kabul gören birimler bulmayı zorlaştı- maktadır. Proje büyülüğünü ifade etmek için en çok kullanılan değişkenler *Kod Satır Sayısı (KSS)* ve *Fonksiyonel Nokta Sayısı (FNS)*.

Proje büyülük ifadelerinin çeşitli olumlu ve olumsuz özellikleri vardır. *Kod satır sayısı*, sayılabilir bir büyülükür ancak maalesef kesin sayı proje sonlarına doğru ortaya çıkar. *Kod satır sayısının ilk aşamalarda tespiti* için ihtiyaç, ürün özelliği, kullanım senaryosu (*use case*), ekran, rapor ve bileşen gibi büyülüklerden yola çıkan dolaylı yöntemler kullanmak gereklidir. *Fonksiyonel nokta sayısı* ise doğrudan sayılabilir ancak ihtiyaç analizinden sonra yani erken aşamalarda belirlenir. Sadece bir ölçü kriterine göre değerlendirme yapmak uygun değildir. Seçilen ölçü biriminin kuvvetli ve zayıf yönlerini dikkate alarak mukayeseli analizler yapılmalıdır.

Projede üretilen tüm büyülükler emek ve zaman gerektirir. Belge ve çizilen akışlar gibi büyülüklerin de tahmini gerekebilir. Ayrıca bazı büyülükler başka çalışmalar sonucu ortaya çıkar ve dolaylı yoldan tespit edilir. Testte çıkabilecek hata miktarı ve hata düzeltme süresi ileriye yönelik tahmin edilebilecek bu tür büyülüklerdir.

7.2.1. *Kod Satır Sayısı*

Yazılım projelerindeki en somut ve ölçülebilir birim (hâlâ) koddur. Bu yüzden yazılım ürününün toplam *kod satır sayısı* en yaygın kullanılan büyülük birimlerinden birisidir. *COCOMO* gibi yaygın tahmin modelleri de kod satır sayısına dayanır.

Kod satır sayısı, proje büyülüğü yanında projedeki ilerlemeyi ifade ederken ve ölçerken de kullanılır. Örneğin *günde üretilen kod satır sayısı* önemli bir ölçütür. Bir yazılım ekibi günde ortalama 250 satır kod üretsin ve bu ekip önceki projelerine benzer 100.000 satırlık yeni bir proje gerçekleştirsün. Bu durumda yeni projede geliştirme için $100.000/250=400$ gün süre gerekeceği tahmin edilebilir. Projede geliştirme aşaması proje süresinin %50'sini alacak şekilde 200 gün olarak tahmin edilirse, bu aşamanın ilk 100 gününde 50.000 satır kod üretileceği öngörlür. Yazılımda genellikle ilk anda üretim daha hızlidır. Sonraki aşamalarda geri dönüşler üretkenliği azaltır. Bu yüzden günlük kod satır sayısını aşamalı değerlendirmelerden sonra kullanılmalıdır.

Satır sayısının bazı dezavantajları bu yöntemin önemini yitirdiği konusunda görüşler ileri sürülmüşe yol açmaktadır [Jones-2009]:

- Son kullanıcı için bir manası yoktur.
- Hesaplama yöntemine göre hesaplanan satır sayısı % 500 fark edilmektedir. Açıklama satırları, tek satırda birden fazla değişken tanımlanması ve aradaki boş satırların hesaba dâhil edilip edilmeyeceği çok önemlidir.
- Yazılımcının kod yazma metoduna bağlıdır. Satır sayısıyla programcı performansını ölçmek, programcılar kısa ve kaliteli yerine, uzun ve kalitesiz kod yazmalarına sebep olabilir.
- Projenin en başında satır sayısını tahmin etmek güçtür. Sonraki değişiklıkların satır sayısına etkisini belirlemek de güçtür.
- Yeni web uygulamalarında *html*, *javascript* vb. birçok farklı dil bulunur.
- Model ve tanım tabanlı programlama gibi modern metodlar, satır sayısı ile proje özellikleri arasındaki ilişkiyi azaltmıştır.

Yukarıda bahsi geçen olumsuzluklara rağmen *kod satır sayısı*, günümüzde önemini korumaktadır. Bunu en önemli sebebi yazılımın koddan oluşması ve kodun sayılabilirliğidir. Kod haricindeki büyüklükler her projede olmamayıpabilir. Bazı projelerde belgeleme eksik bırakılmış veya iş akışı kullanılmamış olabilir. Kod ise her projede vardır!

7.2.2. Fonksiyonel Nokta Sayısı

Fonksiyonel nokta sayısı, bilgi sistemleriyle kullanıcıya sunulan hizmeti esas alan ve günümüzde yaygın kullanılan bir ölçüm birimidir. 1979 yılında *Albrecht* tarafından önerilen bu yöntem 1984 yılından itibaren *Milletlerarası Fonksiyonel Nokta Kullanıcı Grubu* (*IFPUG*: www.ifpug.org) tarafından geliştirilmektedir. Bu yöntemle aşağıdaki özelliklere sahip bir ölçme sistemi önerilir [*Albrecht*-1979]:

- Kullanılan geliştirme teknolojiinden bağımsız
- Kolayca uyarlanabilen
- İhtiyaç analizinden yola çıkarak tahmin edilebilir
- Son kullanıcı için manalı

Fonksiyonel nokta sayısı ihtiyaç analizine dayanır. Sistemdeki tüm analiz maddeleri; harici giriş, harici çıkış, harici sorgu, harici ara yüz ve iç yapısındaki dosyalar şeklinde sınıflanır. "Harici" ifadesi sistemin diğer sistem ve kullanıcılarla ilişkisini gösterir.

- **Harici girişler:** Ekran, form ve dış sistemlerden alınan girişlerdir.
- **Harici çıkışlar:** Rapor, grafik ve farklı programlara gönderilen çıkışlardır.
- **Harici sorgular:** Sistemin anlık ve ham cevaplar olarak ürettiği veri setleri bu kapsamdadır. Veritabanına gönderilip alınan sorgular buna örmektir. Harici çıkış, harici sorguya göre daha formattıdır. Günümüzde veritabanları geliştirdiğim için harici sorgular, harici çıkış ve girişlere yaklaşmıştır.
- **Harici arayüz (dosya)ları:** Diğer sistemlerle iletişim için ortaklaşa kullanılan arayüz ve dosyalardır.
- **İç yapısındaki dosyalar:** Konfigürasyon dosyası ve kod saklama yapısıdır.

Fonksiyonel nokta sayısına dayalı tahminde her özelliğe basit, ortalama ve karmaşık şeklinde bir değer seviyesi verilir. Her sınıfın bir değer seviyesi için karmaşıklığı, *sınıf karmaşıklık katsayısı* ile aşağıdaki tablodaki gibi ifade edilir. Aşağıdaki tablo dan hareketle sisteme yapılacak basit bir girişin karmaşıklığı 3'tür denebilir.

Karmaşıklık	Basit	Ortalama	Karmaşık
Harici Girişler	x 3	x 4	x 6
Harici Çıuşalar	x 4	x 5	x 7
Harici Sorgular	x 3	x 5	x 7
Harici Arayüzler	x 5	x 7	x 10
İç yapısındaki Dosyalar	x 7	x 10	x 15

Dikkat edilirse *dosya ve sistem ara yüzleri* için karmaşıklık katsayıları, diğerlerinden büyütür. Çünkü sistemler arası entegrasyon, en zor konulardan birisidir. Ayrıca sistem entegrasyonu genellikle basit bir işlem olmadıgından *basit değil ortalama* veya *karmaşık* seviyesinde değerlendirilir.

Sistemin FNS açısından toplam büyütüğü; bileşen adedi ile sınıf karmaşıklık katsayıları çarpımlarının toplamıdır. Bu değer *uyarlanmamış toplam sistem büyütüğü* olarak adlandırılır. *Uyarlanmamıştır* çünkü henüz bu değer, yeni geliştirilen yazılım projesinin sahasına uygun hale getirilmemiştir.

$$\text{Uyarlanmamış Sistem Büyüklüğü (FNS)} = \sum_i \text{Giriş (i)} + \sum_i \text{Çıkış} + \sum_i \text{Dosya} + \sum_i \text{Sorgu} + \sum_i \text{Arayüz}$$

Örnek bir proje için *uyarlanmamış toplam sistem büyütüğü* hesabı aşağıdadır.

Karmaşıklık	Basit	Ortalama	Karmaşık	Toplam
Harici Girişler	5×3	4×4	0×6	31
Harici Çıkışlar	1×4	4×5	2×7	38
Harici Sorgular	7×3	5×5	0×7	46
Harici Arayüzler	0×5	1×7	2×10	27
İçyapıdaki Dosyalar	0×7	2×10	1×15	35
Genel Toplam				177

Etki katsayısı ile *uyarlanmamış toplam sistem büyütüğü* çarpılarak yeni geliştirilen yazılım sahasına göre projenin *uyarlanmış toplam sistem büyütüğü* elde edilir. *Etki katsayısı*, 14 faktörün programa etkisini belirten 0.65 ile 1.35 arasında bir sayıdır. Ayrıca bu değer etki katsayısı yerine program büyütüğyle de uyarlanabilir. Bu değerin nasıl belirleneceği *IFPUG web* sitesinden incelenebilir.

$$\text{Uyarlanmış toplam büyütük (FNS)} = \text{Etki Katsayısı} \times \text{Uyarlanmamış Sistem Büyüklüğü (FNS)}$$

Fonksiyonel nokta sayısı ile kod satır sayısı büyütüğü arasında standart dönüşüm tabloları oluşturulmuştur [Stutzke-2005]. Aşağıdaki tabloda *FNS*'nın günümüzün yaygın kullanılan dillerine dönüşümü için kısa bir liste verilmiştir.

	En Az	Ortalama	En Çok
C#	40	55	80
Java	40	55	80
SQL	7	13	15

Bulunan *fonsiyon nokta sayısı*, yukarıda tablodaki sayılarla çarpılarak projenin toplam satır sayısı hesaplanabilir. Yukarıdaki örnekte *FNS*, 177 olarak bulunmuştur. Bunun C# kod satır sayısı eşdeğeri; en az $40 \times 177 = 7080$, ortalama $55 \times 177 = 9735$ ve en çok $80 \times 177 = 14160$ şeklinde bulunur. Kurumsal projelerde elde edilen ders ve tecrübelerle kuruma özel standart çarpanlar oluşturulması uygun olur.

Elbette fonksiyon nokta analizinin de dezavantajları vardır [Jones-2009]. Öncelikle bu yöntem oldukça maliyetlidir. Ayrıca çok fazla alt yönteme bölmüştür ve hangi alt yöntemin kullanılacağını belirlemek oldukça zordur. Fonksiyon nokta analizi ihtiyaç analizinden sonra büyütülük tahmininde kullanılabilir. Oysa firmalar genellikle ayrıntılı analiz yapmadan proje sözleşmeleri yapmak zorunda kalmaktadır. Bu sorunu çözmek için doğruluk kısmen feda edilerek hızlı, analiz tamamlanmadan sonuç üretебilen ve maliyeti az alt yöntemler türetilmiştir. Bu yöntemler belli ayrıntıları göz ardi etmeye dayanır. Belgelerin içeriğine değil büyütügüne veya sadece uygulamaya ilgili sorulara verilen cevaplara göre *FNS* hesaplayan yöntemler mevcuttur.

7.2.3. Proje Büyüklüğünün Farklı İfadeleri

Yazılım projelerinde kod sayısı proje başında doğrudan belirlenebilecek bir büyülüklük değildir. Proje büyülüğünü tespit için çoğu zaman bilinen diğer büyülüklerin kod sayısı şekline dönüştürüerek ifade edilmesi gereklidir. Bu konuda genel bazı standart değişken ve dönüşüm yöntemleri geliştirilmiştir. Ayrıca istenirse kurum içi büyülüklük değişkenleri de geliştirilebilir. Proje büyülüğünü ölçmek için önerilen değişkenler, yazılım geliştirme yöntem ve aşamalarına göre aşağıda sıralanmıştır:

Nesne tabanlı analizin UML ile yapıldığı bir uygulamada	
Senaryo(Use Case)	Senaryo sayısı, adım sayısı
Faaliyet sayısı (Activity)	Akiş sayısı, akışın içerdeği nesne sayısı
Yapısal analiz kullanılan bir uygulamada	
Veri modeli	Varlık sayısı, ilişki sayısı, nitelik sayısı
Akiş	Akiş, akışın içerdeği nesne ve akışlar arasındaki bağlantı sayısı
Kullanıcı Ara yüzü	Ekrان sayısı, rapor sayısı
Tasarım Aşamasında	
Veritabanı ile ilgili	Tablo sayısı, ilişki sayısı, sütun sayısı
Akiş	Akiş, akışın içerdeği nesne ve akışlar arasındaki bağlantı sayısı
Kullanıcı Ara yüzü	Ekrان sayısı, rapor sayısı
Nesne	Sınıf sayısı, sınıf özellik sayısı

Farklı geliştirme yöntemleri için farklı tahmin değişkenleri seçilebilir. Örneğin aşırı programlama için kullanıcı senaryolarına bağlı hikâye sayısı önerilir [Cohn-2006].

Kurum içi tahminin değişkeni geliştirilirken öncelikle aday büyülüklükler ve bunların nasıl hesaplanacağı belirlenir. Her değişkenin proje büyülüğine etki katsayısi, *Fonksiyonel Nokta katsayılarına* benzer şekilde tespit edilir. Örneğin bir veritabanı tablosu veya web sayfasının proje büyülüğünü sayısal etkisi belirlenir. Proje büyülüğü yeni oluşturulan değişkenlere bağlı hesaplanır. Proje sonunda tahmin etkinliği ve doğruluğu kontrol edilir. Gerekirse değişken ve katsayılar tekrar düzenlenir. Projelerde yapılanlar kaydedilerek oluşan veritabanı ile geçmiş projeler karşılaştırılır. Eğilimler tespit edilerek kurumsal seviyede hangi katsayıların kullanılacağı netleşir.

Yazılım bileşenlerine dayalı tahminlerde bileşen sayısı doğrudan belirlenemez. Örneğin projenin ilk aşamalarında kaç tane ekran olacağının bilgisi kesin değildir. Bu yüzden bu tahminler, doğrudan hesaplamaya dayalı tahminlere göre daha az maliyetli, ancak doğruluğu daha azdır [McConnel-2006.1]. Proje büyülüğünü farklı açıdan incelemek birden fazla tahmin yapmayı sağlar. Birden fazla bakış açısıyla güçlendirilen tahminler daha doğru sonuç üretir.

