YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM PROJELERİ
ÖLÇÜM SONUÇLARI VERİTABANININ OLUŞTURULMASI
VE
YENİ YAZILIM PROJELERİNİN MALİYET TAHMİNİNDE KULLANIMI

Bilgisayar Yük. Müh. Murat AYYILDIZ

Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalında Hazırlanan

DOKTORA TEZİ

Tez Savunma Tarihi : Şubat 2007
Tez Danışmanı : Prof. Dr. Oya KALIPSIZ (Yıldız Teknik Üniversitesi)
Jüri Üyeleri : Prof. Dr. Coşkun SÖNMEZ (Yıldız Teknik Üniversitesi)
: Doç. Dr. Selim AKYOKUŞ (Doğuş Üniversitesi)
: Prof. Dr. Servet BAYRAM (Kültür Üniversitesi)
: Yrd. Doç. Dr. Banu DİRİ (Yıldız Teknik Üniversitesi)

İSTANBUL, 2007

İÇİNDEKİLER

KISALTMA LİSTESİ

YSA	Yapay Sinir Ağı
VTYS	Veri Tabanı Yönetim Sistemi
FP	Function Point
İP	İşlev Puan
LOC	Line of Code
IEEE	Institute of Electrical and Electronics Engineers
YEEM	Yıldız Effort Estimation Metrics
TCE	Top-Down Cost Estimation
COCOMO	Constructive Cost Model
CMM	Capability Maturity Model
MLP	Çok Katmanlı Algılayıcılar (*Multi-Layer perceptron*)
RNN	Geri Dönüşümlü Ağlar (*Recurrent Neural Networks*)

ŞEKİL LİSTESİ

ÇİZELGE LİSTESİ

ÖNSÖZ

Bu tez çalışması süresince sonuca gitmede değerli yönlendirmelerini, desteklerini, yardımlarını esirgemeyen değerli hocam Sayın Prof. Dr. Oya KALIPSIZ'a, yönlendirme, fikir ve desteğini esirgemeyen değerli hocalarım Sayın Prof. Dr. Coşkun SÖNMEZ'e ve Sayın Doç.Dr. Selim AKYOKUŞ'a, özellikle teknik anlamda büyük destek olan Sayın Y.Doç.Dr. Sırma YAVUZ'a, tez yazımında ve somutlaştırmada düzeltmeleri, yönlendirmeleri ve desteklerinden dolayı Sayın Prof.Dr. Servet BAYRAM'a ve Sayın Y.Doç.Dr. Banu DİRİ'ye, ilk günden beri her konuda destek olan anneme, babama ve sürekli desteğini esirgemeyen eşime ve kızlarıma ve son olarak da benden desteğini esirgemeyen herkese sonsuz teşekkürlerimi sunarım.

Ocak 2007 Murat Ayyıldız

ÖZET

Bilgisayar yazılımlarında maliyet tahmini son zamanlarda oldukça önem kazanmıştır. Yazılım geliştirme maliyeti her geçen gün artmaktadır. Yazılım maliyet tahmini hem devletler, hem de organizasyonlar için çok önemli bir problemdir. Planlanan zaman ve bütçeyi aşan çok sayıda proje mevcuttur. Bunun temelinde baştan bütçe ve zaman tahminini doğru yapamamaktan kaynaklanan başarısızlıklar yatmaktadır.

Yazılım geliştirme giderek pahallılaşmakta ve bilgi sistem bütçelerinin içinde büyük bir maliyet faktörü olmaktadır. Yazılım geliştirme maliyetleri ölçüm ve kestirim metodolojilerinin yokluğundan dolayı sık sık kontrol dışına çıkmaktadır. Geçen on yıl içinde, bazı araştırmacılar maliyet tahminleme konusunda çalışmışlardır ancak sonuçlar tatmin edici olmaktan uzaktır.

Bu çalışmada yeni bir yazılım ölçüt kümesi oluşturma, oluşturulan yazılım ölçüt kümesi için veri toplama ve bu veri kümesi ile bir yapay sinir ağı kullanılarak yazılım maliyet tahmini modeli geliştirme gerçekleştirilmiştir.

Bu çalışmada, yapay sinir ağı temelli yazılım maliyet tahminleme uygulama sonuçları, başarısızlık nedenleri incelenmiş ve bir yapay sinir ağıyla yeni hazırladığımız ölçüt kümesini kullanarak bir model oluşturulmuştur. Yazılım maliyet tahminleme çalışmalarında ölçüt kümesi seçiminin hayati bir rolü vardır. Özellikle yapay sinir ağı bazlı çalışmalarda ölçüt kümesinin seçiminin önemi göz ardı edildiği görülmüştür. Çalışma sonucu oluşturulan model ile elde edilen sonuçlar, geleneksel ölçütler kullanarak yapılan önceki çalışma sonuçlarıyla karşılaştırılmıştır. Karşılaştırmayı yapabilmek için iki tip veri kullanılmıştır. Birinci kısım önceki çalışmalarda genellikle kullanılan yapıcı maliyet modeli verileri (COCOMO : Constructive Cost Model) ve ikinci kısım olarak yeni oluşturulan ölçüt kümesine uygun olarak Türkiye'de bulunan uluslararası bir firmadan toplanan veriler kullanılmıştır. Yazılım maliyet tahminleme çalışmalarında olan bir diğer zorluk veri toplamanın zaman ve dikkat gerektirdiği gerçeğidir. Üstelik reel sektörde rekabet açısından dezavantaj oluşturma olasılığı nedeniyle, birçok kurum bu konuda topladığı verileri eğitim amaçlı da olsa kullandırmayı uygun görmemektedirler.

Literaturde yaptığımız incelemede yapay sinir ağı kullanarak yapılan maliyet tahminleme yöntemleri çalışmalarında genellikle MLP kullanıldığı saptanmıştır. Bu tez çalışmasında Yapay sinir ağı kullanılarak yapılan yazılım maliyet tahmini modeli oluşturmak için MLP (Multi Layer Perseptron) ve bu alanda kullanılmayan *Elman* yapay sinir ağı modeli uygulanmıştır. Kıyaslamayı sağlayabilmek için MLP ve *Elman* modellerinin ikisinde kullanımında COCOMO 81 ve yeni oluşturulan ölçüt kümesi YEEM (Yıldız Effort Estimation Metrics) kullanılmıştır. Hatasız karşılaştırabilmek için veri kümelerinde eşit sayıda örnek ile test edilmiştir. Daha iyi araştırma adına yeni oluşturulan veri kümesi için daha büyük bir küme ile de çalışmalar yapılmıştır. Ölçüt kümelerinin karakteristiği ve veri miktarı bu çalışmanın incelediği konulardan biridir. YEEM ölçüt kümesi yapay sinir ağı topolojisini oluşturmakta kullanılmıştır. YEEM için toplanan veriler ile MLP ve *Elman* yapay sinir ağları kullanılarak ağ eğitilmiş ve test edilmiştir. Toplanan verilerin daha bütünsel kullanımı için çapraz onaylama metodu kullanılmıştır. Toplanan verinin sınırlı sayıda olması nedeniyle %5,%10 ve %15'lik çapraz onaylama teknikleri uygulanmıştır.

Doğru ve nitelikli bir ölçüt kümesi kullanıldığı sürece yapay sinir ağları yazılım maliyet tahminleme çalışmalarında başarıyla kullanılabileceği görülmüştür.

ABSTRACT

Cost estimation of computer software is getting more important. Software development becomes increasingly expensive. Software cost estimation is a very important problem for governments and organizations. There are a lot of projects which exceeded planned budged and time. Incorrect budget and time planning at the beginning is the main failure reason.

Software becomes increasingly expensive to develop and is a major cost factor in any information system budget. Software development costs often get out of control due to lack of measurement and estimation methodologies. During last decade, some researches on cost estimation have been conducted. In the search of new methodologies to estimate the software development costs but the results were far from being satisfying.

In this study, a new software engineering metric set was developed and the data was collected according to new metric set and a new software cost estimation model was developed by using neural network.

In this study we have explored the reasons of the disappointing results of the existing software cost estimation with neural network studies and implemented different neural network models using augmented new metrics. The metric-set selection has a vital role in software cost estimation studies; its importance has been ignored especially in neural network based studies. The results obtained are compared with previous studies using traditional metrics. To be able to make comparisons, two types of data have been used. The first part of the data is taken from the Constructive Cost Model which is commonly used in previous studies and the second part is collected according to new metrics in a leading international company in Turkey. Another difficulty associated with the cost estimation studies is the fact that the data collection requires time and care. Futhermore, many companies do not share their data because of competition disadvantage possibilities.

In recent literature, we have explored that MLP had been used in software cost estimation with neural network studies. In this study, software cost estimation by using neural network models presented here are based on Multi-Layer Perceptron (MLP) and Elman neural networks which has not been used for this aim. The results obtained are compared with previous studies using traditional metrics. The models presented here are based on Multi-Layer Perceptron and Elman Networks for both COCOMO'81 metric set and for the augmented metric set (YEEM : Yıldız Effort Estimation Metrics). To be able to compare the results accurately tests were run for datasets containing the same number of samples. To investigate further, we have also experimented with larger datasets formed according to augmented metrics. Addressing the issues of the dataset characteristics and the amount of samples in the datasets is one of the purposes of this research. YEEM has been used to construct the neural network topology. The MLP and Elman neural networks has been trainned and tested by using data for YEEM. To make a more thorough use of the samples collected, k-fold, cross validation method is also implemented. Since the amount of samples we have collected are still limited, 5-fold, 10 fold and 15-fold cross validation techniques have been also applied.

It is concluded that, as long as an accurate and quantifiable set of metrics are defined and measured correctly, neural networks can be applied in software cost estimation studies with success.

1. GİRİŞ

Ölçemediğini kontrol edemezsin.