Proje sürecini takip edebilmek için tahmin yaparken projedeki değişimler de dikkate alınmalıdır. Örneğin analizden sonra eklenen ve değişen ihtiyaçların sayısı projenin büyülüğünün nasıl değişeceğini gösterecektir. Tahmin modelleri kısmında değişkenler ve değişkenleri kullanırken dikkat edilecekler ayrıntılı açıklanacaktır.

7.3. Tahmin Modelleri

Tahmin yapılırken en çok kullanılan yöntem, önceki proje değerleri ile yeni projedeki değerlerin kişilerin zihninde karşılaştırılmasıdır. Bu yöntem küçük projelerde etkili olmakla birlikte karmaşık ve büyük projelerdeki ihtiyaçlara cevap veremez. Çünkü bu tür projelerde birbirile ilişkili ayrıntıları, sadece zihinsel olarak değerlendirmek mümkün değildir. Daha resmi ve sistematik yöntemler kullanılmalıdır. Ancak modelin resmi ve karmaşık olmasının doğruluğu artırmak için yetmeyeceği de unutulmamalıdır. Önceki projelerle mukayeseye dayalı yapılan sistematik tahminler, karmaşık modellerden daha etkili olabilir [McConnell-2006.1].

Tahmin modelleri, tahminin hedeflediği değişkenlere göre belirlenir. Emek, süre ve maliyet tahmin edilebilecek ilk değerlerdir. Proje küçükse yapılacaklar belirlenerek bunlar üzerinden ayrıntıdan özete tahmin yapılabilir. Büyük projelerde istatistik teknikleri ile özeten ayrıntıya bir tahmin yapılır. Sonra bu tahminler, doğruluk artırmak için ayrıntıdan özete yapılacak incelemeler ile geliştirilir.

Tahminde kullanılan yöntemler ve bu yöntemler kullanılarak tahmin edilen değişkenlerle ilgili ayrıntılı listelere çeşitli kaynaklara erişilebilir [McConnell-2008]. Model seçerken dikkat edilmesi gerekenler:

- Hangi büyüklüğü tahmin edildiği
- Projenin hangi aşamasında kullanılabilceğini
- Proje büyülüğu
- Tekrar kullanılacak kod miktarı
- Proje geliştirme yöntemine ve ne derecede doğruluk istediği

Kullanılan modelde çıkan sonucun küsuratlı olması tahminin doğruluğunu göstermez [McConnell-2006.1]. Virgülden sonraki rakamlar, tahminin çok ayrıntılı yapıldığı izlenimi verir ancak çoğunlukla gereksizdir. Tahmin sunulmadan önce bunlar tam sayıya çevrilmelidir. 12,65 yerine 13 ay şeklinde bir tahmin daha anlamlıdır.

7.3.1. Önceki Projelere Benzeterek Tahmin Etmek

Projedeki, faaliyetlerin süre ve maliyetini tahmin ederken en geçerli yöntem, aynı ekip tarafından aynı alanda daha önce yapılmış projelerdir. Proje ekibi önceden edindiği tecrübeleri yeni projeye göre tekrar değerlendirmeli ve yeni projenin farklılıklarını da dikkate almalıdır. Farklı ekiplerin tecrübeleri de yol göstericidir. Ancak aynı ekip tarafından yapılan işlerin tahmine etkisi çok daha fazladır. Teknoloji ve alan değişiminin ekip performansını önemli ölçüde etkileyeceği akılda tutulmalıdır.

Çok fazla proje yapmış olmak tecrübe sahibi olmayı garanti etmez. Projeden çıkartılan derslerden faydalananın için geliştirme sürecinde ortaya çıkan veriler dikkatli, periyodik ve planlı şekilde ölçülmelidir. Proje ekibi konuya alakalı yaşadıklarını belli aralıklarla kaydediyor, derse ve tecrübe dönüştürüyorsa sonraki projelerde faydalana bilir. Aksi halde aynı hataları tekrar edecektir.

"Tarih"i tekerrür diye ta'rif ediyorlar; Hiç ibret alınsaydı tekerrür mü ederdi?
Mehmet Akif ERSOY

Her yazılım projesi kendi başına tektir. Kendine özel ihtiyaçları vardır. Kullanıcı ihtiyaçlarının değişimleri ve özellikle de artabilir. Yazılım geliştirme metodu ve teknolojisi değişimlerdir. Yazılım ekibinde de değişimler görülebilir. Projeleri birebir aynı kabul ederek yapılacak tahminler tutarlı olmayacağındır. Tahmin yaparken birbirile benzer özellik taşıyan projelerden alınan değerler mukayese edilmelidir. Aynı tür projelerde, benzer çalışma şartlarında, aynı yazılım geliştirme ortamı kullanırken yapılan ve aynı büyüklükteki projelere dayanan tahminlerin doğruluğu daha fazladır.

Yazılım proje büyülüğü, tahminlerin doğruluğunu etkileyen önemli bir değişkendir. Tahmin yaparken benzer büyülükteki projelerden alınan değerler kullanılmalıdır. Aksi halde beklenmedik sonuçlarla karşılaşılır. Proje modüllere bölünerek, büyülü makul düzeye indirilerek tahmin doğruluğu artırılabilir. Geliştirme ortamı da üretkenliği, dolayısıyla tahminleri etkiler [Jones-2009]. Örneğin önceki projede Java ile kod yazarken ulaşılan üretkenliğin aynısını, Microsoft® VS.Net ile yazarken beklemek uygun olmayacağındır.

Tahmin için yakın tarihte yapılan projeler daha fazla mana ifade eder. Bunların tahmine olan etkisini yüksek tutmak gereklidir. Güncel projeler, çok önceki projelere göre önemli farklılıklar içerebilir. Bu durum tahmin doğruluğunu olumsuz etkiler.

Projedeki değişimlerin ekip üzerindeki etkisi beklenmedik ölçüde büyük olabilir. Projeyi bilen mevcut ekibin değişimine hemen uyum gösterebileceği varsayımları, karmaşılığı artıracak ilave özellik ve yeni teknolojiler söz konusu olduğunda yanlış çıkabilir. Ekibin projedeki değişikliklere uyum sağlayıp sağlayamadığı çok dikkatli takip edilmeli, gerekirse ekibe takviye yapılmalı veya yeni ekip görevlendirilmelidir.

Senaryo: Artan Karmaşıklığın proje geliştirmeye etkisi

Personel programının yeni ve daha kapsamlı bir sürümünün yapılması gereklidir. Bu proje için mevcut programın destek ekibine görev verilir. Proje için üç ay gibi bir süre öngörlür. Proje başladıkta sonra ekibin yeni ihtiyaçlara uygun bir sistem mimarisi geliştirmekte zorlandığı fark edilir. Ancak bunun bir uyum zamanıyla düzelleceği kabul edilir. Planda veya ekipde herhangi bir değişikliğe gidilmez.

Proje tasarımını planlanandan uzun süre ve sonunda ortaya çıkan tasarım önceki projeye benzer özellikler taşıır. Yapılan gözden geçirme toplantılarında yeni isteklerin redeye hiç birinin tasarıma yansıtımıldığı görülür. Projeyi geliştiren ekip bir sistem mimarıyla takviye edilir. Proje tekrar tasarlanır ve isteklere uygun ancak önceki proje göre karmaşık bir proje ortaya çıkar. Tasarım tekrar önceki ekibine verilerek kodlama aşamasının gerçekleştirilemesi istenir.

Proje ekibi karmaşık tasarımın kodlaması konusunda da zorluklar yaşar. Proje ancak planlanan sürenin dört katı zamanda yani yaklaşık bir yılda kullanıma başlanır.

Projelerdeki veriler mümkün olan en kısa sürede toplanmalıdır. Projeye ilgili veri toplama için en doğru zaman proje devam etme anıdır. İşler yapılrken veya biter bitmez işe ilgili bilgiler kaydedilmelidir. Aradan uzun süre geçtiğinde yapılanlar, başlangıç-bitiş tarihleri ve harcanan kaynaklar gibi bilgiler unutulur. Hatta bazı bilgilerin tespit edilmesi tamamen imkânsız olabilir [McConnell-2006.1]. Örneğin bir kişinin birden fazla projeye ilgilendiği durumlarda hangi projede ne kadar kod yazdığı bilgisinin proje sonunda toplanması mümkün değildir.

Benzetme Yolu ile Tahmin süreci

Yeni yapılacak projenin zaman, harcanacak emek ve maliyet gibi özelliklerinin eski projelerde benzeterek tahmin edilmesi diğer tahmin modelleri gibi sistematik ve aşamalı şekilde yürütülmelidir [McConnell-2006.1].

1. Öncelikli olarak hangi kalemlere göre tahmin yapılacağının belirlenmesi gereklidir. Tahmin değişkenleri kısmında anlatılan değişkenler arasından proje ve kurum yapısına uygun olanlar seçilmelidir. Aynı zamanda bu değişkenlerin hangi büyülük cinsinden ifade edileceği (birimi) belirlenir. Aşağıdaki örnekte tablo, ilişki, ekran, rapor ve nesne sayısı değişkenleri belirlenmiştir. *Kod satır sayısı*, ortak büyülük ifadesi olarak seçilmiştir. Yeni ve önceki projelerde her bileşen için toplam kod satır sayısının aynı yöntemle hesaplandığı kabul edilmiştir.
2. Daha önce yapılmış benzer projelerde bileşen veya özellikler için ne kadar toplam kod yazıldığı bulunur. Buradan da bir bileşen türü için ortalama ne kadar kod yazıldığının hesaplanır. Örneğin aşağıdaki tabloda bir ekran için *Ortalama KSS* 306 satırıdır.

Tahmin Değişkeni	Önceki Adet	Önceki Toplam KSS	Ortalama KSS
Tablo Sayısı	15	1.980	132
İlişki Sayısı	12	260	22
Ekran Sayısı	8	2.450	306
Rapor Sayısı	12	9.30	78
Nesne Sayısı	20	1.341	67

3. Ortalama değerler kullanılarak yeni yapılacak projenin toplam tahmini kod satır sayısı hesaplanır.

Tahmin Değişkeni	Yeni Proje Değeri	Önceki Ortalama KSS	Değişken Toplam KSS
Tablo Sayısı	20	132	2.640
İlişki Sayısı	26	22	563
Ekran Sayısı	15	306	4.594
Rapor Sayısı	17	78	1.318
Nesne Sayısı	30	67	2.012
Metot Sayısı	20	132	2.640
<i>Yeni Proje Genel Toplamı</i>			11.126

Yeni projede ilk aşamada bileşen sayısı belirsiz olduğundan doğrudan tek bir sayı değil PERT yöntemi gibi *en küçük, en büyük ve ortalama değer* kullanılarak hesaplanan *beklenen değer* tahmin değişkeni olarak kullanılabilir; bkz. Kısım 7.3.5.

4. Önceki ve yeni proje büyüklüklerinin mukayesesи ve büyüklük tahmininden emek tahminine geçiş yapılması gerçekleştirilir.

Ara Katsayı = Önceki proje büyüklüğü KSS / Yeni proje büyüklüğü KSS

$$\text{Ara Katsayı} = 11126/6961 = 1,59$$

Yeni projedeki toplam emeğin hesabı için aşağıdaki basit eşitlik kullanılır.

Yeni projedeki Toplam Emek= Önceki Projedeki Toplam Emek x Ara Katsayı

Netice olarak önceki proje 30 adam/aylık bir emekle yapılmışsa ikinci proje tahminen yaklaşık $(30 \times 1,59)$ 48 ay sürecektir.

Yazılım projelerinin büyüklüğü ile harcanacak emek arasında üstel ilişki vardır. Eğer yeni proje ilk projenin iki katından daha büyükse harcanacak emek ve zaman, büyüklük artış oranından daha fazla artar. Büyüklük 3 kat artarken emek 5 kat artabılır. Bu yüzden büyüklüğü iki katından fazla fark eden projelerde değişkenler arasındaki oran, ayrı bir *Büyüklük Düzenleme Katsayısıyla* da çarpılmalıdır.

$$\text{Yeni Proje Tahmini} = \frac{\text{Büyüklük Düzenleme}}{\text{(Adam/Ay)}} \times \frac{\text{Önceki Projedeki}}{\text{Katsayı}} \times \frac{\text{Emek(Adam/Ay)}}{\text{Ara Katsayı}}$$

Büyüklük Düzenleme Katsayısı önceki projelerin büyüklükleri arasındaki mukayeseyle bulunabilir. Bölümünde devamında *COCOMO* kısmında büyüklükteki değişimi proje tahminlerine yansıtmak için çeşitli öneriler mevcuttur. Yukarıdaki örnekte ara katsayı 1,59 çıktılarından *büyüklük düzeltme katsayısı* 1 kabul edilmiştir.

7.3.2. *Mevcut Proje Verisi ile Tahmin*

Proje gelişikçe konular netleşir. Netleşmenin kaynağı bilgi artışıdır. Mevcut projede ortaya çıkan veriler, sonraki aşamaları tahmin için kullanılabilen önemli bir kaynaktır. Örneğin bir projede toplam 60 adet test senaryosu bulunsun. İlk 10 test senaryosunun gerçekleştirilmesi ortalama 3 gün alırsa geriye kalan test senaryoları için $50 \times 3 = 150$ adam/gün büyüklüğünde bir çalışma gerekecektir.

Harici kurum ve projelerden elde edilen tecrübeler, mümkün olan en kısa zamanda mevcut kurum ve projedeki tecrübelerle yer değiştirilmelidir [McConnell-2006.1]. Proje ilerledikçe çalışanların yaptıkları işlerle ilgili bilgileri artar ve daha doğru tahminler yapılmaya başlanır. *Yinelemeli geliştirme* gibi yöntemlerin seçilmesi de verileri sürüm tabanlı toplama ve kullanmayı kolaylaştırarak tahminde yardımcı olur.

7.3.3. *Süre Tahmin Genel Formülü*

Projede emek tahminini, en uygun süre tahminine dönüştürmek için genel kabul görmüş formül aşağıdadır. *En uygun süre, ekip sayısı* fayda maliyet açısından en uygun sayı olduğundaki süredir; b.kz. Kısım 4.5.2. Projenin erken aşamaları için uygun bir modeldir.

$$\text{En Uygun Süre} = 3 \times \text{Adam/Ay}^{1/3}$$

Üstel değer olarak 50 adam/aydan büyük yazılım projeleri için $1/3$ küçük projelerde $\frac{1}{2}$ olarak kullanılması önerilir. Buradaki rakamlar kuruma özelleştirilebilir. Bu formüle göre 60 adam/ay emek gerektiren bir proje $3 \times 60^{1/3} = 11,74$ ay sürecektir. Burada küsurat gereksizdir. Elbette ekip sayısına bağlı olarak süre değişecektir. Yukarıda hesaplanan süre projede müşterinin istediği süreyle aynı anlama taşımaz. Müşteri projenin 6 ayda bitmesini isterken tahmini süre 12 ay olabilir.