Tom Demarco, 1982

Bilgisayar yazılımlarında maliyet tahmini son zamanlarda oldukça önem kazanmıştır. Çünkü yazılım geliştirme maliyeti her geçen gün artmakta ve yazılım geliştirmek için daha fazla harcamalar yapılmaktadır. Yazılım maliyet tahmini hem devletler için hem de organizasyonlar için önemli bir problemdir. Planlanan zaman ve bütçeyi aşan oldukça çok sayıda proje mevcuttur. Bu aşmaların çoğunun temelinde baştan bütçe ve zaman tahminini doğru yapamamaktan kaynaklanan başarısızlıklar yatmaktadır.

”Yazılım mühendisliği” bir yazılım ürünü inşa etmek için kullanılan teknikler topluluğunu tanımlamak için kullanılan bir terimdir. Burada ”mühendislik” yaklaşımıyla yönetim, bütçeleme, planlama, modelleme, tasarlama, uygulama, test etme ve bakım konularını içine almaktadır. Yazılım projeleri geliştirirken ölçülebilir hedefler koymakta başarılı olunamamaktadır (Fenton N. ve Pfleeger S., 1997). Örneğin bir yazılıma başlarken onun “güvenilir”, “bakım yapılabilir”, “kullanıcı dostu” gibi olacağı söylenmekte, net tanımlar verilmemektedir. Ölçme işinin açık ve net hedefleri olmalıdır. Yazılım mühendisliğinde ölçütlerin kullanımı teorik ve pratik çalışmalar arasındaki aralığı küçültecek bir anahtardır. Ölçme ve değerlendirme yapabilmek için sistematik ölçüm yapmak gerekmektedir.

Yazılım proje yönetiminde en önemli unsur projenin zamanında yetişmesidir. Mühendisliklerde eski bir kural geçerlidir: 80-20 kuralı. Bir projenin %80’lik kısmı (beklenen işlevselliğin veya tahmin edilen ürün büyüklüğünün % 80'i) proje süresinin %20 sinde tamamlanır. Geriye kalan %20'lik iş ise zamanın % 80'ini alır. Yazılım mühendisliğinde ise benzer kuraldan biraz ayrıntı katılarak söz edilebilir: 40-20-40 kuralı olarak değişen bu kural, toplam çabanın %40'ı kodlama öncesi, kalan %40'ı ise bu ilk tanımlama işlemine karşı düşen test çabalarına ayrılacağını belirtmektedir. %20 ise kodlama çabasının payıdır. Ayrıca erken evrelerde kalite için yapılacak yatırım, bu dağılımlarda hesaba katılmayan bakım çabasını azaltacaktır.

Zamanlama ile ilgili önemli bir nokta da proje zamanını uzatarak kazanılacak toplam çaba miktarıdır. Ayrıca bu kazanılmış vaktin oluşturacağı ikincil yararlar da kalite göstergelerini etkiler. Bu gerçeği Putnam formülü (1.1) ile gösterebiliriz:

$$\text{Çaba} = ((\text{Satır Sayısı}) \times (\text{ÖY}^{0.333} / \text{VF}))^3 \times (1 / \text{PS}^4) \quad (1.1)$$

Burada 'PS' proje süresidir. ‘ÖY’ özel yetenekler katsayısıdır, projeler büyüklüğüne bağlı olarak 0.16 ile 0.39 arasında değmektedir. 'VF' ise 2000'in altında değerlerden başlayan 30000’e kadar ulaşabilen verimlilik faktörüdür.

Diyelim ki yüz bin satır büyüklüğünde bir proje için 36 adam-ay çaba tahmin edilmiştir. 8 kişi ile bu proje 4 yılda tamamlanabilir. Ancak süreyi 5.25 yıla çıkarma şansımız varsa, Putnam formülünün (1.1) ortaya koyacağı sonuca göre toplam çaba 36 adam-ay'dan 11.5 adam-ay'a düşecektir. Yani %31 süre artırımı sayesinde %68 daha az çaba gerektiriyor. Bu sonuç küçük bir süre artırımının sağlayacağı büyük rahatlığı göstermektedir.

Bir iş için gereken çabayı baştan bilmek çok önemlidir. İşin büyüklüğünü bilmeden işleri yönetmek mümkün değildir. Yönetmek için işin maliyet tahminlerini kullanıp projeleri baştan kabul etmek veya reddetmek çok daha faydalıdır. Çok uzun sürebilecek ve çok kaynak gerektirebilecek bir yazılım geliştirme işine girmemek veya girilme kararı verilse bile baştan gerekli iş gücünü organize etmek verimi her aşamada artıracaktır. Bugünün dünyasında yazılım maliyet kestiriminin önemi sürekli artmakta ve yayılmaktadır. Bu konuda yapılmış çalışmalar olmasına rağmen hala bu konu tam olarak aydınlatılabilinmiş değildir ve üzerinde birçok çalışma yapılması gerekmektedir. Bu konu şu an ihtiyaç duyulandan çok daha yavaş ilerlemektedir. Ölçüm yapabilmeyi başarabilmenin içinde bulunan temel problem ölçülmesi gerekenleri ortaya koymaktır.

Bu çalışmanın en önemli amacı daha isabetli maliyet tahmini yapabilecek bir model geliştirmektir. Maliyeti artıran ancak sonuca olması gerekenden daha az etkisi olan etkenlerin bulunması ve bu sayede mümkün olduğunca maliyetin azaltılması durumları incelenmiştir. Yazılım mühendisliğinde başarılı proje kapsam, zaman ve maliyet hedeflerini yakalamış olan projedir. Bu çalışma maliyet tutturma ve dolayısıyla zaman tahmini yapıp buradaki başarımı artırma konusunda bir fayda sağlayacaktır.

1.1 Yazılım Mühendisliğinde Yaşanan Problemler

Yazılım projeleri zaman planına ve bütçe planına uymama yönünde kötü ün yapmış projelerdir. Bunun temel nedeni gereken iş gücünü baştan yanlış tahmin etmektir. Bütün planları bu yanlış tahminler üzerine kurmakla devam eden bu yanlışlıklar zinciri yüksek maliyetlere neden olmaktadır. Yazılım mühendisliğinde yazılım projeleri için gereken iş gücünü, zamanı ve bütçeyi tahmin etme yönünde yapılacak çalışmalar bu problemleri azaltacaktır.

Yazılım maliyet tahminlemesi bir yazılım sisteminin hazırlanması için gereken çaba miktarının tahminleme sürecidir. 1990'larda Standish Group 8000 yazılım projesini incelemiştir (The Standish Group, 1995). Hem kamu hem de özel projelerden oluşmakta olan bu projelerin %90'ının bütçe ve zaman aşımı nedeniyle başarısız oldukları görülmüştür. Bunların %33'ü daha tamamlanmadan iptal edilmiştir. Yaklaşık üçte biri ilk tahmin edilen ve bütçelenen maliyeti %150 ile %200 arasında ve ortalama olarak %189 aştığı, yaklaşık üçte birinin de %200 ile %300 arasında ve ortalamada %222 aştığı gözlemlenmiştir.

Görülmektedir ki yazılım geliştirme projelerinin maliyet tahminlemesi konusunda ciddi problemler ve yüksek maliyetler vardır. Bu yüzden bu konuda çok sayıda değişik çalışmalar yapılmalıdır.

1.2 Çalışmada Gerçekleştirilenler

Diğer mühendislik dallarının tersine, yazılım mühendisliğinde ölçüm yapabilmek genelde zordur ve kesin olamamaktadır. Proje yönetiminde çok önemli olacak ölçme ve bu kavramın üzerinde kurulan tahminleme yöntemleri aracılığı ile maliyet, zaman ve işgücü planlamaları yapılabilmektedir. Ayrıca kaliteye etki eden hataların ve aksamaların yoğunluğu da çok önemli bilgilerdir.

Bugün için karşılaşılan en büyük sorunlardan biri yazılım geliştirme işinin maliyetinin tahminidir. Bu risk yönetimi ve bütçe yönetimi için oldukça kritik bir nokta haline gelmiştir. Günümüzde kullanılan tahminleme modelleri ve yöntemleri genelleştirilmiş ve sadece bir yazılım geliştirme ortamı için düşünülmüş yöntemlerdir. Bu nedenlerden dolayı, bu çalışma yeni bir ölçüt kümesi ve yapay sinir ağı kullanarak bir yazılım geliştirme maliyet tahmini yöntemi ortaya koyabilmeyi hedeflenmektedir. 1.3'te belirtildiği gibi bu maliyet tahminini ortaya koyarken yeni bir ölçüt kümesi oluşturulmuş ve bu ölçüt kümesi üzerine yapay sinir ağı inşa edilmiştir.

Birçok yerde yerel tahminleme yöntemlerinin daha başarılı olabilecekleri öğütlenmiş ve genel bir çözüm bulmanın zor olacağı ve başarısının düşük olduğu belirtilmiştir. Bu bağlamda genel bir çözüm yöntemi bulma adına yazılım geliştirmenin yapıldığı organizasyonu da girdi olarak almak çalışma başarımını artıracağını düşünmekteyiz.

Yazılım mühendisliğinde yazılım geliştirme maliyeti, yazılımın güvenilirliği, programcıların verimliliği gibi bazı önemli parametrelerin bulunmasında tahminleme metotları kullanılır. Bu tahminleme metotları üzerinde araştırmalar yapılmıştır. Bu metotlarda genellikle

matematiksel bir fonksiyon üzerine inşa edilmiştir.

Eşitlik 1.2'de gösterilene benzer formülizasyonlar kullanılmıştır. Bunun yanında yapay zekâ, benzeşim bazlı sonuç çıkarım, regresyon ağaçları ve kural çıkarım modelleri de kullanılmıştır.

$$\text{İş} = a \times (\text{büyüklük})^b \quad (1.2)$$

Yaptığımız bu çalışmada amacımız yazılım geliştirme maliyet tahminleme çalışmalarında kullanılabilinecek bir ölçüt kümesi oluşturmak, reel yazılım geliştirme projelerinden veriler toplamak ve uygun bir yapar sinir ağı modeli bulabilmektir. Oluşturulan ölçüt kümesi gerçek proje çalışmalarından, proje yöneticileri ile görüşmelerden ve tecrübelerinden alınan veriler üzerinde çalışma yapılarak çıkarılmıştır. Bu çalışmada yapay sinir ağları kullanılarak yazılım maliyet tahminlemesi çalışması ve modellenmesi gerçekleştirilmiştir.

1.3 Yapılan Çalışma

Bu çalışma temelde 3 kısımdan oluşmaktadır. Bunlar:

1. Yeni bir yazılım ölçüt kümesi oluşturma
2. Oluşturulan yazılım ölçüt kümesi için veri toplama
3. Yapay sinir ağını bir araç olarak kullanarak oluşturulan yeni veri kümesiyle yazılım maliyet tahmini modeli oluşturma

Oluşturulan ölçüt kümesinin sağladığı faydalar ve altı çizilebilecek bazı noktalar:

1. Ölçüt kümesi hiyerarşik bir yapıya sahiptir ve ana ölçütler, ölçütler ve alt ölçütler olarak 3 katmanlı olarak kurgulanmıştır.

2. İşlev puanı bir ölçüt olarak alınarak melez bir yapı oluşturulmştur.

3. Plan ve tahminleri de bir ölçüt olarak alarak hem halen çok kullanılan uzman görüşü ile melez bir yapı oluşturulmuş hem de projenin bu plan ve tahminlemenin yarattığı etki de dikkate alınmıştır.

4. Reel sektörün görüşleri doğrultusunda oluşturulmuştur. Oluşturulan ölçüt kümesi için toplam 28 proje yöneticisinin katıldığı vaka çalışmaları yapılmış ve onaylama adımı gerçekleşmiştir.

5. Reel sektörden bu ölçüt kümesine uygun olarak 109 adet yazılım geliştirme projesinden veriler toplanmıştır.

Yapay sinir ağı kullanılarak yapılan yazılım maliyet tahmini modeli ile bu alanda kullanılmayan *Elman* yapay sinir ağı modelini uyguladık. Bunun yanı sıra başarıyı artırmak için %5, %10 ve %15'lik çapraz onaylamalar kullandık. Yazılım maliyet tahminlemede yapay sinir ağı kullanılarak yapılan çalışmalarda böylece yeni olarak *Elman* yapay sinir ağı modelini kullanmanın yanı sıra çapraz onaylama da kullanılmıştır. Çapraz onaylama zaten onaylama için olması gereken bir kontroldür ancak benzer çalışmalarda çapraz onaylama kullanılmamıştır.

1.4 Bölümler

İkinci bölümde yazılım ölçütleri, yazılım ölçütlerinin önemi ve kullanım alanları açıklanmıştır (Fenton, N.E. ve Pfleeger S., 1998; Lorenz M. ve Kidd J., 1994; Chidamber, S.R. ve Kemerer C.F., 1994; Morris K.L., 1989).

Üçüncü bölümde mevcut yazılım maliyet tahminleme yöntemleri ve önemi anlatılmıştır (Fenton, N.E. ve Pfleeger S., 1998; Hihn J. ve H. Habib-Agahi ,1991; Sommerville I., 1992; Boehm B., Bradford C., Horowitz E., Madachy R., Shelby R., Westland C. ,1995).

.Dördüncü bölümde yapay sinir ağları kısaca anlatıldı. Burada daha çok kullanılan yapay sinir ağlarına değinilmiştir (Heaton J.,2005; Wittig, G. ve G. Finnie, 1997; Boetticher G.D, 2003; (Idri A, Khoshgoftaar T.M. ve Abran A.,2002;Briand L. C., K. El Eman, F. Bomarius, 1998;Kirsten R., 2001;Tsuneo Y., Tohru K.,1999)

.Beşinci bölümde yeni oluşturduğumuz ölçüt kümesini, oluşturma adımlarının neler olduğu ve oluşturduğumuz ölçüt kümesi yer almaktadır (Samaraweera L.G., 1996; Weyuker E.J., 1988; Briand L., Morasca S. ve Basili V., 2002; Whitmire S. ,1997; Reed, R. D. ve Marks, R. J. ,1999; Poels G. ve Dedene G. ,2000; Poels G. ve Dedene G., 2000; Daskalantonakis, M.K.,1992).

Altıncı bölümde elimizde var olan ölçüt kümesini yapay sinir ağı kullanarak tahminleme yapma çalışmaları gerçekleştirilmiştir.

Son bölüm olan yedinci bölümde sonuçlar mevcuttur.

2. YAZILIM ÖLÇÜTLERİ

Mühendisliğin temel yapı taşlarından biri ölçmedir. Yazılım mühendisliğinin bir disiplinden çok bir ideoloji gibi kalmasının temel nedeni yazılım mühendisliğinde ölçüme gereken önemin verilmemesidir.

Ölçme gerçek dünyadaki varlıkların özelliklerine onları tanımlayabilmek için açıkça tanımlanmış kurallara göre sayı veya sembol atanması işidir. Bir varlığı ölçmek diye bir süreçten bahsedemeyiz. Bir varlığın bir özelliğini tanımlanmış bir kurala göre ölçme işinden bahsedebiliriz.

Ölçüt bir sistemin veya bir parçanın verilen bir özelliğinin nicel ölçüm derecesidir. Ölçüt hesaplanabilir veya birleşik bir gösterge olabilir. Bir yazılım ihtiyacının büyüklüğünün ölçülebileceği fikrini ilk ortaya çıkaran ve bir sistem kuran Allan Albrecht'tir. 1979'da IBM'de çalışan Albrecht işlev puan analizi adıyla bir metot ortaya koymuştur. California'da bir danışmanlık firması olan TRW'de bulunan birçok proje üzerinde çalışarak, Boehm COCOMO'yu (*Constructive Cost Model*) ortaya koymuştur. COCOMO göreceli olarak açık ve net bir modeldir.

2.1 Yazılım Mühendisliği Ölçütleri

Ölçüt terimi genellikle belirli bir öğe veya bir süreç için yapılan belirli ölçümler kümesini ifade etmek için kullanılır. Yazılım Mühendisliği ölçütleri;

1. Yazılım Mühendisliği ürünleri (Tasarım, kaynak kodu, test vakaları)
2. Yazılım Mühendisliği süreçleri (Analiz, tasarım, kodlama aktiviteleri)
3. Yazılım Mühendisliği kişileri (Test eden kişilerin üretkenliği ve verimi)

ölçmek için kullanılan ölçüm birimleridir. Dolayısıyla yazılım mühendisliği için oluşturulacak ölçüt kümelerinde ürün, süreç ve kişiler temel yapı taşlarıdır şeklinde düşünülebilinir.

2.2 Yazılım Ölçütlerinin Önemi

Ölçütler yapılan çalışmanın temelidir. Gerçek dünyaya ölçütler sayesinde dokunabilmekteyiz. Ölçütler sayesinde verileri toplayabiliyor ve bunlar üzerinde model inşa ediyoruz. Bu açıdan her çalışmada kullanılan ölçütler ve tanımlamalarının başarısı doğrudan çalışma sonucunu etkilemektedir.

Yazılım dokunulamayan ve düşünsel bir ürün olmasından dolayı fiziksel ürünlere oranla daha

fazla, daha net ve daha kapsamlı ölçütlere ihtiyaç duyar. Üstelik yazılım ölçütleri fiziksel ürünlerin ölçütlerine nazaran daha zor tanımlanabilinir ve daha zor anlaşılabilinir olması büyük dezavantajlardır.

Yazılım mühendisliği ölçütleri konusunda son 20 yılda az olmakla beraber bir takım çalışmalar vardır. Ancak çok ağır ilerlemektedir. Veri toplamakta ciddi problemler vardır. Çünkü bu veriler birçok şirket için gizlidir ve paylaşılması şirket için rekabet ortamı için dezavantaj oluşturmaktadır. Şirketler kendileri için bu verileri tutmaya yakın zamanda başladılarsa da paylaşılmamasından dolayı karşılaştırma ve üzerinde çalışma imkânı oldukça düşüktür.

Yazılım mühendisliğinin mühendisliğin temel taşları olan ölçme ve değerlendirme sürecine dahil olabilmesi için ölçütler konusunda çok çalışma yapılması gerekmektedir. Çalışanların, sürecin, ürünün başarı ve başarısızlığı tanımlamada ölçütlere ihtiyaç vardır. Aksi durumda nitelikli bir tanımlama gerçekleştiremeyiz. Yazılım geliştirme sürecimizde gelişimi sağlayabilmek için de ölçütlere ihtiyacımız vardır. Ne durumda olduğumuzu ve nereye gitmeye çalıştığımızı netleştirmemize yaramanın yanı sıra sürekli izlememize dolayısıyla kilitlenme ve gerileme durumlarını yakalamamızı sağlar. Bu tezin de konusu olan yazılım maliyet tahminlemesi yapabilmemiz için kesinlikle ölçütlere ihtiyacımız vardır.

2.3 Yazılım Ölçütleri Kullanımının Faydaları

Eğer yazılım ölçütleri düzgün ve etkili olarak kullanılabilinirse sağlayacağı bir çok yarar vardır. Bu yararlardan bazıları :

1. Bir ürünün, bir sürecin veya bir kişinin başarı veya başarısızlığı derecesinin nicel olarak tanımlanabilmesi sağlanır.
2. Ürünlerimizde, süreçlerimizde ve kişilerdeki gelişim, gelişim eksikliği, bozulma ve gerileme tanımlanabilir ve ölçülebilir.
3. Teknik ve yönetimsel kararlar vermemizde yarar sağlar.
4. Eğilimleri tanımlamamızı sağlar.
5. Mantıklı tahminler yapmakta kullanılabilinir.