7.3.4. Ortak Değerlere Çevirme

Dolaylı olarak tahmin edilen test durumu, hata sayısı gibi büyülüklüklerin tahmininde *vekil veya dönüşüm tabanlı tahmin* (*proxy based estimate*) teknikleri kullanılır. Bu teknikler *bulanık mantık*, *standart bileşen* ve *elbise büyülüklüğü*dür. Öncelikle projedeki değerlerin ortak bir değere nasıl çevrileceği belirlenir. Sonra ortak değerler önceki projelerle karşılaştırılır. Bu tekniklerin doğruluğu, hesaplamaya dayalı tekniklere göre daha azdır. Mümkünse hesaplamaya dayalı teknikler tercih edilmelidir.

Bulanık Mantığa Dayalı Tahmin

Geliştirilecek yazılım ürününü özelliklerin gerçekleştirilmeye zorluğuna göre çok basit, zor gibi isimlendirilmesine dayanan tahminler bulanık (*fuzzy*) tahmin yöntemi olarak isimlendirilir [Putman-1992]. Aşağıdaki gibi bir tablo oluşturulur.

Ölçek	Her özellik için ortalama KSS
çok basit,	100
basit,	205
ortalama,	425
karmaşık	860
çok karmaşık	1.780

Bulanık tahmin katsayıları kurumun önceki projelerine dayanır. Önceki projeler incelenerek özellik türleri, ürün ölçüği (büyük, orta, küçük) ve her ölçek için katsayı belirlenmelidir. Sonra belirlenen katsayılar yeni projelerde tekrar incelenir ve gereklirse düzenlenir.

Katsayılar belirlenirken, büyülüklükleri arasında en az iki kat fark olması önerilir [Putman-1992]. Yöntemi doğru uygulayabilmek için herhangi bir sınıftaki özellik değeri ortalamadan çok fazla sapmamalıdır. Örneğin yukarıdaki tabloya göre *basit* olarak sınıflanan bir özellik 1500 satır kodla gerçekleştirilirse genel doğruluk azalır. Bulanık tahminle yeni bir projenin büyülüğünün hesaplanması:

	Her özellik için ortalama KSS	Yeni Projedeki Özellik Sayısı	Tahmin Edilen Toplam KSS
çok basit,	100	26	2.600
basit,	205	35	7.175
ortalama,	425	15	6.375
karmaşık	860	11	9.460
çok karmaşık	1.780	5	8.900
Genel Toplam			34.510

Bulanık tahmin teknigi doğrudan emeği ölçmek için de kullanılabilir. Her özellik türü için harcanacak emek yukarıdaki gibi tabloya aktarılır ve toplam emek bulunur.

Bulanık mantığın özellikleri kategorize etme şekli açısından farklı sürümleri vardır. Örneğin aşırı programlama için hikâye noktaları (gerekşim veya özellik) önerilmiştir [Cohn-2006]. Bunlara zorluk derecesine göre ikinin katları veya Fibonacci (1, 2, 3, 5, 8, 13 ...) sayıları atanarak her yineleme neticesinde tahmin edilen ve gerçekleşen değerler mukayese edilir.

Standart Bileşenler

Standart bileşenlere dayalı tahminde, tahmin bileşenin kod sayısı cinsinden ifade edilmesiyle gerçekleştirilir. Bu yöntem genellikle benzer türde yazılım projeleri geliştiren şirketlerde kullanılır. Bileşenler küçük, orta ve büyük şeklinde sınıflanıp her sınıf için bir KSS büyülüğu belirlenir.

	Basit	Orta	Karmaşık
Giriş Ekranı	500	1200	2.500
Gözlem Ekranı	300	700	1.500
Rapor	200	450	1.000

Bileşenler arası ilişkilerden yola çıkarak da çeşitli katsayılar belirlenebilir. Örneğin ekranlar tablo sayısına göre “tek tablo içeren ekranlar ortalama 300 satır”, “İlişkili iki tablo içeren ekranlar 1.000 satır kod içerir” şeklinde sınıflanabilir.

Elbise büyülüğu

Elbise büyülüyü yöntemi, özelliklerin iş değerlerini ve maliyetlerini müşterilerin anlayacağı şekilde (küçük, orta, büyük gibi) kabaca sınıflamak için kullanılır.

Özellik	İş Değeri	Geliştirme Maliyeti
Özellik 1	Büyük	Büyük
Özellik 2	Büyük	Küçük
Özellik 3	Küçük	Büyük

Maliyeti yüksek ve iş değeri düşük özellikler projenin ilk aşamalarında tekrar değerlendirilerek elenebilir. Böylece bu tür özellikler için analiz ve tasarım gibi çalışmalar yapmaya gerek kalmaz. İş değeri ve geliştirme maliyetine çarpımına toplam bir değer atanması değerlendirmeyi kolaylaştırır. Örneğin:

- Her ikisi de eşitse: 0
- Çok büyük maliyetli ve iş değeri küçükse – (negatif)
- Küçük maliyetli ve iş değeri büyüğe + (pozitif)

7.3.5. Program Gelişim ve Gözden Geçirme Tekniği (PERT)

Program gelişim ve gözden geçirme teknigi (PERT , “Program Evaluation and Review Technique”) projenin tamamının veya belirlenmiş bir kilometre taşının zamanında tamamlanıp tamamlanamayacağıyla ilgili bir olasılık dağılımı oluşturmayı sağlar [Malcolm-1959]. Özellikle zamanla ilgili tahmin geliştirilmesinde kullanılır.

PERT metodunda her alt görev için üç değişken tanımı yapılır.

- **İyimser:** Görevin bitebileceği en kısa süre
- **Ortalama veya büyük ihtimal:** Görevin genellikle tamamlandığı süre
- **Kötümser:** Görevin bitebileceği en uzun süre; en kötü durum tüm şartların en kötüye gittiği durumdur.

İyimser, ortalama ve kötümser zaman parametreleri kullanarak görevler ve projenin beklenen değeri (σ) ve standart sapması (μ) hesaplanır. Bu değerler kullanılarak da projenin istenen zamandan önce veya sonra bitme olasılık dağılımı belirlenir. *PERT*, üç noktalı dağılım tekniğinin bir alt türüdür. Üç noktalı dağılımda; $((O + M + P) \div 6)$ şeklinde basit ortalama alınır. *PERT* için genelde kullanılan formül ise:

$$\text{Tek Bir Görev İçin Beklenen Süre} = (O + 4M + P) \div 6$$

Tahminler genelde iyimserdir. Bu yüzden kötümser tahminin etkisini artıran farklı *PERT* formülleri de önerilmiştir [Stutzke-2005].

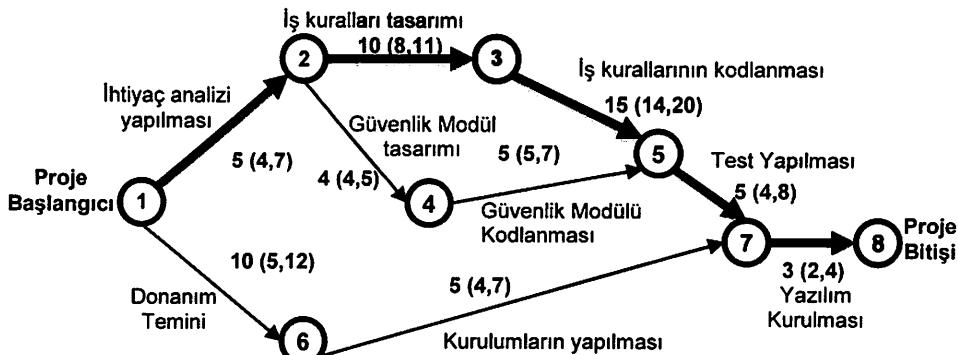
$$\text{Tek Bir Görev İçin Beklenen Süre} = (O + 3M + 2P) \div 6$$

PERT metoduyla doğru sonuç alabilmek için bir görevin iyimser, ortalama ve kötümser zaman değişkenlerini doğru tahmin etmek gereklidir. Bunun için önceki projelerden edinilen tecrübeler, en iyi ve kötü alternatifler de düşünürlerek incelenmelidir.

Dikkat edilirse en iyi olaslığın olma ihtimali 6 da 1'dir. Eğer bölümleme yapıılırken sadece en iyi olasılıklar dikkate alınırsa projenin en iyi sürede bitme ihtimali aşağıdaki gibi hesaplanabilir.

$$\text{Projenin en iyi sürede bitme ihtimali} = (1/6)^n \quad (n: \text{görev sayısı})$$

5 tane görev olan bir proje için bu rakam 7776'de 1 olur ki bu da ihtimalin ne kadar düşük olduğunu gösterir. Buradan hareketle her şeyin iyi gideceği düşünürlerek yapılan tahminlerin doğru çıkma şansının ne kadar düşük olduğu tekrar hatırlanabilir.



PERT yöntemi kritik yol yöntemiyle birlikte projenin belli bir sürede bitme ihtimalini hesaplamak için kullanılabilir. Projedeki görevler arası ilişkileri ve görevlerin en iyi, ortalama, en kötü sürelerini gösteren şemaya *PERT* şeması ismi verilir; bkz. Şekil 7.2. *PERT*, genellikle *normal(Z)* ve *beta* istatistiksel dağılımlarıyla ifade edilir. Bu kısımda kullanılacak değerler, kritik yol kısmında verilen plan üzerinden oluşturulmuştur; bkz. Kısım 5.8.3. Bu yöntemde yapılanlar aşağıda sıralanmıştır:

1. Kritik yoldaki tüm görevlerin beklenen değerleri toplanarak kritik yol için beklenen değer bulunur.

$$\begin{array}{l} \text{kritik yol için} \\ \text{beklenen süre} \end{array} = \frac{(4 + 5 \times 4 + 7)}{6} + \frac{(8 + 10 \times 4 + 11)}{6} + \frac{(14 + 15 \times 4 + 20)}{6} + \frac{(4 + 5 \times 4 + 8)}{6} + \frac{(2 + 3 \times 4 + 4)}{6}$$

PERT formülüyle kritik yol için beklenen süre 39 hafta olarak bulunur (Kritik yol ortalama süre üzerinden hesaplandığında 38 haftadır). Kötümser tahminin etkisini arturan *PERT* formülü kullanılırsa süre 41 hafta olarak hesaplanacaktır. Bu değer projenin bitiş süresi olarak % 50 emin olunan değerdir.

2. Kritik yol üzerindeki her görev için standart sapma bulunur. Eksi sayıların etkisini gidermek için bunların kareleri toplanır ve *toplam standart sapma* hesaplanır.

$$\text{Standart Sapma} = \mu = (\text{En İyi} - \text{En Kötü}) \div 6$$

$$\text{Yayılım(varyans)} = \sigma = (\text{Standart Sapma})^2$$

$$\text{Toplam } \sigma = \frac{(7 - 4)^2}{6} + \frac{(11 - 8)^2}{6} + \frac{(20 - 14)^2}{6} + \frac{(8 - 4)^2}{6} + \frac{(4 - 2)^2}{6}$$

$$\text{Toplam Standart Sapma} = \sqrt{\sigma_{\text{TOPLAM}}} = 3,51$$

3. Projenin bitmesi istenen sürenin gerçekleşme ihtimali, Z dağılımı ile hesaplanır.

$$Z = \frac{\text{Projenin bitmesi istenen süre} - \text{Toplam Beklenen Süre}}{\text{Toplam Standart Sapma}}$$

Projenin Bitmesi istenen süre 35 hafta olsun. Z değeri

$$Z = \frac{35 - 39}{3,51} = -1,13$$

Burada Z değeri, ihtimal hesabında kullanılacak ara değerdir. Normal olasılık dağılımında verilen bir Z değerinin hangi olasılığa karşı düştüğü tablo şeklinde birçok kaynakta hazır verilir. Bu çalışmada internetteki bir tablo kullanılmıştır¹. Bu tablodan alınan değerle, projenin 35 haftada bitme ihtimali yaklaşık %13 olarak bulunur. Eğer projenin bitmesi istenen süre 41 hafta olarak verilirse Z değeri 0,56 ve bu sürede bitme ihtimali % 71 olarak bulunur.

¹ http://en.wikipedia.org/wiki/Standard_normal_table (Erişim: Haziran, 2012)

Olasılık dağılımı kullanılarak standart sapmanın belli katlarına göre projenin bitme ihtimali de incelenebilir. Farklı standartlar, *PERT* yöntemi için farklı istatistiksel dağılımlar kullanmaktadır. Bu da emin olma derecesini etkiler. Bu kısımda oldukça karmaşık olan matematiksel formüllere girmeden sadece sonuçlar incelenecaktır. Örneğin *PMI eminlik derecelerini* aşağıdaki gibi kabul eder [Newell-2002].

- Kritik yol beklenen değeri $\pm 1 \times$ standart sapma için %66
- Kritik yol beklenen değeri $\pm 2 \times$ standart sapma için %95
- Kritik yol beklenen değeri $\pm 3 \times$ standart sapma için %99

Genellikle % 95 emin olunan ihtimal müşteriye taahhüt edilmektedir. Yukarıdaki değerler PMI tarafından düzenlenen proje yönetimi profesyoneli (*PMP, Project Management Professional*) sınavlarında sorulduğundan özellikle dikkat edilmelidir.

7.3.6. Uzman Yargısı

Yazılım ekibinin yaptığı tahminlerde, daha önce benzer projelerde görev yapmış kişilerin tecrübelerinden gerek kurum içinde gerek danışmanlık kapsamında istifade etmesi uygun olur. Ayrıca proje ekibinde bu şekilde tecrübeli birkaç kişi bulundurmak, projenin ilerlemesini hızlandırır ve birçok sorunu henüz oluşmadan çözer. Uzmanın tahminleri kendi bilgi, beceri ve önceki ekibine göre yapacağı; işi doğrudan yapmayan kişilerin iyimser olmaya meyilli olduğu akılda tutulmalıdır. Bu yüzden konuya farklı açılardan bakacak en az üç uzmandan fikir almak uygun olur.

Uzmanlardan tahmin alınması sadece basitçe uzmana konuyu sorup cevap almaktan ibareت değildir. Uzman tahminlerinin kuralsız ve sezgiye dayalı olması yerine yapısı ve kuralı belli olması, model tabanlı tahminlere yakın doğruluk elde etmeye imkân verir [Jørgensen-2002]. Bu konudaki teknikler *uzman yargısı* (Expert Judgement) başlığı altında toplanır. Bu tür tahminlerde doğruluğun artması için tahmin yapılmadan önce yapılacak işler küçük parçalara bölünerek ayrıntılı hale getirilmelidir.