2.4 Yaygın Kullanılan Yazılım Ölçütleri

Günümüzde en çok kullanılan ölçütler şu şekildedir:

- Büyüme tertibi (Büyük O)

- Kaynak kodda bulunan satır sayısı (LOC)
- Çevrimsel karmaşıklık (*Cyclomatic Complexity*)
- İşlev puanı
- Satır sayısı başına düşen hata sayısı
- Müşteri ihtiyacının satır sayısı
- Sınıf ve arayüz sayısı
- Bağıntı
- Bağlama

Bu ölçütlere yenilerini de eklemek mümkündür. Bunların temel amacı basit şekilde ürünün büyüklüğünü ve karmaşıklığını elde etmektir. Tüm mühendislik dallarında olduğu gibi yapılacak işin büyüklüğü en önemli kriterdir. Dolayısıyla ürün büyüklüğü diğer bir ifadeyle ürünün hacmi konusunda çalışmalar daha fazladır. Yazılım mühendisliğinde ürün büyüklüğü, geliştirilen yazılımın kod satır sayısı, program dosya sayısı, veritabanı büyüklüğü, algoritma karmaşıklığı ile tanımlanabilir.

Ayrıntılı tasarım yapıldığında geliştirilecek olan yazılımın ne kadar büyük olduğunu tahmin etmek çok zordur. Yazılım ölçütleri bu tahmin uzayını küçültmede pratik araçlardır. Yetenek Olgunluk Modeli (*CMM : Capability Maturity Model*) veya ISO 9000 gibi yazılım geliştirme yönetim metodolojileri daha çok süreç ölçütleri üzerine eğilmişlerdir. Bu ölçütler sayesinde yazılım geliştirme sürecini izleme ve kontrol altına almaya çalışmışlardır. Süreç ölçütlerinden en çok kullanılanlar:

- Gereksinimlere göre yapılan değişiklik sayısı
- Bir haftada kullanılabilinecek programlama zamanı
- Harcanan her saat başına düşen hata sayısı
- Programın ortalama beklenmedik kapanma (*crash*) sayısı
- İlk sürümden önce ihtiyaç duyulan yama sayısı

2.4.1 Nesneye Dayalı Yazılım Mühendisliği Ölçütleri

Son yıllarda nesneye dayalı yazılım mühendisliği ölçütleri konusunda daha fazla çalışma mevcuttur. Mark Lorenz ve Jeff Kidd yaptıkları bir çalışmada ilginç gözlemlere ulaşmışlardır. Bunlardan bazıları (Lorenz M. ve Kidd J., 1994; Chidamber, S.R. ve Kemerer C.F., 1994) :

- Nesne sayısının destekleyici nesnelere oranının 1'e 2.5 olduğunu gösterdiler. Kullanıcı arayüzü daha gelişmiş uygulamaların daha fazla destekleyici nesnelere ihtiyaç duyduğunu gösterdiler.

- Bir metot için yazılan ortalama satır sayısı arttıkça nesne yönelimliliğin azaldığını belirttiler.
- Kullanılan uygulama dili değiştikçe bir nesne yazmak için gereken ortalama adam saat sayısının büyük oranda değiştiğini ve ihtiyaç duyulan büyüklüğün de değiştiğini belirttiler.

Nesne yönelimli yazılım geliştirme yaşam döngüsü tipik prosedürel yaklaşımdan çok daha basit bir döngü değildir. Ancak ölçüt somutluğu konusunda bir takım avantajları vardır. Bir çalışmada (Morris K.L., 1989) verimlilik ölçümü açısından bazı önemli ölçüt gözlemleri yapılmıştır. Bunlardan önemli olanları:

- Bir sınıfta yazılan ortalama method sayısı
- Kalıtım ağacında en büyük derinlik
- Sınıflar arası bağlılık
- Bağıntı derecesi
- Sınıf kütüphanesi etkinliği
- Kalıtsal metotların yeniden kullanım derecesi
- Ortalama metot karmaşıklığı
- Uygulamada ayrışan parçaların büyüklükleri

şeklindedir. Bu ölçütlerle bir takım yargılara ulaşmak mümkündür. Örneğin ortalama metot sayısı ile ilgili olarak:

- Bu sayı büyüdükçe sınıfların büyüklüğü ve karmaşıklığı artmaktadır. Bununla beraber testler daha karmaşıklaşmaktadır.
- Sayı çok büyük olmasıyla geliştirilebilirlilik oldukça zor olmaktadır.

Nesne yönelimli yazılım ölçütlerinin sağladığı yararların başında nesne yöneliminin doğasından kaynaklanan bir kolaylık sağlaması ve bazı ölçütlerin sınıf testleri maliyeti konusunda bir takım iyi fikirler vermesidir. Ancak sınıf karmaşıklığı konusunda sapmalar gayet olasıdır. Yazılım maliyeti konusunda iyi bir bilgi verememektedirler. Çok önemli bir nokta ise bu ölçütlerin ancak tasarım aşamasında ortaya çıkabilmeleridir. Planlama aşamasında birçoğuna ulaşmak mümkün değildir.

2.4.2 Yazılım Kalite Ölçütleri

Yazılımın kalitesinin ölçülmesi genellikle zordur. Bu yüzden genellikle kalite ölçütleri kullanılmaktadır. Genellikle kullanılan kalite ölçütleri :

- Kullanım kolaylığı : Yazılımın öğrenilmesinin ve kullanılmasının kolaylık derecesidir.
- Doğruluk : Yazılımın amaçlanan fonksiyonları doğru ve eksiksiz olarak yerine getirmesidir.
- Genişleyebilirlik : Yazılımın genişleyebilme derecesidir.
- Hassaslık : Sonuçların sayısal doğruluk derecesidir.
- Başarım : Yazılımın amaçlanan fonksiyonları istenen hızda yerine getirmesidir.
- Bütünlük : Amaçlanan tüm fonksiyonların gerçekleştirilme derecesidir.
- Denetlenebilirlik : Yazılımın standartlara uyum derecesinin denetlenebilirliğidir.
- Belgelendirme : Yazılımın tamamının belgelendirme derecesidir.
- Tutarlılık : Yazılımın tüm geliştirme süreci boyunca aynı tasarım ve belgelendirme tekniklerinin kullanılmasıdır.
- Verimlilik : Çalışmada başarım derecesidir.

Kalite ölçütleri sayesinde maliyetlerin düşmesi, verimin artması ve izlenebilirliğin artması artmaktadır. Ölçüt kullanımı yazılım geliştirme kurumları için yazılım geliştirme kültürü ve disiplininde yer alan bir durumdur.

2.5 Yazılım Ölçüt Yaklaşımı ile İlgili Eleştiri

Yazılım ölçüt yaklaşımı ile ilgili bir takım eleştirilerde mevcuttur. En önemlileri şu şekildedir:

1. Etik olmaması: Bir kişinin performansının değerlendirilmesinin ve yargılanmasının bir kaç sayısal değişkene indirgemenin etik olmadığı yönünde görüşler vardır. Bir yönetici en yetenekli çalışanına projenin en zor kısmını verebilir. Bu durumda bu kısmı geliştirmek en uzun zamanı alması ve en fazla hatanın buradan üremesi muhtemel olduğu yönünde görüşler mevcuttur. Yazılım geliştirmede kısımlar homojen yayılmadığından ve baştan öngörmek zor olduğundan kişilerin performansını ve başarısını ölçütlerle ölçmenin yanıltıcı sonuçlar verebileceği yönünde düşünceler mevcuttur.
2. Basite indirgeme: Sayılarla yönetim şekli yöneticiler için tercih edilebilecek bir

yöntemdir. Basit olmasının yanı sıra gelecek planları yapmak ve karar vermeyi kolaylaştırmaktadır. Ancak çalışanların tecrübelerinin kalitesinin önemsenmemesi yönünde eleştiriler vardır.

3. Çarpıklık: Süreç ölçütleri çalışanların davranışları üzerinde çarpıklık yaratabilir. Temel amaç yönetimin izlediği ölçütlere odaklanabilmektedir. Örneğin satır sayısı ile performans değerlendirmesi yapılıyorsa, çalışanlar gereksiz kodlamalar yapabileceği, kısa çözümler bulma yerine uzunca çözüme gitmeye çalışacakları yönünde ve hatta hiç kullanılmayan kodlar olabileceği yönünde eleştiriler vardır.

4. Hatalılık: Bilinen hiç bir ölçütün anlamlı ve doğru olmadığı yönünde görüşler vardır. Satır sayısı sadece yazılana bakmakta ve problemin zorluğunu dile getirmemektedir. İşlev puan problemin karmaşıklığını daha iyi ölçmek için oluşturulmuş olmasına rağmen kişisel hükümlere ihtiyaç duyduğu ve değişik kişilerin değişik sonuçlara ulaşabildiği ve bu işlev puanı kullanımı zor bir hale getirdiği görüşleri mevcuttur.

Bu eleştiriler yetersiz sayıda ve uygun olmayan ölçütler için mantıklı görünmektedir. Ancak yazılım geliştirme sürecini her yönden ele almaya çalıştığımızda, ihmal edilen kısımların azalması ile endişeler azalacaktır.

Endüstri tecrübeleri ölçütlerin tasarımının kişilerin davranışını kaçınılmaz olarak etkilediğini söylemektedir. Basit bir örnek olarak yazılım geliştirme sürecinde işlev puan bazında ücretlendirmeyi ele alabiliriz. İşlev puan bazında ücreti küçük çıkarmanın en kolay yolu işlev puanı küçük çıkarmaktır. İşlev puanı ölçmek için net bir standart olmadığı için ve değişik uygulamaları olduğu için yanlış kullanıma müsaittir.

Test bazlı yazılım geliştirmede yazılım geliştirenlerin testi geçme ölçütlerini kullanılabileceği yönünde düşünceler vardır. Bu durumda temel amaç nihai üründen ziyade testi geçmek olacaktır. Yanlış test yazılma riski çok ciddi bir risktir.

Ölçütlerin kullanımı ile performans değerlendirmesi yapmak veya bir ürünü değerlendirmenin oluşturacağı durumlardan kaçınmanın en güzel yollarından birisi dengelenmiş ölçütlerin kullanılmasıdır. Birbirini dengeleyici ölçütler sayesinde görev alan kişilerin davranışlarını ve süreci daha fazla kontrol altına almış olabiliriz. En çok önerilen ölçütler şu şekildedir:

- Zaman Planı
- Risk
- Maliyet

- Kalite

Bunlardan birine diğerlerinden daha fazla önem verilmesi durumunda takımın motivasyonu üzerinde bir dengesizlik oluşturulabilinir.

Dengeli skor kart (*Balanced-Scorecard*) birden fazla performans perspektifini bir yazılım ölçütü kümesi kullanarak yönetebilecek bir araçtır. Bu araç kullanılarak verim artımı sağlanabilir.

3. YAZILIM MALİYET TAHMİNLEME

Yazılım maliyet tahmini, üretilecek olan bir ürün veya hizmetin maliyetinin ne olacağının nicel olarak tahmin edilmesi işidir. Yazılım maliyet tahmini, bir proje için plan yapılması sırasında başlayan bir ihtiyaçtır. Tanımlanan projenin ne kadar sürede tamamlanacağı, maliyetinin ne olacağının tahmini yapılmaksızın tutarlı bir plan yapmak güçleşir. Eldeki bilgi başlangıçta doğru tahmin yapmak için yetersizdir. Olanak varsa tahminleri olduğunca geciktirmek, yapılacak tahminin doğruluğu açısından faydalı olabilmektedir. Ancak beklemek her zaman mümkün olmayabilir. Günümüzde eldeki bilgiler ve tecrübeler ile ilk tahminler yapılsa da proje devamı boyunca tahmin hesaplarını devamlı yenilenmekte ve planlarda revizyonlar yapılmaktadır. En doğru tahmin proje tamamlandığı zaman yapılan tahmindir. Doğru kestirimin zorluğu yanında ona duyulan ihtiyaç da ortadadır.

Yazılım maliyet tahminleme konusunda özellikle 1970'lerden bu yana çeşitli çalışmalar yapılmaya çalışılmış ancak henüz istenen başarıma ulaşılamamıştır. Bu konuda oldukça yavaş ilerleniyor olması sebeplerden bir tanesidir. Bu bölümde mevcut önemli yazılım maliyet tahminleme yöntemleri açıklanmaya çalışılmıştır.

3.1 Yazılım Maliyet Tahminlemenin Gelişimi

İlk yazılım ölçüm yöntemlerinden biri satır sayısını sayma yöntemidir. Satır sayısı sayma yöntemi uzun süredir tartışılan bir yöntemdir. Bir programcının verimliliği, satır sayısı sayarak ölçülmeye çalışılmıştır. 70'lerin ortasında geliştirilen çevrimsellik karmaşıklığı (*Cyclomatic Complexity*) temel olarak yazılım karmaşıklığını ölçmeye yarayan bir graf teoreminin kullanımıdır. Çevrimsellik sayısı, minimum yol sayısının bulunması işidir. 1979'da yazılım ölçütleri konusunda yazılan ilk kitap olan "Software Metrics" adında ki kitap Tom Gilb tarafından yayınladı. 1979'da Albrecht tarafından işlev puanın (*FP : Function Point*) tanımlanması önemli bir adımdır.

Eskiden yazılım geliştirme bugüne kıyasla oldukça basit iken satır sayısına veya kelime sayısına bakarak tahminler yapılabilirdi. Burada bir fonksiyon ortaya konabilir ve yeni yazılım geliştirme projelerinde bu fonksiyon kullanılabilinirdi. Daha sonra bunun yetersiz kalmasıyla beraber kullanıcı etkileşimi, ekran sayısı, kullanılan dosya sayısı gibi parametreler de eklendi. Kod satır sayısı (*LOC : Line of Code*) ve işlev puanı (İP) maliyet tahmininde en çok kullanılan yöntemlerdi. Parametrik modeller ise teorik bazda ve temelde yapılan eşitliklerden yararlanmaktadırlar.

Yazılım Maliyet Tahminleme çalışmalarında öncelikle yapılması gereken yazılım geliştirme çabasının tahminlenmesini sağlamaktır. Bunu sağlamak için ürünün büyüklüğü, sistemin karmaşıklığı ve uygulama alanını göz önünde bulundurmak gerekir. Ürünün sonuç maliyetini oluşturan tek etken olarak ürünün büyüklüğünün düşünülmesi genel bir yanlıştır. Ancak hala yaygın olarak kullanılmakta ve tek veri ile sonuç maliyet tahmin edilmeye çalışılmakta ve dolayısıyla bütçe aşmaları yaşanmaktadır (Fenton, N.E. ve Pfleeger S., 1998).

Bu düşünme tarzında

$$Ç = f(s) \times g(A) \tag{3.1}$$

Ç : Tahmin edilen Çaba

f(s) : Ürünün büyüklüğü

g(A) : Ayarlama fonksiyonu

olduğu düşünülmektedir. Ancak buradaki başarım oldukça düşüktür. Yazılım maliyetinden bir takım bilgilere ulaşma yöntemi de kullanılmaktadır. Yazılım maliyet tahminlemede uzun süre kullanılan bu düşünme tarzında çoğunlukla (3.2)'de ki eşitlik kullanılmıştır.

$$Çaba = A\ (KLOC)^{b} \tag{3.2}$$

Kaynak kodun satır sayısının bir kuvvetin bir düzeltme çarpanı ile beraber kullanarak gereken çabayı verdiği düşünülmüştür. Gerçekleşen bazı yazılım geliştirme projelerinden elde edilen verilerden eşitlik 3.2'de bulunan A ve b sabitleri elde edilmeye çalışılmıştır. Diğer bir ifadeyle elde edilen belli sayıda proje verileri kullanılarak, matematiksel olarak A ve b sabitleri bulunulmuştur. Bu bölümde tanıtılan tahmin yöntemleri, satır sayısı temelinde olsa da, işlev puanına dayandırılması da mümkün olabilmektedir. Hesapların temelindeki önemli bir nokta, gereken çabanın ürünün büyüklüğünün bir kuvveti şeklinde hızla artmasıdır. Bunun sonucu olarak rasgele yaklaşımla yazılmış bir programın yazılması başta oldukça kolay olmasına rağmen ilerleme oldukça süreç zorlaşmaktadır. Birkaç bin satıra yakın büyüklükteki bir programı bir hafta süresinde yazmak mümkündür. Daha sonra program büyüdükçe hız yavaşlar. Bu program birkaç on bin satıra ulaştığında bir satır ilave etmenin bedeli bir kaç günlük belki de bir kaç aylık çabadır. Dolayısıyla eklemenin oluşturacağı yan etkileri takip etmek zorlaşmıştır.

Temel olarak iki sebepten dolayı yazılım maliyet tahmini yapmak oldukça zordur. Birinci neden yazılımın soyut, elle dokunulamayan, fiziksel olarak alışılmış ürün tanımı dışında olmasıdır. İkinci neden ise yazılım geliştirme işi fiziksel bir işten ziyada entelektüel bir iş olmasıdır.

Londeix 1987'de makro tahminleme tekniklerinin ve mikro tahminleme tekniklerinin olduğunu belirtmiştir. Boehm 7 farklı yazılım maliyet tahminin olduğunu belirtmiştir. Bunlardan 2 tanesi Parkinson tekniği ve Kazanmak için maliyet (*price-to-win*) tekniğidir. Bu teknikler realisttik yazılım maliyet tahminlemesinden çok yönetimsel amaçlar için kullanılmaktadır. Yukarıdan aşağıya ve aşağıdan yukarıya teknikleri uzman yargılarda, parametrik ve benzeşim tekniklerinde kullanılmaktadır. Uzman yargı tekniği bir uzman seçme ve bu uzmanın tecrübe, bilgi birikimi ile tahmin etmesi üzerine kuruludur. Parametrik ve benzeşim teknikleri önceki verileri bir veritabanında kaydederek maliyeti tahmin etmede kullanılır. Bu çalışmada temelde parametrik benzeşim bazlı maliyet tahminleme tekniği kullanacaktır. Parametrik modellerde genelde kullanılan formül proje karakteristiğine göre değiştirilmektedir. Modelin doğruluğu ve tutarlılığı analistin bu parametreleri ortaya çıkarma yeterliliğiyle beraber artar. Ölçüt veritabanının kullanım verimliliği ve doğrulu ile beraber başarım artacaktır. Burada ölçütlerin bulunmasında deneysel ve gözlemsel parametrik model ve teorik parametrik model kullanılabilmektedir. Deneysel parametrik modelde geçmişteki projelerden elde edilen verilerden elde edilen fonksiyonlar kullanılır. Parametrelerin ve sabitlerin bulunmasında istatistiksel analizler yapılır. Ünlü Rayleigh eğrisi bu şekilde ortaya çıkmıştır. Daha sonra Putnam 1978'de bu eğriyi geliştirmiş ve proje çaba ölçütlerini eklemiştir. Burada hiperbolik sekant fonksiyonu kullanmıştır. Parr 1980'de (Parr, F.N., 1980.) bir alternatif önermiştir. Bu modelde problemlerin yazılım geliştirme sürecinde çözüldüğü öngörülmüştür. Dolayısıyla kod satır sayısı doğrudan karmaşıklığı vermemesi, gözlemsel kanıtlar bunu desteklemektedir.

Benzeşim bazlı tahminleme modellerinde aynı organizasyonda yapılan önceki projelere bakılır. Tahminlemede yapılmış projelerde benzer karmaşıklığa ve büyüklüğe bakılır. Gözlemler göstermektedir ki benzeşim bazlı modeller parametrik modellerden daha başarılı olmaktadır. Bu modellerde bir veya daha fazla uzman bulunmaktadır.