Bir koordinatör nezdinde farklı mekânlardaki birden fazla uzmanın görüşü alınarak, bunların ortalamasına göre resmi ve prosedürel bir yöntemle tahmin yapılması *delphi tahmini* (*delphi estimate*) olarak standartlaşmıştır. Bu yöntemdeki işlemler:

1. Koordinatör uzmanlara konuyu açıklar ve ayrı ayrı tahminler istenir.
2. Tahminler alınır. Birleştirilerek tekrar uzmanlara gönderilir ve farklılıkların sebeplerini dikkate alarak tekrar inceleme yapmaları istenir.
3. Uzmanlar tahminleri inceleyip tekrar gönderir.
4. Uzlaşma sağlamak için ilk üç işlem koordinatör kılavuzluğunda 3-4 defa tekrar eder.
5. Tahminler yine de birbirine yaklaşmıyorsa süreç geniş aralık ve yüksek belirsizlikle neticeLENİR.

Geniş Bant Delphi şeklinde toplantılarla tahminin olgunlaşmasına yönelik bir yöntem de önerilmiştir [Boehm-1981]. Yukarıdaki ilk anlatılan yöntemde toplantı yerine koordinatörün iletişimini sağlama öngörmüştü. Bu yöntemindeki işlemler:

1. Koordinatör projenin ve ürünün özelliklerini tanımlayan bir tahmin formu hazırlar ve uzmanlara dağıtır. Böylece toplantı öncesi ön hazırlık başlamış olur.
2. Uzmanlar, kişisel tahminlerini hazırlar.
3. Uzmanların tahminleri tartışır ve uzlaşma arayacağı bir toplantı düzenlenir.
4. Uzmanlar, tahminlerini isimsiz bir şekilde verir.
5. Koordinatör, bir özet hazırlar ve kendi kişisel tahminleri ile mukayeseli değerlendirme yapmaları için uzmanlara dağıtır.
6. Tahminler tekrar tartışırlar. İmzasız oylanır. Herkes kabul etmezse 3. adıma dönülür.
7. Tek nokta veya bir aralık olarak son tahmin ortaya çıkar.

Uzmanlara sadece sınır değerler değil aynı zamanda en iyi ve en kötü durumlardaki tahminleri de sorulmalıdır. PERT kısmında da bahsedilen beklenen değer formülleri kullanılarak nihai tahmine ulaşılır. Aşağıda bir projede ekran geliştirme süresiyle ilgili bir *uzman tahmin* örneği verilmiştir. *Süre* gün, hafta veya ay cinsinden olabilir.

	En İyi	Ortalama	En Kötü	Beklenen Değer
Birinci uzman	5	6	10	6,5
İkinci uzman	3	5	6	4,9
Üçüncü uzman	4	7	10	7

Uzman tahmininde basit bir ortalama almak yerine farklılığın sebepleri araştırılmalıdır [McConnel-2006.1]. Yukarıdaki örnekte niçin bir uzman aynı iş için 4,9 hafta tahmin ederken diğeri 7 hafta tahmin etmektedir. Bunun sebepleri tespit edildikten sonra mutabakat oluşturulması daha sağlıklı tahmin yapmayı sağlar.

Uzman tahmini yöntemi toplantılar gerektirdiğinden maliyetlidir. Doğruluk oranı çok yüksek değildir [Jørgensen-2002]. Özellikle yeni veya birçok disiplini içeren alanlara yatırım yaparken genel değerlendirme yapmak için kullanılması önerilir.

7.3.7. SLIM

SLIM (Software Lifecycle Management) Larry Putman tarafından 1970'lerde önerilmiştir. Bu yaklaşımında yazılım geliştirmede harcanan emek ve planlanan süre ilişkisi iki ihtimal denklemiyle gösterilir. *Emek, adam/ay* cinsinden hesaplanır.

$$\text{Emek} = (\text{İstihdam Endeksi}) \times \text{Süre}^3 \quad \text{Norden-Rayleight denklemi}$$

$$\text{Emek} = \left(\frac{\text{Tahmini Sistem Büyüklüğü(Kod Satır Sayısı)}}{\text{Üretkenlik Endeksi}} \right)^3 \times \text{Süre}^{-4}$$

Yukarıdaki iki denklemin çeşitli ihtimal hesabı ve benzetimler kullanılarak çözümüyle projenin en az emek-zaman çiftinin bulunması hedeflenir. Bu denklemlerin çözümü oldukça karmaşık olduğundan ayrıntıya girilmeyecektir. Sadece çözülmüş halleri örneklenecektir.

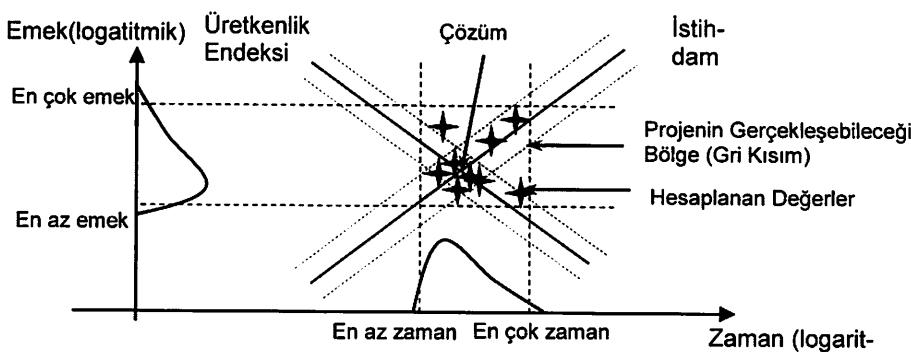
Örneğin *Boehm* bu denklemlere çözüm olarak en küçük geliştirme zamanını tespit için aşağıdaki eşitliği önermiştir.

$$\text{En Küçük Süre} = 2,15x(\text{Emek})^{0,33} \quad [\text{Boehm}-1981]$$

Yukarıdaki yaklaşımı göre 60 adam/ay emek gerektiren bir projenin en kısa bitme süresi, $2,15x(60)^{0,33} = 8,30$ yaklaşık sekiz buçuk ay olarak bulunur. 120 adam ay için bu değer 10,43 ay olarak bulunur. Dikkat edilirse adam ay miktarı iki katına çıkmasına rağmen proje en kısa süresi sadece iki ay artmıştır.

7.3.8. Monte Carlo Benzetimi

Tahmin yöntemleri birçok değişken içerir. Tahminin sağlıklı olması için *değişken değerlerindeki değişimin*, toplam süre ve emeğe etkisini ölçmek gereklidir. Değişken değerlerinin değiştirilip yüzlerce deneme yapılması *benzetim yöntemleri* kullanarak gerçekleştirilebilir. Bu amaçla *Monte Carlo Benzetimi* yaygın kullanılır. Bu benzetim 100 noktalı bir istatistikî dağılım kullanır. Görevlere süre vb. değerler atanarak; her noktada üretkenlik, büyülüklük ve ekip büyümesiyle ilgili bir sonuç üretir. Sonrasında bu üç büyülüklük kullanılarak en az emekle en az zamanı sağlayan tahmin aralığı hesaplanır.



Şekil 7.3. Monte Carlo benzetim neticesi [Fairley-2009]

Değerlendirme aşamasında istenilen sonuca ulaşmak için uygun *değişken değer aralıkları* hesap edilir. Örneğin her hesaplama için bin adet çevrim yapılır. Çözüm, projenin gerçekleşebileceği bölge içinde hesaplanan değerlerin yoğunlaştiği noktalarda belirlenir; bkz. Şekil 7.3. Projenin süre veya ekiple ilgili hedefleri bu aralıkta seçilir. Bu karmaşık hesaplamlar için hazır yazılım kullanılması uygun olur. *Monte Carlo Benzetimi*, *SLIM* denklemi çözümü ve *PERT* dağılım hesaplarında da kullanılır.

7.3.9. *Yapıcı Maliyet Modeli (COCOMO)*

Yapıcı maliyet modeli, 1981 yılında Barry Boehm tarafından önerilmiştir [Boehm-1981]. Sonraki yıllarda (1995) yazılım teknolojisi alanında gelişmelere paralel olarak *COCOMO II* sürümü geliştirildi [USC-1.4, USC-2.1]². Bu bölümde özellikle yeni sürüm açıklanacaktır. Bu yeni sürümde, uygulama yaşam döngüsü ile yazılım ürünü arasında bütünleşme ve kodun tekrar kullanımı konularındaki değişkenlerin önemi artmıştır. Zamanla yazılım geliştirmenin donanıma bağımlılığı azaldığından ve donanım temini kolaylaşlığından bunlarla ilgili değişkenler kaldırılarak, belgeleme ihtiyacı ve tekrar kullanım hedefi gibi uygulamaya yönelik yeni değişkenler eklenmiştir.

COCOMO oldukça karmaşık bir modeldir. Bu modeldeki tüm değişken ve denklemler uygulamada kullanılmayabilir. Ancak *COCOMO modeli* sadece yazılım sürecinde hangi değişkenin nasıl etki yaptığını farklı açıdan değerlendirmek için dahi incelemeye değer!

COCOMO II regresyon analizine dayanır. Önceki proje verilerinden istifade edilerek denklem kurmaya *regresyon analizi* ismi verilir. Bu denklemlerin genel ifadesi:

$$\text{Emek} = a \times (\text{Büyüklük})^b$$

$$\text{Zaman} = c \times (\text{Emek})^d$$

Yukarıdaki denklemlerde sırayla büyülüktен emek, emekten de zaman hesaplanır. Bu denklemlerde a, b, c ve d aranan sabit değerlerdir. Büyüklüğü ifade etmek için genellikle *toplam kod satır sayısı* kullanılır. b değişkeni yazılım geliştirmede emek ile büyülüklük arasındaki, üstel ve doğrusal olmayan ilişkiyi göstermek için kullanılır. *COCOMO* tekniğinde sabitlerin bazıları hesaplamayla bulunurken bazıları da projelerdeki tecrübelерden yola çıkarak doğrudan önerilir.

COCOMO II'de önce büyülüklük hesabı yapılır. Bu yapılrken kodun tekrar kullanımı ve ihtiyaçlardaki değişkenlik hesaba katılır. Ayrıca bilgisayar destekli yazılım geliştirme ortamlarıyla (*ICASE*) yapılan yazılım geliştirmede harcanan emek ve zamanı hesaplamak için geliştirilen bir alt (*Uygulama Bütünleştirme*) model de mevcuttur.

COCOMO II, proje sürecinde gelinen aşamanın tahmine etkisini yansıtacak *Erken Tasarım, Mimari Sonrası ve Bakım* modelleri önermektedir. Bu modeller arasında, *proje büyülüğünün belirlenmesinde* kullanılan giriş değişkenleri ve hesaplama yönteminde çeşitli farklılıklar vardır. Konunun sonunda alt modelleri büyülüklük ve emek açısından birlikte incelmek amacıyla genel bir değerlendirme yapılacaktır.

² Bu kaynaklardan *COCOMO* kısmının genelinde birçok noktada yararlanılmıştır

COCOMO II modelinde projeyi etkileyen faktörler *maliyet kaynağı* (cost driver) ve *ölçek çarpanı* (scale driver-factor) şeklinde ikiye ayrılır. *Maliyet kaynağı* geliştirilen ürün, projeyi etkileyen şartlar, altyapı ve personel uzmanlığını alanlarındaki değişenlerin emek ve süreye etkisini yansıtır. *Ölçek çarpanı*, proje büyüklüğünün etkisi ni yansitmaya yöneliktir.

Maliyet kaynağı sayısı ve katsayıları, kullanılan *COCOMO* alt modeline göre değişir. *Erken tasarım modelinde* 6, *mimari sonrası modelde* 17 adet *maliyet kaynağı* vardır. *Maliyet kaynağı* çok düşükten aşırı yükseğe kadar 6 seviyede değer alır. Bazı çarpanlar 6'dan az seviyede değer alabilmektedir. Bir *maliyet kaynağının* her seviyede aldığı değerler *emek(effort) çarpanı* adını alır.

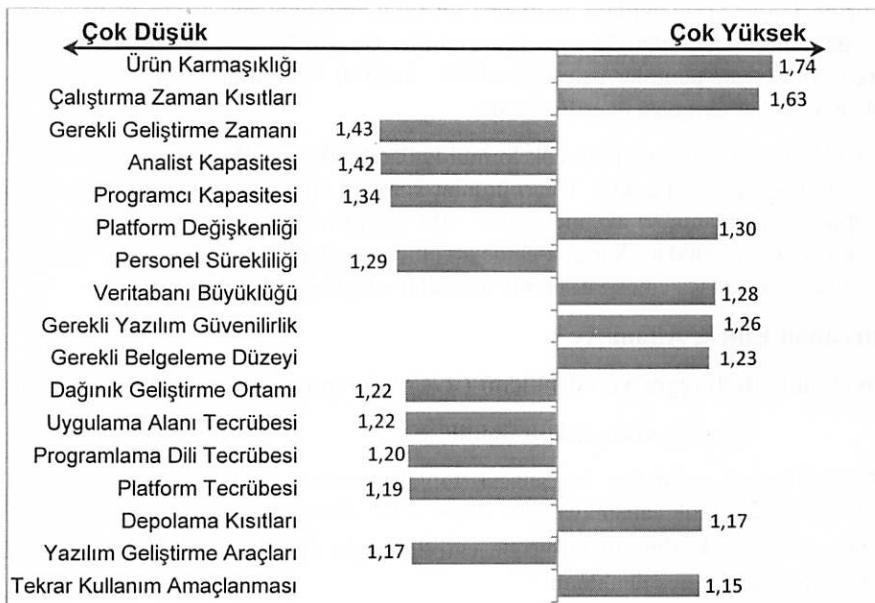
Mimari sonrası modelde kullanılan *maliyet kaynakları* en geniş kümeyi oluşturur. *Erken tasarım modelinde* kullanılan *maliyet kaynak* değerleri de mimari sonrası modeldeki değerlerden hesaplanır. Bu yüzden aşağıdaki tabloda *mimari sonrası modelde* kullanılan *maliyet kaynakları* ve *emek çarpanları* verilmiştir. *Erken tasarım modelinde* kullanılan *maliyet kaynak* tablosu, konusu geldiğinde gösterilecektir.

	Emek Çarpanları	Çok düşük	Düşük	Normal	Yüksek	Çok Yüksek	Çok Aşırı Yüksek
Ürün	Ürün Karmaşıklığı	0.73	0.87	1.00	1.17	1.34	1.74
	Gerekli Yazılım Güvenilirliği	0.82	0.92	1.00	1.10	1.26	-
	Veritabanı Büyüklüğü	-	0.90	1.00	1.14	1.28	-
	Tekrar Kullanım Amaçlanması	-	0.95	1.00	1.07	1.15	1.24
	Gerekli Belgeleme Düzeyi	0.81	0.91	1.00	1.11	1.23	-
Altyapı	Çalıştırma Zaman Kısıtları	-	-	1.00	1.11	1.29	1.63
	Depolama Kısıtları	-	-	1.00	1.05	1.17	1.46
	Altyapı Değişkenliği	-	0.87	1.00	1.15	1.30	-
Kısisel	Analist Kapasitesi	1.42	1.19	1.00	0.85	0.71	-
	Programcı Kapasitesi	1.34	1.15	1.00	0.88	0.76	-
	Personel Sürekliliği	1.29	1.12	1.00	0.90	0.81	-
	Uygulama Alanı Tecrübesi	1.22	1.10	1.00	0.88	0.81	-
	Programlama Dili Tecrübesi	1.20	1.09	1.00	0.91	0.84	-
Proje	Altyapı Tecrübesi	1.19	1.09	1.00	0.91	0.85	-
	Yazılım Geliştirme Araçları	1.17	1.09	1.00	0.90	0.78	-
	Dağıtık Geliştirme Ortamı	1.22	1.09	1.00	0.93	0.86	0.80
	Gerekli Geliştirme Zamanı	1.43	1.14	1.00	1.00	1.00	-

Gerekli Geliştirme Zamanı dışındaki kaynaklar bileşen seviyesine uygulanabilir. Bu çarpan ise sadece proje seviyesinde uygulanır ve diğer kaynaklardan ayrı incelenir.

Değişken değerlerinin ezberlenmesinden ziyade karşılaştırmalı incelenmeler önemlidir. Örneğin emek çarpanı en yüksek olan maliyet kaynağı projeye en büyük etkiyi yapacaktır; bkz. Şekil 7.4. Bu maliyet kaynaklarının *Ürün Karmaşıklığı*, *Çalıştırma Zaman Kısıtları* ve *Gerekli Geliştirme Zamanı* olduğu sekilden kolayca görülebilir.

Emek çarpanının en alt ve üst değerinin oranı bu çarpanın emek ve maliyete ne kadar büyük bir etki yapabileceğini gösterir. Örneğin *Ürün Karmaşıklığının* değişmesi projedeki toplam emege $1.74 / 0.73 = 2.38$ kat etki edecektir.



Şekil 7.4. Maliyet kaynaklarının etkilerinin sıralanması

Proje büyülüklüğü ifade etmek için 5 adet ölçek(büyüklük) çarpanı kullanılır. Hesaplama yapılrken projeye, ekibe ve kuruma uygun seviye seçilir. Bu faktörler çok düşükten, çok aşırı yükseğe uzanan 6 seviyede değer alır.

Ölçek Çarpanları	Çok Düşük	Düşük	Normal	Yüksek	Çok Yüksek	Çok Aşırı Yüksek
Benzeri Yaşanmışlık	6.20	4.96	3.72	2.48	1.24	0.00
Geliştirme Esnekliği	5.07	4.05	3.04	2.03	1.01	0.00
Mimari/Risk Çözümleme	7.07	5.65	4.24	2.83	1.41	0.00
Ekip Kaynaşması	5.48	4.38	3.29	2.19	1.10	0.00
Süreç Olgunluğu	7.80	6.24	4.68	3.12	1.56	0.00

Kurum ve projenin hangi ölçek değerine sahip olduğunu tespiti için ayrıntılı yöntemler önerilmiştir. Anketler ilk önerilen yöntemdir. Ayrıca süreç olgunluğunun, CMMI standarıyla eşleştirilerek aşağıdaki gibi belirlenebileceği ileri sürülmüştür.

Süreç olgunluğu tablosu:

Süreç Olgunluğu	7.80	6.24	4.68	3.12	1.56	0.00
	CMMI 1	CMMI 1*	CMMI 2	CMMI 3	CMMI 4	CMMI 5

COCOMO modelindeki tüm sabitler, uygulanmadan önce kurumun gerçek projelerinden alınan değerlerle düzenlenmelidir. Bunun için *COCOMO* modelindeki emek, süre ve bunların aşamalara dağılımında kullanılan değerler gerçek projelerle karşılaştırılır. Gerçek projelerdeki tanımlar *COCOMO* modelindeki aşama ve her aşamada yapılanların tanımlarıyla aynı olmayıpabilir. Bu yüzden öncelikle *COCOMO* ve gerçek projedeki tanımlar eşleştirilmelidir. Sağlıklı bir kalibrasyon için en az beş noktadan değer alınması önerilmektedir.

COCOMO hesaplaması için birçok formül kullanılmaktadır. Bu yüzden hazır bir ürün kullanılması uygun olacaktır. Ticari ürünler yanında internette bulunabilecek ücretsiz hesaplama uygulamaları da mevcuttur³. Bu uygulamalara proje büyüklüğü, tekrar kullanılacak kod miktarı, kuruma-ekibe ait emek ve ölçek çarpanları girilir. Çıkış olarak emek, süre ayrıca emek ve sürenin aşamalara dağılım elde edilir.

Harcanan Emek: Adam/Ay

Büyükükle ilgili regresyon denklemi *COCOMO* modelinde aşağıdaki şekli alır.

$$AA = A \times (\text{Büyüklük})^E \times EUF$$

AA (*Adam/Ay*) proje için harcanacak toplam emektir. *COCOMO* hesabında *Bir Adam/Ay*; 152 saatlik çalışmaya karşı düşer. *EUF* (*Emek Uyarlama Faktörü*), kullanılan emek değişkenlerinin birbirileyle çarpılmasıyla oluşur ($EUF = \prod E\mathcal{C}$). Böylece *COCOMO* genel emek denklemi

$$AA_{\text{Nominal Zaman}} = A \times (\text{Büyüklük})^E \times \prod_{i=1}^n E\mathcal{C} \quad (1)$$

Burada ilk olarak aranan A , E sabitlerinin değerleridir. Bu değerler *Boehm* tarafından geliştirilen ilk *COCOMO* modelinde 63 proje üzerindeki çalışmalarla aşağıdaki gibi bulunmuştur [Boehm 1981]. Bu modelde projeler 3 ana kategoriye ayrılır. *Uygulama projesi* ki bunlar en az emek gerektirir. Geliştirme araçları kullanılarak müşteri için geliştirilen kurumsal projelerdir. *Yarı bağımlı proje* veritabanı ve derleyici gibi sistem programlarıdır. *Gömülü sistem* ise gerçek zamanlı ve daha büyük bir sistemdir.

Yazılım Projesi	<i>A</i>	<i>E</i>	<i>C</i>	<i>D</i>
Uygulama	2,4	1,05	2,5	0,38
Yarı bağımlı	3	1,12	2,5	0,35
Gömülü	3,6	1,2	2,5	0,32

COCOMO II modelinde ise 161 proje incelenerek $A=2.94$ olarak verilmiştir. Uygulama kategorisi kavramı, ölçek çarpanlarıyla (\mathcal{OC}) değiştirilerek tahmin denklemi aşağıdaki şekilde geliştirilmiştir. Konunun devamında bu çarpanlar kullanılacaktır.

³ <http://diana.nps.edu/~madachy/tools/COCOMOII.php> (Tavsiye değil örnekleme amaçlıdır. Erişim 2012)

$$E = 0,91 + 0,01 \times \sum_{j=1}^5 \text{ÖÇ}_j$$

E değeri küçükse proje büyündükçe maliyet azalır. Bir olması tam denge noktasıdır. Birden büyük olduğunda ki yazılım projeleri hep böyledir, maliyet projenin büyümeye oranından daha yüksek bir oranda büyür. Her şey mükemmelse tüm ölçek faktörleri sıfırdır. Bu durumda *E* değeri 0.91 olur. Ancak tipik projelerde *E* değeri 1.1 ile 1.24 arasında değişkenlik gösterir. Örnek: Aşağıdaki proje için *E* değerinin hesaplanması

Benzeri Yaşanmışlık	Firma önceden benzer uygulama geliştirdiği için	2.48
Geliştirme Esnekliği	Müşteri sürece müdahale olduğundan	3.04
Mimari/Risk Çözümleme	Fazla bir risk analiz yapılmadığından	4.24
Ekip Kaynağması	Hep aynı ekleyle çalışıldığı için	1.10
Süreç Olgunluğu	Firmanın süreçleri belirsizliğinden	6.24
Toplam		17.1

Yukarıdaki tabloda toplam 17.1 olduğundan $E=0,91+0,17=1.08$ olarak belirlenir. Bu değer 100 Bin Satır Kod içeren bir projede uygulanırsa:

- Örnekteki *E* değeri (1.08) için $AA=2.94 \times 100^{1.08}=425$ adam/ay olur.
- Eğer $E=0.91$ olursa (ölçek çarpanları etkisi 0 kabul edilirse) $AA=2.94 \times 100^{0.91}=194$ adam/ay olarak bulunur.
- Eğer proje için tüm ölçek değerleri çok düşük olursa $\sum_{j=1}^5 \text{ÖÇ}_j = 31.6$, $E=1.23$ ve proje süre tahmini $AA=2.94 \times 100^{1.23}=847$ adam/ay olur.

Yukarıdaki ikinci ve üçüncü durum arasında 4 kata yakın bir fark olması ölçek çarpanlarının etkisinin ne denli büyük olabileceğinin kanıdır.

Yukarıdaki örnekte emek çarpanlarının hepsi normal kabul edilmiştir. Bu yüzden $(EUF=\prod E\mathcal{C}) = 1$ olarak bulunmuştur ve proje emek hesabına etki etmemiştir. Eğer *mimari sonrası model* kullanılır ve tüm emek çarpan değerleri *yüksek kabul edilirse* $EUF=0,95$ olur. Bu durumda $E=1.08$ için projedeki toplam emek $2.94 \times 100^{1.08} \times 0,95 = 403$ adam/ay bulunur. Tüm değerler düşük kabul edilirse $EUF=1,46$ ve toplam emek = 620 adam/ay bulunur. Buradaki fark emek çarpanlarının etki düzeyini gösterir. Dikkat edilirse ölçek çarpanları 4 kat etki ederken, emek çarpanları 1,5 kat etki etmiştir. Tüm çarpan değerlerin aynı anda düşük ve yüksek olması gerçek hayatı pek manalı değildir. Sadece örnekleme amaçlı verilmiştir.

Proje Süresinin Tahmini

COCOMO II modeli, toplam emeği proje süresine dönüştüren tahmin eşitliğinde değişiklik önerir; bkz. Kısım 7.3.3. Alt modellerde bazı istisnalar içermekle birlikte önce toplam emek sonra proje süresi hesaplanır. Toplam emek ile toplam zaman ilişkisi:

$$PGS_{\text{Nominal Zaman}} = C \times (AA)^F \quad (2)$$

PGS Nominal Zaman (*Proje Geliştirme Süresi*) ortalama bir ekiple projenin zamanı sıkıştırmadan geliştirme süresidir. Bu aşamada *gerekli geliştirme zamanı* çarpanı dikkate alınmamıştır. Burada aranan C ve F sabitleridir. Bu değerler *COCOMO II* modelinde aşağıdaki gibi verilmiştir.

$$F = D + 0.2 \times 0.01 \times \sum_{j=1}^5 \text{ÖÇ}_j \quad C = 3.67 \text{ ve } D = 0.28 \text{ sabit olarak verilmiştir.}$$

Ölçek çarpanlarının hepsinin değerini çok yüksek alınarak toplam değer ($\sum_{j=1}^5 \text{ÖÇ}_j$) = 6,32 şeklinde hesaplaşın. Bu durumda F değeri 0,29 olarak bulunur. Proje geliştirme süresi de $3,67 \times (60)^{0,29} = 12,03$ Ay olarak bulunur. Bu değer 7.3.3 kısımlarında 60 ay için hesaplanan proje süresine (11,74) oldukça yakındır. Ancak eğer tüm ölçek değerleri çok düşük alınırsa ($\sum_{j=1}^5 \text{ÖÇ}_j = 31,6$) bu durumda hesaplanacak süre $F = 0,34$ olur. Bu durumda toplam proje geliştirme süresi 14,46 Ay olacaktır.

Proje Büyüklüğü Hesaplama Yöntemleri

COCOMO II modeli proje büyüklüğünü için *Kod Satır Sayısını* (bin satır biriminde) kullanır. Fonksiyonel nokta sayısı da kod satırına dönüştürülerek kullanılabilir. Ayrıca *Uygulama Bütinleştirme Modelinde* uygulama büyülüğünü belirlemede kod satır sayısından farklı olarak *nesne nokta sayısı* değişkeni kullanılır.

COCOMO II'de kod satır sayısı sadece yeni yazılan kodları içermez. Buna ilave olarak proje sürecinde ihtiyaçların değişimisinin koda etkisi ve kodun tekrar kullanımı da dikkate alınır. Öncelikle tekrar kullanım da dikkate alınarak proje büyülüğu hesaplanmalı sonrası bu büyülüük üzerinden tahmin yürütülmelidir.

Tekrar Kullanım için Proje Büyüklüğü

COCOMO II modelinde tekrar kullanımın yazılım sürecine etkisi iki tür tekrar kullanılan kod üzerinden incelenir. Tekrar kullanılan (*kara kutu*) kodlar, üzerinde değişiklik yapılmadan doğrudan tekrar kullanılabilir. Uyarlanmış (*Beyaz kutu*) kodlar ise entegre edilmeden önce düzenlenmesi gereken kodlardır. Uyarlanmış kodların tekrar kullanım etkisi üstel olarak kabul edilmiştir. Çünkü küçük miktardaki kod değişikliği için bile büyük emek gerekir. Kodu değiştirebilmek için tümü anlaşılmalı ve tüm modül ara yüzleri kontrol edilmelidir.

1. Otomatik Tekrar Kullanım

Yazılım geliştirirken, yazılmış bir kodu otomatik olarak tekrar düzenleyen (*reengineering*) veya farklı platform kodlarına dönüştüren araçlar kullanılabilir. Böylece çok miktarda kod az bir emekle farklı platformlara taşınabilmektedir. Bu tür kodlara otomatik dönüştürülen kod (*automatically translated code*) ismi verilir. Bu kodlar için:

$$\frac{\text{Değiştirmeksızın}}{\text{Entegre için Harcanan Emek (adam/ay)}} = \frac{\text{Tekrar Kullanılan}}{\text{Tüm KSS}} \cdot \left(\frac{\% \text{Otomatik üretilen KSS}}{100} \right) \quad (3)$$

Mühendislerin Kod Entegrasyon üretkenliği Boehm tarafından ayda 2400 satır olarak verilmiştir. Eğer 100.000 satırlık bir kodun %40'si otomatik olarak oluşturuluyorsa $(100.000 * (40/100))/2400 \sim 17$ adam/ay gibi bir emek gerekecektir.