3.2 Yazılım Maliyet Tahminleme Yöntemleri

Yazdırılacak bir programın değerini saptamak için bir şekilde onu ölçmemiz gerekir. Kolaylığı ve doğrudan ölçülebilirliği açısından en fazla kullanılan yazılım ölçme yöntemlerinin temeli satır sayısıdır. Bir programın büyüklüğü denince ilk akla gelen kaç satırlık kaynak kodu ile üretildiğidir. Bazen bu, programın karmaşıklığı için de bir ölçüm olarak ifade edilir. Ancak, benzer işlevi değişik programcılar farklı büyüklükte kodlarla temin edebilirler. Programın satır sayısı büyüklüğü, onun karmaşıklığı hakkında tam doğru bir fikir

vermeyebilir. Sonuçta önemli olan verilen para karşısında ‘ne kadar’ işlevsellik alındığıdır. Bu düşünce ile geliştirilen ‘işlev puanı’, satır sayısına karşı bir seçenek olarak karşımıza çıkmış bir yazılım ölçüsü birimidir. Bu ölçü, satır sayısı gibi açıkça tanımlanmış bir doğrudan ölçüm birimi değilse de yazılımı değerlendirmek için yaygınca bir şekilde daha anlamlı bulunmaktadır. Bu iki en yaygın temel ölçüden başka yöntemlerle de program karmaşıklığı ve büyüklüğü ölçülmektedir.

Yazılım Ölçümlerinde doğrudan ve dolaylı ölçülebilen büyüklükler vardır. Doğrudan ölçülebilen büyüklüklerden bazıları:

- Zaman
- Satır sayısı
- Maliyet
- Çaba
- Bildirilen hata sayısı
- Çalışan kişi sayısı

Dolaylı ölçülebilen değerlerden bazıları:

- Karmaşıklık
- Kalite
- İşlevsellik
- Güvenilirlik
- Bakım kolaylığı
- Kullanırlılık
- Performans
- Güncelleştirilebilinirlik

Yazılım geliştirme şirketleri proje adı, satır sayısı, çaba, maliyet, doküman sayfa sayısı, hata sayısı, bozukluk sayısı, personel sayısı ve başka değerleri de içine alan bilgileri tutmaya çalışırlar (Samaraweera L.G., 1996). Şirketler, bu gibi bilgileri toplayarak bir tarihçe birikimi sağlamak isterler. Buna dayanarak ileride alacakları projeler için kestirimde bulunurlar. Bu bilgileri kullanarak personel maliyeti, program satırı maliyeti, hatalılık, dokümanlılık gibi sonuçları basit matematiksel hesaplamalarla bulabilirler. Bunların yanında “hata / adam x ay”, “satır sayısı/ adam x ay”, “doküman sayfası/kod satır sayısı”, “doküman sayfası / adam x ay” gibi değerlere de ulaşabilirler. Doğrudan kod satır sayısı yerine genellikle daha geçerli olan 1000 satıra karşı düşen KLOC birimi, yazılım ölçümlerinde bir standarttır. Bir başka önemli nokta da verilen bilgilerin sadece programlama değil, projenin analiz, tasarım, onarım gibi

bütünü için hesaplandığıdır. Temel olarak yazılım maliyet tahminleme yöntemlerini iki ana grup altında inceleyebiliriz: algoritmik modeller ve algoritmik olmayan modeller.

Heemstra 364 organizasyonda hangi tahminleme metotlarının kullanıldığını incelemiş (Heemstra F. J.,1992) ve 51 adet tahminleme yöntemi olduğunu ortaya koymuştur. Ancak modeli olan kişinin bir modeli olmayandan daha iyi bir tahmin yapmadığını göstermiştir. Uzman Görüşü bu modeller arasında en doğruya yakın olduğunu belirtmiştir.

Algoritmik modeller oldukça az kullanılmaktadır. Yapılan bir araştırmada (Hihn J. ve H. Habib-Agahi ,1991) yazılım maliyetini tahmin eden kişilerin sadece %7'sinin algoritmik bir model kullandıklarını ortaya koymuştur.

3.2.1 Satır Sayısı Yöntemi ile Kestirim

Bu yöntemde, proje tahmin edilen alt birimlerine ayrıştırılır. Parçala - yönet stratejisi sonucunda ortaya çıkan, üzerinde tahmin yapılması daha kolay olan daha küçük her birim için satır sayıları önerilir. Bu kestirimler yapılırken de en küçük, en olası, ve en büyük ihtimaller belirlenip bunlarla bir ortalama işlemi yapılabilinir. Bir birim için tahmin edilecek en küçük satır sayısına "minimum", en olası satır sayısı tahminine "olası**"**, ve en büyük tahmin değerine de "maksimum" denecek olursa, o birim için:

Satır sayısı kestirimi eşitlik (3.3)'deki gibidir.

$$\frac{(\text{minumum} + (4 \text{ x olası}) + \text{maksimum})}{6} \tag{3.3}$$

Birimler için ayrı ayrı tahminler yapılır ve daha önceki deneyimlerden benzeri birimlerin geliştirilmesindeki şirketin verimliliği gibi değerler kullanılarak satır sayısı tahminlerinden çaba, zaman ve maliyet kestirimlerine varılır. Projenin bütünü için, birimlerin çaba, zaman ve maliyet kestirimleri toplanarak değerler elde edilir. Birimlerin satır sayıları toplanarak proje bütünü hakkında çaba ve zaman gibi kestirim hesaplarını bir kerede yapmaktan kaçınılmalıdır. Satır sayısı büyüklüğü ile diğer sonuç değerlerinin doğrusal olmayan bir ilişki ile bağlantılı oldukları hatırlanmalıdır.

3.2.2 İşlev Puanı Yöntemi ile Kestirim

İşlev puanı yöntemini kullanarak kestirim yapılabilinir. Eğer proje ile ilgili girdi çıktı gibi özellikler tahmin edilebiliyorsa bunlar kullanılarak geliştirilecek sisteme ait bir değer elde edilir. Satır sayısı tekniğinin tersine bu yöntemde bir yazılım birimi için doğrudan büyüklük

tahmini yapma zorluğu yoktur. Aksine, ihtiyaçların belirlemesi çalışmalarında ortaya çıkabilecek değerler kullanılarak sonuca varılabilir.

İşlevsellik doğrudan ölçülemeyeceğinden, bir yazılım projesinde işlevselliğe etkisi olan birçok etken bir arada incelenerek ürüne olan etkileri ağırlıklandırılabilinir. Sonuç olarak bir sayı ortaya çıkar ve bu rakam değişik projeleri göreceli olarak değerlendirmekte yararlı olabilir. İşlev puanını satır sayısının yerine konarak diğer dolaylı ölçümlere ulaşılabilinir. İki ayrı yaklaşımı birleştirici 'dönüştürme' tabloları bulunmaktadır. Satır sayılı veya işlev puanlı ölçmelerde birinden diğerine geçmek, kabaca değerlerle mümkündür. Çizelge 3.1 bu çevrimi değişik programlama dilleri açısından vermektedir. Çizelge 3.1 sayesinde elimizde bulunan bir işlev puan için ortalama kaç satır kod gerektği bulunulabilinmektedir. Çizelge 3.1'de bulunan dönüştürme nesne tabanlı değil, prosedürel yaklaşıma göre oluşturulmuştur. Nesne tabanlı yaklaşımda gerekli satır sayısı ciddi oranda azalmaktadır. Ancak satır sayısının azalması gereken iş gücünün doğrudan azalması veya artması anlamı taşımamaktadır.

Çizelge 3.1 Satır Sayısı / İşlev Puan dönüşümü (Pressman R.S., 2005)

Yazılım Geliştirme Dili	Ortalama	Medyan	En Az	En Fazla
Access	35	38	15	47
Ada	154		104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
Cobol	77	77	14	400
Java	63	53	77	
JavaScript	58	63	42	75
JSP	59			
Oracle	30	35	4	217
Perl	60			
PL/I	78	67	22	263
PowerBuilder	32	31	11	105
SQL	40	37	7	110
VisualBasic	47	42	16	158

İşlev puanı hesaplamak için karmaşıklığa etki eden faktörlerin ağırlıklandırılmaları ve sonuçta karmaşıklığa olan etkileri denenerek bu ağırlıkların ayarlanması şeklinde yöntemler uygulanmıştır. Bulunan sonuçları, verilen formüllere proje ile ilgili doğrudan ölçebileceğimiz büyüklükleri girerek kullanabiliriz. İşlev puanı hesaplama için iki işlem yapmak gerekir. Önce 'Toplam Sayı' denilen nesnel ölçümlere dayanan değeri bulmak gerekmektedir.

Çizelge 3.2 İşlev Puan hesaplama çizelgesi

Ölçme parametresi	Adet	Basit	Orta	Karmaşık	Sonuç
Kullanıcı girdi sayısı	…	3	4	6	…
Kullanıcı çıktı sayısı	…	4	5	7	…
Sorgu sayısı	…	3	4	6	…
Kütük (dosya) sayısı	…	7	10	15	…
Dışsal arayüz sayısı	…	5	7	10	…
Toplam Sayı	…	…	…	…	…

Dışsal arayüz sayısı, kütükler ve bunun gibi her türlü makine tarafından anlaşılan bilgi aktarım noktaları sayısıdır. Kullanıcı girdileri, kullanım sırasındaki veri girişleridir. Toplam sayı bu şekilde bulunduktan sonra işlev puan 3.4'deki eşitlik ile hesaplanır.

$$İP = (\text{Toplam Sayı}) \times (0.65 + (0.01 \times K_i)) \quad (3.4)$$

Burada 14 adet K_i 'Karmaşıklık Ayarlama Faktörü' mevcuttur ve aşağıdaki belirtilen soruların cevaplarından meydana gelir. Bu sorulara verilecek cevaplar 0 ile arasında değer alır. Böylece ortaya çıkacak 14 değer toplanır ve 0.01 ile çarpılarak formüle konur.

Karmaşıklık Ayarlama Faktörleri (K_i) (Sommerville I., 1992) :

1. Güvenilir yedekleme ve kurtarma işlemi gerekli mi?
2. Veri iletişimi gerekiyor mu?
3. Dağıtık işlem ve süreçler var mı?
4. Çabukluk önemli mi?
5. Sistem mevcut ve fazla yüklü bir ortamda mı çalışacak?
6. Çevrimiçi (*on-line*) veri girişi gerekecek mi?
7. Çevrimiçi (on-line) giriş, fazla ekranlı veya fazla işlemli mi?
8. Ana kütükler çevrimiçi (*on-line*) olarak mı güncellenecek?
9. Girdi, çıktı, sorgulama ve kütükler karmaşık mı?
10. İç süreç (*internal process*) karmaşık mı?
11. Program yeniden kullanılabilir olarak mı tasarlanıyor?
12. Dönüştürme (*conversion*) ve kurma (*installation*), tasarımın içinde yer alıyor mu?
13. Değişik kuruluşlarda çoklu kurmalar tasarlanıyor mu?
14. Kullanıcının kolaylığı ve uyarlamasına göre tasarlanıyor mu?