Kodun değiştirmeden entegre edilmesi, projenin normal geliştirme sürecinden ayrı bir faaliyet olarak düşünülür. *Otomatik dönüştürülen kod*, proje için gerekli toplam emek ve zaman hesabının girişi olarak kullanılan, *projenin toplam eşdeğer kod satır sayısı* büyülüğünün dışındadır. Çünkü sistemin bu kısmı zaten yazılmış kabul edilir. Bu yüzden *otomatik dönüştürülen kod* için harcanan emek de proje için harcanan emek dışında değerlendirilir. Eğer bu tür kodlar eşdeğer kod sayısı içerisinde gösterilirse iki defa eklenmiş olur. Bunu önlemek için *Eşdeğer KSS eşitliğine* ($1 - \text{Otomatik Üretilen}/100$) terimi eklenmelidir.

Yukarıdaki örnekte projenin eşdeğer kod büyülüğü otomatik üretilen kod çıkartıldıktan sonra kalan 60.000 olarak bulunur. Proje ekibinin bu 60.000 satırı geliştirmek için harcayacağı emek hesaplanır ve bu değere 17 adam/aylık emek ilave edilerek proje için gerekli toplam emek bulunur.

2. Uyarlayarak Tekrar Kullanım

Diğer sistemlerden alınan kodun değerlendirildikten sonra değiştirilerek yeni sisteme entegrasyonu, otomatik entegrasyona göre farklıdır ve daha fazla emek gerektirir. Bu durumda öncelikle emeğin değil, tekrar kullanılırken üretilen *eşdeğer yeni kod satır sayısının* hesaplanması gereklidir. *Eşdeğer yeni kod satır sayısı*, uyarlayarak tekrar kullanılan kod miktarı ile bu kodu uyarlamak(değiştirmek) için harcanan emekten yola çıkararak bulunur. Değiştirmek için harcanan emek *Uyarlama Ayar Çarpamı(UAÇ)* olarak isimlendirilir ve aşağıdaki 4 bileşenden oluşur.

- **Değerlendirme:** Tekrar kullanım kararı alınabilmesi için gösterilen çabanın ölçüsüdür. Analiz neticesinde bazı kodların tekrar kullanımına karar verilirken, bazlarının da tekrar kullanımından vazgeçilebilir.
- **Anlama Bileşeni:** Tekrar kullanılacak kodu açıklama ve aşinalık için gösterilen çabadır.
- **Uyarlama Ayar Bileşeni:** Tekrar kullanılan kısımlarda yapılan değişikliklerin maliyetidir. Bu bileşende tasarım, kodlama ve entegrasyon aşamalarındaki değişim maliyetleri hesaba katılmaktadır.
- **Programcı Yabancılığı:** Programının ilgili platformdaki tecrübesini yansıtır.

Uyarlama Ayar Çarpamı hesaplama formülleri ayrıntısına inilmeyecektir. Ancak en kötü durumda değeri %100'ü geçebilir. Yani en kötü durumda kodu değiştirmek, yeni yazmaya göre daha maliyetli olabilir. *Eşdeğer yeni kod satır sayısı* hesaplanırken doğrudan ve emek harcamadan tekrar kullanılan kod miktarı da düşülmelidir.

$$\frac{\text{Eşdeğer yeni KSS}}{\text{Gereken KSS}} = \frac{\text{Tekrar Kullanım İçin}}{\text{Uyarlanması(Değişmesi)}} \cdot \left(1 - \frac{\% \text{Otomatik üretilen KSS}}{100}\right) \times \text{UAÇ} \quad (4)$$

Eşdeğer yeni kod satır sayısı hesaplandıktan sonra tahmin standart denkleminde büyülüklük parametresi olarak bu değer kullanılır. Daha sonra *Eşdeğer yeni kod satır sayısı* üzerinden hesaplanan emeğe, önceki kısımda açıklandığı gibi *Değiştirmeksızın entegre için harcanan emek* eklenerken projede harcanan toplam emek bulunur.

3. İhtiyaçların Değişkenliği

COCOMO modelinde, ihtiyaçların proje süresince değişiminin kod büyülüğüne dolayısıyla emek ve zamana etkisinin hesaplanması gerektiğini ileri sürürlür. Bunu yansıtmak için çeşitli denklemler önerilir:

$$\text{Büyüklük} = \left(1 + \frac{\text{Koddaki Gelişim ve Değişkenlik}}{100} \right) \times \text{Teslim Edilecek Büyüklük} \quad (5)$$

Müşteriye teslim edilecek ürünün toplam kod satır sayısı 100 bin olsun. Geliştirme sürecinde ihtiyaç ve teknoloji değişimi sebebiyle 10 bin satır yeni kod eklensin ve 15 bin satır kod değiştirilsin. Bu durumda; *Koddaki Gelişim ve Değişkenlik* = $25000/100.000=0,25$ olarak bulunur. Proje ekibi aslında 100 bin satır yerine, 125 bin satıra sahip bir proje geliştirecek kadar emek ve zaman harcayacaktır.

COCOMO Emek ve Zaman Tahmin Alt Modelleri

COCOMO II modeliyle, yazılım yaşam döngüsü ve tahmin arasındaki ilişki geliştirilmiştir. Geliştirme aşamasında ilerleyen bilgi düzeyine göre farklı tahmin modelleri kullanımı sağlanmıştır. Bu açıdan bakıldığından *COCOMO II* üç alt modele sahiptir. Bunlar *Erken Tasarım, Mimari Sonrası Model* ve *Bakım Modeli* şeklidendir. Yazılım geliştirme sürecinin farklı aşamalarında farklı emek çarpanları kullanılır.

CASE araçları ile yapılan geliştirmelere yönelik dördüncü bir model olarak *Uygulama bütünlendirme modeli* önerilmektedir. Bu modelde büyülüklük hesabı koda göre değil *neste nokta sayısına* göre yapılır. Emek ve zaman hesaplama yöntemi de farklıdır. Dolayısıyla büyülüklük, emek ve zaman açısından ayrı bir model olarak değerlendirilir.

1. Erken Tasarım Modeli

Projede ihtiyaçların onaylandığı, ancak tasarımın henüz tamamlanmadığı erken aşamalarda kullanılan tahmin modelidir. En önemli faydayı, kullanıcı ihtiyacını karşılayan farklı tasarım seçeneklerini mukayese için kullanıldıklarında sağlar [Sommerwill - 2011]. Bu aşamada amaç, hızlı ve yaklaşık bir maliyet tahmini yapmaktır.

$$\text{AA} = A \times (\text{Büyüklük})^E \times \prod_{i=1}^n E\zeta$$

Erken tasarım modelinde kullanılan emek tahmin denklemi aynıdır. Ancak maliyet kaynakları birleştirilerek n sayısı 7'ye ($6 + \text{Gerekli Geliştirme Zamanı}$) indirilmiştir. Birleştirme işleminde birkaç maliyet kaynağı toplanarak bir araya getirilir. Örneğin bu aşamada kullanılan personel kapasitesi; *mimari sonrası modelde* kullanılan *analist kapasitesi*, *programcı kapasitesi* ve *personel sürekliliğinin* aynı veya yakın seviyedeki emek çarpanlarının çeşitli yüzdelerle toplanmasıyla oluşturulur. (Bu kısımda yeni emek çarpanlarının nasıl oluşturulduğuyla ilgili ayrıntılara inilmeyecektir.)

Erken tasarım modelinde tekrar kullanım için geliştirme ve gerekli geliştirme zamanı maliyet kaynakları için çarpan değerleri mimari sonrası modelle aynıdır. Aşağıda erken tasarım modeli maliyet kaynaklarına ait emek çarpanları verilmiştir.

Emek çarpanları	Çok Aşırı Düşük	Çok düşük	Düşük	Normal	Yüksek	Çok Yüksek	Çok Aşırı yüksek
Personel Kapasitesi	2,12	1,62	1,26	1	0,83	0,63	0,50
Ürün güvenilirliği ve karmaşıklığı	0,49	0,60	0,83	1	1,33	1,91	2,72
Platform Zorluğu	-	-	0,87	1	1,29	1,81	2,61
Personel Tecrübesi	1,59	1,33	1,12	1	0,87	0,74	0,62
Araçlar	1,43	1,30	1,10	1	0,87	0,73	0,62
Tekrar Kullanım İçin Geliştirme	-	-	0,95	1	1,07	1,15	1,24

2. Mimari Sonrası Model

Mimari Sonrası Model, yazılım sürecinde mimari aşamasından sonra geliştirme ve bakım için harcanacak emeğin tahmininde kullanılır. En ayrıntılı modeldir. Burada kullanılan maliyet kaynağı sayısı ($16 + \text{Gerekli Geliştirme Zamanı}$) şeklinde 17 adettir. Bu liste konu başında verilmiştir.

$$AA = A \cdot (S)^E \cdot \prod_{i=1}^{17} EM_i \quad E = B + 0,01 \cdot \sum_{i=1}^5 SF_i$$

3. Bakım Modeli

Bakım *modeli*, proje bakım maliyetiyle ilgili de önerilen modeldir. Bu modelde emek hesabında ürünün tamamı değil, sadece değişen kısmı göz önüne alınır. Var olan kodu değiştirmenin etkisini hesaba katmak için *yazılım anlaşılırlığı* ve *programcı yabancılığı* şeklinde iki ilave çarpan kullanılır. Eğer ürünün %50'inden fazla değişiyorsa bu yeni bir projedir. Bakım olarak görülemez.

$$\text{Büyüklük}_{BAKIM} = \left[\begin{array}{c} \text{Ana kod} \\ \text{Büyüklüğü} \end{array} \times \begin{array}{c} \text{Değişen} \\ \text{Yüzde} \end{array} \right] \times \begin{array}{c} \text{Bakım Uyarlama} \\ \text{Faktörü} \end{array} \quad (7)$$

$$\text{Değişen} = \frac{\text{Eklenen Büyüklük} + \text{Değişen Büyüklük}}{\text{Ana Kod Büyüklüğü}}$$

Bakım uyarlama faktörü değişen kodu anlaşılırlığını hesaba katmak için kullanılır.

$$\text{Bakım Uyarlama} = 1 + \left(\frac{\text{Yazılım Anlaşılırlığı}}{100} \times \text{Programcı Yabancılığı} \right)$$

Bakım Modelinde, gerekli geliştirme zamanı maliyet kaynağı kullanılmaz. Çünkü bakım çevrim süresi sabit kabul edilir. *Tekrar kullanım amaçlanması* maliyet kaynağı da tasarımda bu yönde değişiklikler yapılması yeni proje gibi anlaşılacağından kullanılmaz. *Gerekli Yazılım Güvenilirliği* için de farklı emek çarpan değerleri kullanılır. Bu durumda kullanılan çarpan sayısı 17 değil 15 olarak belirlenir.

$$AA_{Bakım} = A \cdot (\text{Büyüklük}_{BAKIM})^E \cdot \prod_{i=1}^{15} EM_i \quad E = B + 0,01 \cdot \sum_{i=1}^5 SF_i$$

Bakım zamanı belirli bir faaliyettir. Bu yüzden zamanı hesaplamak yerine, bakım için gerekli personelin hesaplanması gerçekleştirilir.

$$\frac{\text{Bakım İçin Gerekli}}{\text{Tam Zamanlı Personel}} = \frac{AA_{\text{Bakım}}}{\text{Sabit Bakım Zamanı}} \quad (8)$$

4. Uygulama Bütünleştirme Modeli

Uygulama Bütünleştirme modeli, var olan bileşen, kod parçacığı(*script*) ve veritabanı tasarımlarını kullanarak prototip hazırlama ve uygulama geliştirme maliyetlerinin hesaplanması sırasında kullanılan modeldir. Bilgisayar destekli yazılım geliştirme ortamlarıyla (*ICASE*) yapılan yazılım geliştirmedeki emeği adam/ay cinsinden hesaplamakta kullanılır. Burada ürünün kapasitesi ve geliştiricinin tecrübeşi öne çıkar.

Uygulama büyülüğünü belirlemeye *Nesne Nokta Sayısı* değişkeni kullanılır. *Nesne Nokta Sayısı* üretilenek ekran, rapor, modül, kod parçacık satırı ve veritabanı programlamada gerekli kod satır sayısı bilesiminden oluşur. *Nesne Noktası Üretkenliği* programcının *ICASE* arasındaki tecrübeyle doğru orantılıdır.

$$AA = \frac{\text{Yeni Nesne Noktası.} \left(1 - \frac{\% \text{Tekrar Kullanılan}}{100} \right)}{\text{Nesne Noktası Üretkenliği}} \quad (6)$$

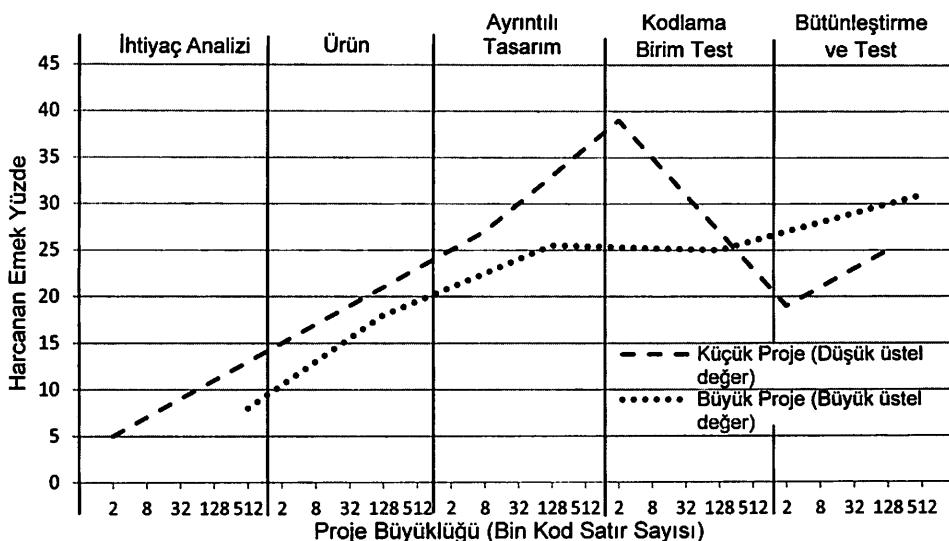
Nense Noktası Üretkenliği geliştirme ekibinin verilen araçta ayda kaç adet nesne noktası üretebileceğini (*Nesne Noktası/Ay*) cinsinde ifadesidir. Burada tekrar kullanımın maliyeti hesaba katılmamıştır.

Ortalama bir programcının Nesne noktası üretkenliği kılavuzlarda 13 olarak verilmiştir. Bu durumda 400 nesne noktalık bir proje hiç tekrar kullanım yoksa $400/13 \sim 31$ adam/ay emek gerektir.

Emek ve Zaman Tahmin Dağılımları

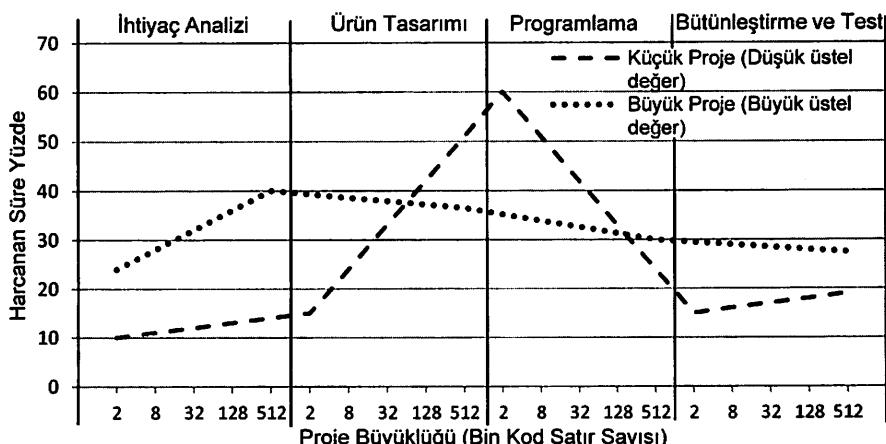
COCOMO modeli sadece harcanan emek ve zamanın hesabında kullanılmaz. Ayrıca emek ve zamanın; proje büyülüğu ve diğer faktörlere göre proje süreci içerisinde nasıl bir dağılım göstereceğini tahmin etmek için de kullanılır. Bu konuda emek, proje büyülüğu, süreç aşaması ve süre dağılım tabloları sunulmaktadır. Tabloların ayrıntılarına girmek yerine grafik üzerinde emek ve zaman dağılımları kısaca incelenecektir.

COCOMO II içinde *şelale* ve *model temelli tasarım-RUP* (Rational Unified Process) şeklinde iki yaşam döngüsü modeli bulunur. Bu kısımda *şelale* modeliyle ilgili yaklaşımlar inceleneciktir.



Şekil 7.5 Şelale Modelinde Harcanan Emeğin Dağılımı

COCOMO II modelinde, Şelale Modeli için yazılım süreci ihtiyaç analizi, ürün tasarım, ayrıntılı tasarım, kodlama ve birim test, bütünlendirme ve test aşamalarından oluşur; b.kz. Şekil 7.5. İhtiyaç analizinde yaşam döngüsü hedefleri belirlenir. Sonraki aşamalarda yaşam döngüsü mimarisi oluşturulur. COCOMO II modeline göre küçük projelerde en büyük emeğin kodlama ve ayrıntılı tasarıma harcadığı görülebilir; b.kz. Şekil 7.5. Büyük projelerde ise en büyük emek bütünlendirme ve test kısmında harcanır.

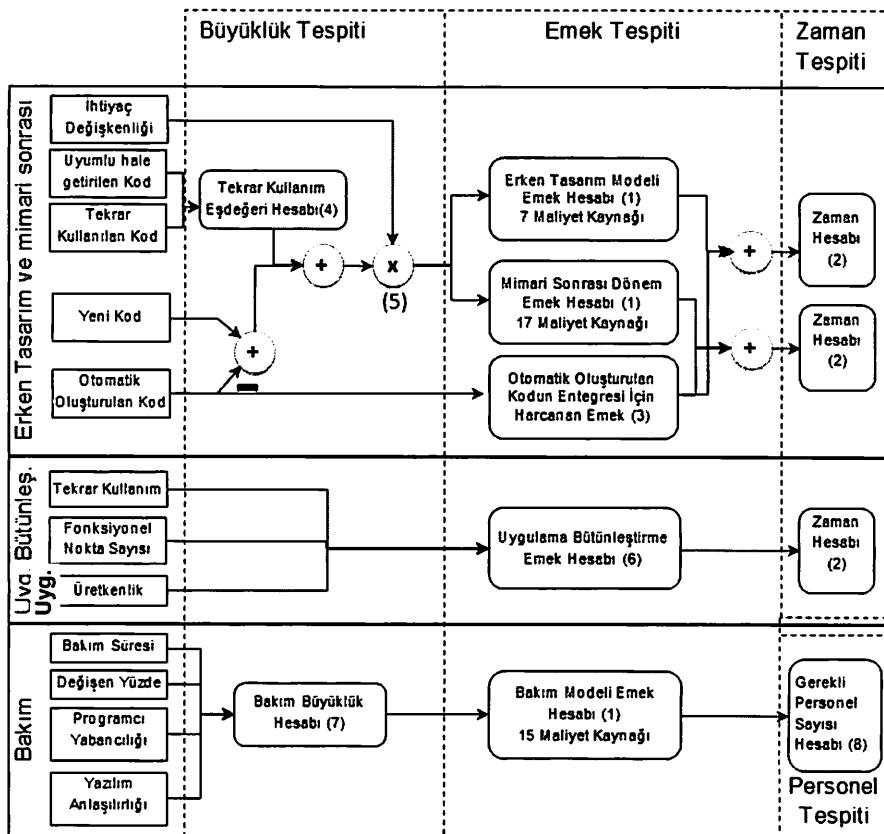


Şekil 7.6. Şelale Modelinde Sürenin Dağılımı

Süre dağılımında ayrıntılı tasarım, kodlama-birim test aşamaları programlama ismi altında birleştirilmiştir. Küçük projelerde analize ayrılan zaman oldukça küçüktür; bkz. Şekil 7.6. Büyük projelerde ise sürenin büyük kısmı analiz ve tasarımda harcanmaktadır. Dikkat edilirse büyük projelerde analize ve tasarıma harcanan zaman fazla ancak emekse düşüktür. Bunun sebebi analiz ve tasarımın küçük ve uzman bir çekirdek ekip tarafından yapılan, fakat zaman alan işlemler olmasıdır.

COCOMO Model Hesaplamaları Bütünleşik Hali

COCOMO modeli birçok alt model ve her alt model için farklı hesaplamalar içermektedir. Farklı hesaplama şekillerinin bütünüşik gösterilimi hangi durumda hangi hesabın yapılacağının anlaşılmasımda kolaylık sağlar; bkz. Şekil 7.7. Şekilde girişler dikdörtgen ve işlemler oval şeklärle gösterilmiştir. İşlemlerin hangi denklemle yapıldığı işlem isminden sonra parantez içerisinde gösterilmiştir. Dikey eksen işlem sırasını yatay eksen ise kullanılan alt modeli gösterir. Hesaplamlarda büyülük, emek ve zaman sırası izlenmektedir. Şekli inceledikten sonra konunun ilgili bölümlerinin tekrar gözden geçirilmesi yapının anlaşılmasını kolaylaşacaktır.



Şekil 7.7. COCOMO alt model ve hesaplamaları

7.3.10. Genel Kabuller ve Endüstri Standartları

Yazılım geliştirme endüstrisinde birçok projede edinilen tecrübeler birleştirilerek, genel yöntem ve standart değerlere ulaşılmıştır. Bunlardan istifade etmek tahmin yapmayı kolaylaştırır. Örneğin yapılan çalışmalarla bin satırda düşen hata miktarının endüstri ortalaması 25 olarak bulunmuştur. Geliştirilen yeni bir yazılım projesinin testinde 100 satırda 10 tane hata bulunması, kaliteli ve hatasız bir yazılım geliştirildiğinden çok 15 tane hatanın bulunamamış olduğunu gösterir [McConnell-2004.1]!

Harici tecrübeler genel olabileceği gibi geliştirilen proje alanına da özel olabilir. Örneğin kurumsal kaynak planlama sistemi geliştirme alanında programcı üretkenliği, endüstri standarı olarak günlük 50 satır kabul edilsin. Proje yöneticisi başka hiçbir bilgi yoksa bu üretkenlik seviyesine göre tahmin yapar. Elbette bu durumda ekipteki kişilerden genel ortalamayı gerçekleştirmeleri beklenecektir.

Tahminde kullanılan standartların yapısı ve büyülüklüğü geliştirmek istenen yeni proje ile benzer olmalıdır. Örneğin kod satır üretkenliğiyle ilgili bir tahmin yapılaçka kullanılan yazılım dili aynı olmalı, referans alınan standartta değişken tanımlama şekli ve açıklama satırı gibi kodu oluşturan parçaların nasıl kullanıldığı dikkate alınmalıdır. Referans alınan standartta açıklama satırları toplam kod satırına eklenmiyor, ancak geliştirilen projede ekleniyorsa tahminler en başından tutarsızdır.

Sanayideki ortalama tecrübeleri kullanmak, çok kısa veya uzun süre gibi uç noktalarda tahminler yapma hatasını azaltır. Örneğin bazı araştırmalarda kurumsal kaynak planlama projelerinin kuruma uyarlanması için ortalama devreye alma süresi 23 ay olarak belirlenmiştir [Umble-2003]. Eğer bir proje ekibi, bu tür bir projeyi 3 ay içerisinde devreye alma iddiasında bulunursa bu iddia geçersizdir!

7.3.11. Tahmin Modellerinin Mukayesesı

Tahmin modellerinin etkinliği, çeşitli araştırmalarla ele alınmıştır [Jørgensen-2002]. Buna göre eğer yapılan işe ve projeye göre özellikle iyileştirilmemişse;

- Uzman tahminleri → endüstri ortalamalarına dayanan tahminlere göre daha doğru sonuçlar üretirler.
- Önceki proje verilerine dayanan tahminler → uzman tahminlerine göre daha doğru sonuçlar üretirler.

Projeyle ilgili bilgi artışı, tahmin doğruluk artışını da beraberinde getirir. Projeye özel yaklaşımlar genel yaklaşımlardan daha doğru sonuç üretir.

Tahmin maliyeti ve tahmin doğruluğu arasındaki ilişkiye de dikkatle incelenmelidir. Maliyeti düşük, doğruluğu yüksek yöntemler tercih edilmelidir. Talep, yapılacak ve geliştirilen ürünün kısımlara ayrılarak incelenmesi hemen her yöntemde yararlıdır. Ancak bölümlemenin ve özellikle buna ihtiyaç duyan uzman tahmini gibi yöntemlerin ilave bir maliyeti vardır.

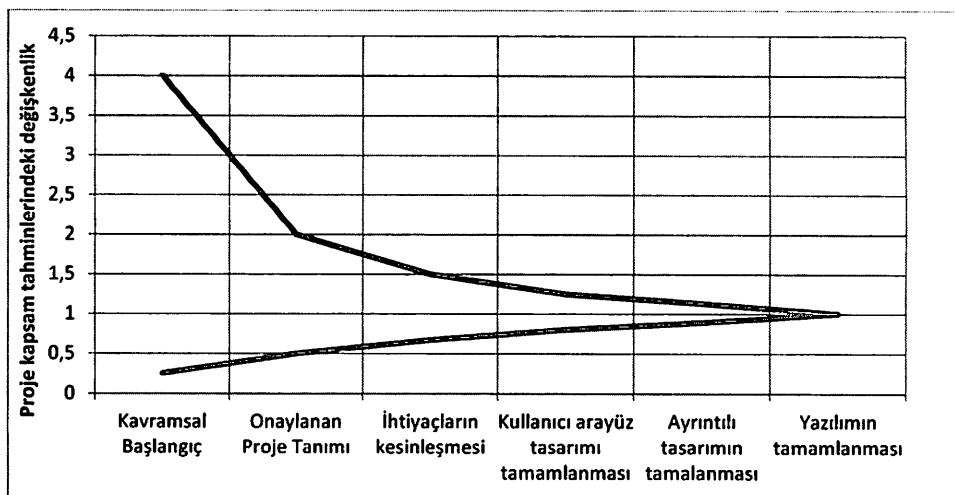
Tahmin doğruluğunu artırmak için aynı projede birden fazla modelle tahmin yapılmalıdır. En az üç model kullanılması önerilir. Çıkan sonuçları değerlendirirken, doğrudan ortalama almak yerine aradaki farkın sebeplerine yoğunlaşmak daha doğru olacaktır. Örneğin bir tahmin 10 ay digeri 30 ay tutuyorsa proje 20 ay sürecek şeklinde ortalama bir tahmin yerine niçin arada 20 ay fark olduğu araştırılmalıdır.

7.4. Projelenin İlerlemesiyle Tahminin Güncellenmesi

Proje geliştirmenin ilk aşamalarında belirsizlikler daha fazladır. Yapılan tahminler bu belirsizliklerden etkilenir. Yeni bilgilere sahip olunmasıyla tahmin edilen ve gerçekleşen değerler karşılaştırılır. Karşılaştırma sonucunda tahminler de güncellenmelidir.

7.4.1. Belirsizlik Konisi

Yazılım geliştirme süreci ilerledikçe yapılacaklarla ilgili bilgiler netleşir ve ayrıntılı hale gelir. İlk aşamada yapılan tahminler geneldir ve doğruluk oranı düşüktür. İlerleyen aşamalarda yapılacak tahminlerde, hatanın azalması beklenir. Bu durum belirsizlik konisi olarak isimlendirilmiştir; bkz. Şekil 7.8 [McConnell-2006.1]. İlk aşamalarda ne kadar çaba gösterilirse gösterilsin belirsizlikleri tamamen yenmek mümkün değildir. Bunun yerine proje gelişmesine paralel olarak tahminlerin güncellenmesi çok daha etkin bir çözümüdür.



Şekil 7.8 Belirsizlik konisi [McConnell-2006.1]

7.4.2. Tahmin ve Gerçekleşen Neticelerin MukayeseSİ

Tahmin edilen ve gerçekleşen değerlerin karşılaştırılması proje performansı, tahmin doğruluğu, tahmin hataları ve hataların sebepleriyle ilgili fikir verir. Projelenin gelecekteki başarısı da bu yöntemle değerlendirilebilir. Proje sonundaki karşılaştırma, projeden gerekli derslerin çıkartma ve sonraki projelerde başarının artması için gereklidir.