Hangi temel seçilirse seçilsin, ortaya çıkan ölçümler, diğer dolaylı ölçümlere ulaşmak için kullanılabilir. Kalite doğrudan ölçülemeyen bir kavramdır. Kalite ile ilgili birçok parametre

dolaylı olarak belirtilebilir:

- Doğruluk: Yazılımın istenilen işlevi ne derecede yerine getirdiğinin bir ölçüsüdür. Hata sayısı / KLOC olarak ölçülebilir.
- Bakım/onarım kolaylığı: İstenen bir değişikliğin ne kadar çabuk gerçekleştirildiği ile değerlendirilir. Ortaya çıkan bir hata ile bu zaman süreci başlar. Yazılım değişmiş olarak tekrar çalışmaya başlayıncaya kadar ölçülür. Bu süre, bakım kolaylığı daha yüksek olan yazılımlar için daha kısadır.
- Bütünlük: Saldırıya karşı koyabilme yeteneğidir (3.5):

  Bütünlük = S (1 - tehdit x (1 - güvenlik)) (3.5)

  Değişik tehditler, ilgili saldırının ortaya çıkma ihtimali olarak hesaplanır ve onlara karşı sistemin dayanma olasılığı da 'güvenlik' değerini oluşturur.
- Kullanım kolaylığı: Bu özelliğin ölçülmesi daha ileri düzeyde 'dolaylılık' gösterir. Sistemin kullanılması için gereken fiziksel çaba veya bilgi seviyesi, sistemi makul bir verimlilikle kullanabilmek için gereken alışma süresi ve kullanıcılar arasında yapılan anketler gibi yöntemlerle değerlendirilebilir.
- Bozukluk giderme verimliliği: İdeal olarak projenin tesliminden önce bütün hatalarının bulunarak giderilmesi istenir. Bozukluk Giderme Verimliliği tanım olarak (3.6)'da gösterilmekteidir.

  Bozukluk Giderme Verimliliği = HÖ / (HÖ + HS) (3.6)

  burada HÖ, proje tesliminden önce bulunan hatalar, HS ise Proje tesliminden sonra bulunan hatalardır.

3.2.3 Putnam Metodu

Putnam metodunda, sistem geliştirme sürecinin zamana bağlı olarak gösteren çaba ve maliyet eğrileri bulunmaktadır. Gerçekçi bir projede hesaplanan adam x ay değeri, süreç boyunca sabit kalmaz. Dolayısıyla bazı ayların personel gereksinimi, diğerlerine göre farklı olacaktır. Bu çaba-zaman eğrisi gözlenerek personel istihdamı ayarlanabilir. Aynı kurum içerisinde farklı projeler arasında eleman kaydırmaları da Putnam eğrilerinin (3.7) bir sonucu olarak mümkün olur (Putman W.L., 1978):

$$\text{Çaba} = ((\text{Satır Sayısı}) \times (\text{ÖY}^{0.333} / \text{VF}))^3 \times (1 / \text{PS}^4) \quad (3.7)$$

Burada 'PS' proje süresidir. (3.7) eşitliğinde ‘ÖY’, özel yetenekler katsayısıdır. 5.000 ile 15.000 satırdan oluşan küçük projeler için özel yetenek değeri 0.16 iken, 70.000 satırlık veya daha büyük projeler için 0.39 dur. ‘VF’ ise verimliliğe bağlı olarak değişir: gerçek zamanlı sistemler için 2000 olan değer, sistem programları ve iletişim programları için 10.000 ve iş bilgi sistemleri için 28.000 dir. Verimliliğe etki eden bazı faktörler:

- Geliştirilen uygulamanın karmaşıklığı
- Yazılım geliştirme ortamı
- Takımın yetenekleri ve deneyimi
- Yazılım mühendisliği uygulanmasının niceliği
- Kullanılan programlama dilinin soyutlama düzeyi
- Süreç olgunluğu ve yönetim uygulamaları

3.2.4 Halstead Tekniği

Halstead tekniği, bir yazılımın iç yapısını inceler (Fenton, N.E. ve Pfleeger S., 1998). Yazılım içerisinde kullanılan değişik işlemlerin sayısını ve bu işlemlere parametre olarak kullanılan değerlerin sayısını kullanarak bir karmaşıklık hesaplaması yapar. Dolayısıyla satır sayısından bağımsız, doğrudan programın yaptığı işleve yönelik bir değerlendirme girişimi olarak ortaya çıkmıştır. Daha küçük ve karmaşık projeler için bir değerlendirme aracı olarak kullanılmıştır. Veriler ve program yapılarındaki işlem dışı etkenlerin hesaba katılmadığı bir gerçektir.

Bu ölçümün ortaya çıkış nedeni, test amacı ile bir programın akışı diyagramını çizecek olursak, her patikanın ele alınması için ne kadar iş yapılması gerekir sorusudur. Bu akış diyagramı incelendiğinde, patika sayıları, karar noktaları denilen düğümlerin sayıları toplamı artı '1' dir. Karar noktaları ise 'IF' komutları (koşutlar) ve çevrimlerin başlangıç noktalarıdır.

Halstead tekniği M.H. Halstead tarafından geliştirilmiştir. Amacı programlama çabasını bulmaktır.

Ölçülebilir ve sayılabilir özellikler:

n_1 = Bu kodlamada kullanılan ayrık veya tekil operatör sayısı

n_2 = Bu kodlamada kullanılan ayrık veya tekil operand sayısı

N_1 = Kullanılan toplam operatör sayısı

N_2 = Kullanılan toplam operand sayısı

Bu ölçütler Halstead tarafından tanımlamıştır ki:

1.Sözlük n şu şekilde tanımlanabilir n= n1 + n2

2.N'in kodlamadaki uzunluğu şu şekilde tanımlanabilir: N = N1 + N2

Operatörler "+", "*" olabilir veya bir ifade olabilir. Sabit bir ifadede bulunan operand sayısı da sabit veya değişkenden oluşur. Burada uzunluk N ile sözlük n arasındaki ilişkinin farkında olmak önemlidir. Deneysel olarak bulunan bir matematiksel ifade (3.8) gösterilmiştir.

$$N' = n_1 \log_{2(}n_{1)} + n_2 \log_{2\,(}n_{2)} \qquad (3.8)$$

N' burada program uzunluğuna oldukça yakın bir sonuç vermektedir. Aynı algoritma daha alt seviye programlama dilleri için de kullanılabilinir. Pascal'da program yazmak assembler'dan daha kolaydır.

Bilgi içeriği bu programın hacmini verir. Bunu ölçmek için bir takım ölçüt ve formüle ihtiyacımız vardır.

- Program'ın Hacmi (V): Bir algoritmanın kodlamasının uzunluğu (3.9)'da gösterilmiştir.

$$V = N \log_2 n = N \log_{2(}\, n_1 + n_2) \qquad (3.9)$$

- Program Seviyesi (L): Bu program hacmi ile potansiyel hacim (V^*) arasındaki ilişkidir (3.10). Sadece çok net ve açık olan algoritmaların program seviyesi aynı veya çok yakın olabilir.

$$L = V^* / V \qquad (3.10)$$

- Program Seviyesi Eşitliği: Bu aslında program seviyesi yakınsama eşitliğidir (3.11). Potansiyel hacim bilinmediği bilinmediği durumlarda kullanılır.

$$L' = n^*_1 n_2 / n_1 N_2 \qquad (3.11)$$

- Bilgi içeriği (3.12)'de gösterilmiştir.

$$I = L' \times V = (2n_2 / n_1 N_2) \times (N_1 + N_2) \log_2(n_1 + n_2) \qquad (3.12)$$

Bu eşitlikte sağdaki tüm değerler doğrudan olarak ölçülebilir. Bu potansiyel hacimle doğrulattırmaktadır. Aynı zamanda potansiyel hacim kullanılan dilden bağımsızdır ve bilgi içeriği de dilden bağımsızdır.

- Programlama Çabası: Programlama çabası var olan bir algoritmayı bir dilde gerçek kodlamaya dönüştürme aktivitesidir. Programlama çabasını bulmak için bazı ölçüt ve formülleri bulmak gerekir.

- Potansiyel Hacim (V'): Belirli algoritmayı çözen olası en küçük hacimdir (3.13).

$$V' = (n*1 + n*2) \log_2 (n*1 + n*2) \quad (3.13)$$

- Çaba Eşitliği (E): Temel mental ayraçların sayısıdır (3.14).

$$E = V / L = V^2 / V' \quad (3.14)$$

- Zaman Eşitliği (T'): Bu kavram insan beyninin işleme oranıyla ilintilidir. İnsan beynini kullanım oranıyla ilgili olduğunu psikolog John Stroud ortaya koymuştur. Stoud birçok ayrık temel ayrımımı zamansal olarak tanımlamıştır. S ile ifade edilen Stroud sayısı saniyedeki işlem sayısıdır. S sayısı 5 ile 20 arasında değişmektedir. Sonuç olarak zaman eşitliği (3.15)'de gösterimiştir.

$$T' = (n_1 N_2 (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 n) / 2 n_2 S \quad (3.15)$$

Halstead yönteminin faydaları:

- Hesaplanması basittir.
- Hata oranını tahmin edebilmektedir.
- Her programlama dili için kullanılabilinmektedir.
- Bakım maliyetini ve çabasını tahmin edebilmektedir.
- Zaman planlaması yapmakta ve raporlamada faydalıdır.
- Programın tüm kalitesini ölçebilmektedir.
- Derinlemesine program yapısını analiz etmeye gerek yoktur.
- Program çaba ve olası hata sayısını tahmininde birçok endüstriyel çalışma Haltead'ın kullanımı desteklemektedir.