Tahminde yapılan hatanın değeri aşağıdaki gibi hesaplanır:

$$\text{Tahminde yapılan hata} = \left| \frac{(\text{tahmin} - \text{gerçekleşen})}{\text{gerçekleşen}} \right|$$

$$\text{Ortalama Hata} = \frac{1}{n} \times \sum_{i=1}^n |Hata_i|$$

Aşağıda örnek olması için bir insan kaynakları sisteminin kullanıcı ara yüz tasarım tahmin ve gerçekleşen süreleri verilmiştir.

	Tahmin Edilen Değer	Gerçekleşen Değer	Hata Oranı
Personel giriş Ekranı	5	7	0,29
Unvan Giriş Ekranı	3	2	(-)0,50
Birim Giriş Ekranı	3	4	0,25
Maaş Giriş	10	15	0,33
Çocuk Bilgileri Giriş	4	2,5	(-)0,60
Toplam / Ortalama	25	30,5	0,39

$$\text{Ortalama Hata} = \frac{1}{n} \times \sum_{i=1}^n |Hata_i| = 0,39$$

$$\text{Toplam Hata Oranı} = \frac{(\Sigma \text{tahmin} - \Sigma \text{gerçekleşen})}{\Sigma \text{gerçekleşen}} = \frac{30,5 - 25}{30,5} = 0,18$$

Önceki projelerde üzerinden hesaplanan hatanın dağılımı yeni yapılacak projedeki hata hakkında fikir verecektir. Hata oranı %18 olduğu durumda yeni projede %18 civarında bir hata yapılması beklenebilir.

PERT yaklaşımıyla yapılan tahminlerdeki hata oranı da çeşitli şekillerde hesaplanabilir. Burada tahmin edilen değer olarak *beklenen değer*(expected value) kullanılır [Conte-1986].

$$\text{Tahminde yapılan hata} = \left| \frac{(\text{tahmini beklenen değer} - \text{gerçekleşen})}{\text{gerçekleşen}} \right|$$

	En İyi	Ortalama	En Kötü	Beklenen	Gerçekleşen	Hata
Personel giriş Ekranı	4	5	9	5,50	7	0,21
Unvan Giriş Ekranı	1	3	4	2,83	2	(-)0,42
Birim Giriş Ekranı	2	3	4	3,00	4	0,25
Maaş Giriş Ekranı	9	10	14	10,50	15	0,30
Çocuk Bilgileri Ekranı	2	4	5	3,83	2,5	(-)0,53
Toplam / Ortalama	18	25	36	25,66	30,5	0,34

PERT metodu kullanımı, ortalama tahmin hatasını genellikle azaltır. Çünkü yazılım projeleri genellikle gecikir ve en kötü tahmin değeri bu gecikmeyi belli oranda yansıtır. Buradaki örnekte ortalama hata 0,39'dan 0,34 seviyesine inmiştir. Toplam hata da %18'den %17'ye gerilemiştir. "Maaş Giriş Ekranı" gibi bazı görevlerde en kötümser tahminin dahi sınırlarını aşan süre uzamaları görülebilir. Bu tür durumların kaynakları da özellikle analiz edilmelidir.

Pareto kuralındaki değerler (80-20) kabul edilebilir hata sınırının belirlenmesi için kullanılabilir. Eğer önceki projede kullanılan hata oranı %20'den fazlaysa bu yeni projede sağlıklı bir tahmin yapılmasını güçleştirir. Önceki projelerde hatalar çok farklı bir dağılım gösteriyorsa tahmin sürecinde ciddi sorunlar vardır. Yazılım geliştirme sürecinin yapı ve yönetiminde de önemli eksiklikler söz konusu olabilir.

Tahmin hatası tahmin yöntemleri arasında mukayese için de önemlidir. Kişi veya yöntemlerin tahmin hataları aşağıdaki örnek tabloda gösterilmiştir. Bu örnekte geçmiş değerlerle karşılaştırma en az hata yapılan yöntemdir. Yeni projelerde, bu yönteme daha çok güvenilebilir.

Yöntem	Ortalama Hata
Uzman Tahmini	%20
Geçmişle Karşılaştırma	%11
COCOMO	%23

Aynı yöntemi kullanarak yapılan tahminlerin kendi içerisinde mukayese edilmesi de önemlidir. Uzman tahminleri arasında yapılacak mukayesede, hangi uzmanın daha doğru tahmin yaptığı belirlenir. Bu uzun vadeli çalışmalara yön verir.

7.4.3. *Tahminin Güncellenmesi*

Projenin ilerlemesi ve yapılacakların netleşmesi ilk başta yapılan tahminlerin tekrar ele alınmasını gerekli hale getirir. Ayrıca ilerlemeye bağlı olarak planda da birçok değişiklik olacaktır. Bu değişiklikler de önceki tahminlerin tekrar düzenlenmesini gerektirir. Tahminlerin sistematik olarak tekrar güncellemesi, tahmin doğruluğunu arttırmır. Böylece projede kontrolü sağlamak kolaylaşır. Tahminin kontrol ve güncellenmesi proje kilometre taşlarına paralel olarak aşağıdaki aralıklarla yapılabilir:

- **Periyodik:** Kontrol için *her 15 günde bir* gibi aralıklar belirlenir.
- **Geliştirme aşamasına bağlı:** Analiz ve tasarım gibi aşamaların başında-sonunda.
- **Tekrar tahmini tetikleyen şartların belirlenmesi:** Analizin değişmesi, teknolojik değişiklikler gibi.
- **Sürümüler:** Artımlı geliştirme yönetiminde her sürüm için yapılacak işler tekrar tahmin edilmelidir.

Tahminlerin tekrar düzenlenmesinde tutarlılık açısından aynı giriş ve çıkışlar kullanılmalıdır. Ancak proje sürecinde farklı aşamalarda farklı tahmin yöntemleri kullanılabilir. Bazı tahmin modelleri proje başında etkiliyken bazıları da ilerleyen aşamalarda daha doğru sonuçlar üretir [McConnell-2006.1].

Proje baştan ne kadar gecikmişse oransal olarak diğer tahminler o kadar uzatılmalıdır. 6 ay sürecek bir projede 4 haftalık ilk kilometre taşı 1 hafta gecikmişse tahminleri proje sonuna sadece 1 hafta ilave edecek şekilde güncellemek hatalıdır. Bu durumda projenin $1/4 = \%25$ uzama ihtimali vardır. Proje tahmini 8 aya çıkartılmalıdır.

Tahminlerin değişmesi üst yönetim ve müşterilere verilen sözlerin de değişmesini gerektirir. Bu değişimin açıklanmasında tahminlerin aralık şeklinde yapılmış olması önemli katkı sağlar [McConnell-2006.1]. Örneğin proje süre tahmini olarak % 80 ihtimalle 9 ay, % 95 ihtimalle 12 ay şeklinde bir tahmin yapılsın. Müşteriye yüksek gerçekleşme ihtimali olan 12 ay söylensin ve proje ekibi kendi içerisinde 9 ayı hedeflesin. Süredeki %10'luk uzama proje süresini 10 aya çıkartacaktır. Proje süresi müşteriye zaten 12 ay olarak söylendiğinden, uzama müşteriye hiç yansımayacaktır.

7.5. Tahmin Araçları

Tahmin yöntemlerinin altyapısındaki algoritma ve benzetimleri, kurum içerisinde geliştirilen yazılımlarla veya elle gerçekleştirmek önemli bir maliyettir. Bunun için tahminine yardımcı birçok yazılım geliştirilmiştir⁴. Bu ürünlerin ortak özellikleri aşağıda kısaca listelenmiştir [McConnell-2006.1]:

- **Farklı projelerin mutabakatı:** Önceden yapılmış birçok farklı projedeki tahmin ve büyülüklük değerleri, merkezi olarak saklanarak yeni yapılacak proje tahmininde kullanılabilir. Tahmin yazılımlarıyla proje süresini üstel etkileyen büyülüklük gibi değerlerin uyuşturulması da mümkündür.
- **Benzetim yeteneği:** Farklı giriş ve çıkış değerleri için, süre ve emek tahminleri hesaplanması ve grafik ortamda gösterilmesi özellikleidir. Binlerce döngü gerektiren bu benzetimlerin elle yapılması mümkün değildir.
- **İstatistik analiz:** Tahminin yapılması belli istatistikî dağılımlara dayanır. % 90 ihtimalle 16 ay, % 80 ihtimalle 14 ay gibi çeşitli ihtimaller ortaya çıkar. Tahmin araçları bu hesapları otomatik yapmayı sağlar.
- **Gerçekçilik:** Tahmin aracı, insanlardan daha gerçekçi olabilmektedir. Hiç kimse belli uzunluktan fazla bir tahmini söylemek istemez. Ancak tahmin aracı bu konuda aracılık edebilir.
- **Unutulabilecek kısımları tahmin:** Test sayısı ve belge uzunluğu gibi az kullanılan değerler de bu araçlarla tahmin edebilmektedir.
- **Ya olursa(What if) analizi:** Tahmininin dayandığı varsayımları değiştirilerek bunun sonucu etkisi hesaplanabilir.
- **Artımlı geliştirme yöntemlerini uygulayabilme:** İhtiyaçları sürümlere bölme, sürümler temelli büyülüklük ve emek tahmini yapabilme özellikleidir.
- **Risk analizi:** Tahmin gerçekleşme riski üzerinden analiz yapılabilmesidir.
- **Aynı projede farklı yöntemleri karşılaştırma:** Birden fazla yöntem uygulayarak alınan sonuçları mantıksal mukayese edebilmeye özelliğidir.
- **Endüstri ortalamaları:** Birçok tahmin aracı, endüstride benzer işlerin ne kadar sürdüğü bilgisini içinde bulundurur.

⁴ <http://www.projectmanagementguides.com/tools/project-estimation-tools/> (Erişim: Eylül, 2012)

- Tahmin hata dağılımı:** Tahminde oluşabilecek hata oran ve dağılımlarını gösteren bilme fonksiyonu sunulur.
- Yazılıma ait özel faktörleri kullanabilme:** Nesneye yönelik tasarım veya veritabanı tablo değerleri gibi özelliklerin tahmin algoritmalarına dâhil edilebilmesi mümkündür.
- Raporlama:** Tahminleri farklı açılardan değerlendirmek için hazır rapor ve grafikler sunulur.
- Proje yönetim araçlarıyla bütünlleşme:** Yapılan tahminlerin proje yönetim araçlarına otomatik aktarılması ve taahhüde dönüşmesi sağlanabilir. Verilen sözlerin hangi tahmin oranına karşı düşlüğü de saklanarak geriye dönük analizler yapılabilir.

7.6. Özet

Tahmin, eldeki analiz, tasarım gibi bilgilerle projenin emek, süre ve maliyet değerlerini öngörmektedir. Tahmin ve planlama arasında çok yakın bir ilişki vardır. Sağlıklı tahmin sağlıklı planlamayı ve sağlıklı planlama da sağlıklı ilerlemeyi getirir. Tahminden sapan plan önemini yitirir ve kişiler kendi başlarına çalışmaya başlar. Her tahmin belirsizlik içerir. Proje yöneticisi paydaşlara belirsizlikle ilgili bilgi vermelidir.

Tahmin analitik yöntemlerle sistemli olarak yapılmalıdır. Tahmin sayılabilen değişkenlere, bunlar bulunamıyorsa dolaylı hesaplamalara dayanmalıdır. Hiçbirisi yoksa kişisel yargılarla tahmin yapılır. Aşırı kötümserlik ve özellikle de aşırı iyimserlikte kaçınmak gereklidir. Çözümün anlamlı parçalara ayrılması, sınıflama ve önceki projelerin incelenmesi, tüm yöntemlerin temel adımlarıdır. Projede yapılacak tüm işler tahmin konusudur. Küçük de olsa hiçbir adım atlanmamalıdır.

Tahmin yöntemleri önceki projelere benzetme ve mukayese, mevcut proje verisini kullanma, ortak değerlere çevirme, *PERT*, uzman yargısı, *SLIM*, *COCOMO* ve genel endüstri değerleriyle mukayese olarak sayılabilir. Proje tahmini için önceki projelerde edinilen tecrübeler en önemli araçtır. Önceki projelerin süresi, görevlendirilen kişi sayısı, harcanan kaynak ve üretilen yazılımlar, yeni projeler için yol göstericidir. Bu tecrübelere istifade edebilmek için bunları yazmak ve saklamak gereklidir.

Bazı tahmin modelleri, belli yazılım problemlerini hedeflerken bazıları da tüm süreçlerde kullanılabilirler. Hangi durumda hangi modelin kullanılacağıının seçimi çok önemlidir. Tahmin modelleri içerisinde *Monte Carlo* gibi benzetimler kullanılarak proje emek ve süre ilişkileri farklı giriş-çıkış çiftleri için incelenebilir.

PERT yöntemi en iyi, ortalama ve en kötü durumların projeye etkisini yansıtarak daha doğru tahmin yapmaya yardımcı olur. Ayrıca projenin ne kadar süreceğle ilgili ihtimal hesaplarında da kullanılır. *Uzman yargısı* yönteminde birçok uzmanın tahmini sistematik bir süreçle incelenerek doğru tahmine ulaşmaya çalışılır. *SLIM* yöntemi matematiksel denklemlere dayalı tahminler yapmayı sağlar.

COCOMO, proje geliştirme sürecinde harcanan emek, zaman ve bu büyüklüklerin proje aşamalarına dağılımının hesaplanması için önerilen çok kapsamlı bir modeldir. Öncelikle tekrar kullanım da dikkate alınarak, proje için gerekli toplam emek hesaplanır. Sonrasında gelinen aşamaya göre erken tasarım veya mimari sonrası alt modeli kullanılarak süre tahmini yapılır. Bu tahminlerde proje, temelde 17 maliyeti kaynağı ve 5 ölçek çarpanıyla karakterize edilmektedir. Bunlara ilave birçok farklı değişken de kullanılır. *COCOMO* yöntemiyle *ICASE* ürünleriyle yapılan geliştirme için de emek ve zaman tahmini yapılabilir. *Bakım* için de bir alt model vardır.

Tahmin yönteminin karmaşık ve çok fazla değişkene sahip olması doğru sonuçlar üreteceğini kanıtlayamaz. Karmaşık matematiksel modellerin sonuçları önceki projelerdeki tecrübelерden daha faydalı değildir.

Projenin ilk aşamalarında belirsizlikler daha fazladır ve yapılan tahminlerin başarısını etkiler. Başta yapılan tahminler ortaya çıkan ihtiyaçlar ve değişimlerle birlikte güncellenmelidir.

Tahmin, karmaşık ve maliyetli bir işlem olduğundan hazır tahmin araçları kullanımı tavsiye edilir. Bu araçlarla bilgi merkezi hale getirilir, projeler karşılaştırılabilir, benzetim, risk analizi ve çok çeşitli raporlamalar gerçekleştirilebilir.