Halstead yönteminin zayıf yanları:

- Bütün/tamamlanmış koda bağımlıdır.
- Tahminsel hesaplama modeli olarak kullanımı çok azdır. Uygulama tasarım aşamasında McCabe modeli daha uygun gözükmektedir.

3.2.5 Çevrimsellik Karmaşıklığı

Çevrimsellik Karmaşıklığı (*Cyclomatic Complexity*) bir programın iç yapısı ile ilgilidir ve komutlardaki ardışıllıktan çok çevrim ve karar verme gibi noktaların karmaşıklığı etkilediği gerçeğine dayanır. Bu gibi noktaların toplam sayısı ile ilgili bir ölçümdür. Ayrıca program yapısı bir graf'a dönüştürüldüğünde bu sayının oluşacak çevrimlerin sayısı ile olan ilgisini gözlemlemek oldukça kolaydır. Örneğin bu teknik, programlama ödevlerini teslim eden öğrencilerin diğerlerinin ödevlerinden ne derece faydalandığını ortaya çıkarmak için kullanılmıştır: Programda yapıların yerleri değiştirilebilir ve farklı isimlendirmeler kullanılarak iki programdaki benzerlikler, göz ile anlaşılmaları çok zor bir duruma getirilebilir.

Çevrimsel karmaşıklık, bir karmaşıklık ölçüsü olarak McCabe tarafından geliştirilmiştir. Bir programın karmaşıklığını ölçen bir sistemdir. Bu sistem program içindeki bağımsız yolları saymaktadır. Diğer bir deyişle şart durumları sayılmaktadır.

Bir grafın (G) çevrimsel karmaşıklığı (CC) hesaplanması (3.16)'da gösterilmiştir.

$$CC(G) = \text{Kenar Sayısı} - \text{Düğüm Sayısı} + 1 \tag{3.16}$$

Birçok deneyde görülmüştür ki sonuçta eğer CC 5 ile 9 arasında ise hata oranı 0'a yaklaşmaktadır.

3.2.5.1 Çevrimsellik Karmaşıklığının Güçlü Yönleri

- Uygulamak kolaydır.
- Halstead ölçüt'lerine kıyasla yaşam döngüsünde daha erken hesaplanabilmektedir.
- Test için gereken minimum çaba ve en iyi alana yoğunlaşılmasını sağlar
- Test yapmaya rehberlik eder.
- Kolaylıkla bakım ölçüt'leri olarak kullanılabilinir.
- Çeşitli tasarım tiplerine göreceli karmaşıklık verebilmektedir ve kalite ölçüt'ü olarak kullanılabilmektedir.

3.2.5.2 Çevrimsellik Karmaşıklığının Zayıf Yönleri

- CC program kontrolü karmaşıklığını ölçer, veri karmaşıklığını değil.
- Kümelenmiş ve kümelenmemiş döngülerin ağırlığı eşittir. Bununla beraber derin kümelenmiş şartlı yapıları anlamak kümelenmemişlere kıyasla daha zordur.

- Yanıltıcı bir bakış açısıyla karşılaştırmalara ve karar yapılarına bakmamızı sağlayabilir. Bu açıdan toplanma-yayılma metodu daha uygun ve veri akışı için uygulanabilir bir yöntemdir.

3.2.6 Toplanma ve Yayılma Karmaşıklığı

Henry ve Kafura toplama ve yayılma karmaşıklığını tanımlamıştır. Bu bir bileşenin ve genel yapısındaki veri akışı sayısını bulmamızı sağlamaktadır. Veri akış sayısı değiştirme prosedür parametre sayısını ve bu modüllerden çağrılan prosedürleri saymaktadır (3.17).

$$\text{Karmaşıklık} = \text{Uzunluk} \times (\text{Fan-in} \times \text{Fan-out})^2 \quad (3.17)$$

Uzunluk, kod satır sayısı, McCabe CC olabilir. Henry ve Kafura ölçüt'lerini UNIX sistemi kullanarak test etmişlerdir.

3.2.7 COCOMO Modeli

1981 yılında, Dr. Barry Boehm "Software Engineering Economics" adlı kitabında COCOMO'yu (*Constructive Cost Model*) tanıttı. İsminin kısaltması ve yılı nedeniyle genellikle bu modele "COCOMO 81" denmektedir. Oldukça ilgi görmüş bir maliyet kestirim modelidir. Yazılım projesinin çaba, plan ve maliyetlerini tahminlemeye temel olması için önerilen algoritmik modeller vardır. Bunlar kavramsal olarak benzerdir fakat farklı parametre değerleri kullanırlar. Gereken çabanın, program büyüklüğünün bir üstel değerine bağlı olması prensibi ve endüstriden toplanan bilgiler ışığı altında geliştirilmiş bir kestirim metodu olan COCOMO (*Constructive Costing Model*) deneysel, gözlemsel bir modeldir. COCOMO 81, birçok yazılım projesinden veri toplanarak, ardından gözlemlerle uyuşacak en iyi formülü keşfetmek için bu verilerin analiz edilerek çıkarılmış bir modeldir. 1987 yılında, Yazılım Mühendisliği Enstitüsünde yapılan 3. COCOMO kullanıcı grup toplantısında Ada COCOMO ve artımsal COCOMO tanıtıldı. 1988, 1989 yollarında Ada COCOMO'da geliştirmeler tanımlandı.

3.2.7.1 COCOMO'nun seçilme nedenleri

1. COCOMO'da kullanılan projeler çeşitlilik göstermektedir. Farklı alanlarda yapılmış projelerdir.
2. COCOMO oldukça yaygın olarak kullanılmaktadır ve değerlendirilmektedir.
3. COCOMO'nun ele aldığı projeler iyi ele alınmış projelerdir. İyi seviyede dokümante edilmiştir.

4. COCOMO modeli iyi seviyede dokümante edilmiş bir modeldir. Model ayrıntılı olarak açıktır. COCOMO modeli hem ticari hem de ticari olmayan kurumlarca desteklenmektedir.
5. 1981'de ilk defa ortaya çıkmasından (Boehm B.W., 1981), Ada yazılım geliştirme için iyileştirilmesine, en yeni versiyonunun 1995'de yayınlanmasına kadar uzun bir soy ağacı mevcuttur. Böylece sürekli iyileştirme içinde olmuştur. Bu zamanda yaşanan gelişmelere paralel olarak sürekli güncelleştirilmiştir.

3.2.7.2 COCOMO 81

COCOMO modelinin ilk versiyonu (COCOMO 81), maliyet tahmini analizinin ayrıntılarını yansıtan üç seviyeli bir modeldir (Boehm B.W., 1981).

- Birinci seviye : Başlangıç ve kaba bir tahmin sağlanmakta
- İkinci seviye : Proje ve süreç çarpanlarını kullanarak bunu değiştirir
- Ayrıntılı seviye : Projenin farklı aşamaları için tahminler üretir

Yapılacak hesapların kapsamları açısından üç değişik modelden oluşur. Basit model, orta ve detaylı modeller. Ayrıca bu modellerde kullanılacak problemler, 'organik, yarık ayrık ve gömülü sınıflar altında toplanmıştır:

1. Organik problemler için küçük boyuttaki programcı takımları, bildikleri ortamlarda iyi anlaşılmış uygulamaları geliştirirler. İletişim problemleri azdır ve elemanlar hızlıca işlerini halledebilecek durumdadırlar.

2. Yarı ayrık problemlerde ise ekipte tecrübeli ve tecrübesiz elemanlar bulunabilir. İlgili sistemler konusunda deneyimleri sınırlı olabilir ve geliştirilen sistemin her kısmını bilmeyebilirler.

3. Gömülü problemler: Geliştirilecek yazılım, sistemin donanım, kurallar, işletim süreçleri veya yazılım gibi diğer bileşenleri ile çok kuvvetli bağlantılar oluşturur. Gereksinim değişiklikleri ile problemleri halletmek oldukça zordur. İhtiyaç belirtiminin geçerlilik incelenmesi çok maliyetlidir. Yazılımın geliştirilmesinde görev alanların tüm kısımlara hakim olma olasılığı iyice azalmıştır.

COCOMO bu model ve problem sınıfı saptamalarından sonra ortaya çıkan formüllerle tahmin hesaplama yolunu önerir. Çizelge 3.3'te problem sınıflarına göre basit model için formüller gösterilmektedir.

Çizelge 3.3 Basit COCOMO Modelleri

Problem	Çaba	Süre
Organik	Çaba = 2.4 $(KLOC)^{1.05}$	Süre = 2.5 $(Çaba)^{0.38}$
Yarı ayrık	Çaba = 3 $(KLOC)^{1.12}$	Süre = 2.5 $(Çaba)^{0.35}$
Gömülü	Çaba = 3.6 $(KLOC)^{1.20}$	Süre = 2.5 $(Çaba)^{0.32}$

Orta detaydaki model ise sistemin güvenilirlik, veri tabanı büyüklüğü, işletme ve kayıt sınırlandırmaları, personel özellikleri ve kullanılan yazılım araçları gibi diğer özelliklerinin hesaba katılması amaçlanmıştır. Çizelge 3.5'te orta detaydaki COCOMO ölçütleri gösterilmiştir. Belirli bir takım özelliğin, proje açısından etkileri ayrı ayrı ağırlandırılarak katsayılar ortaya çıkarılır. Çizelge 3.6'da yazılım geliştirme çaba çarpanları gösterilmiştir. Bu faktörler, ilgili özellik için düşük (<1), nominal (1) veya yüksek (>1) olarak saptanırlar. Katsayılar birbiri ile çarpıldığında "Çaba Ayarlama Katsayısı" (ÇAK) (*Effort Adjustment Factor*) bulunur. Bu katsayı 0.9 ile 1.4 arasında bir değer alır ve Çizelge 3.4'te gösterilen Orta COCOMO modeli formülleri kullanılarak çaba hesaplaması ile sonuçlandırılır. Süre hesaplaması ise Basit COCOMO modelinde olduğu gibi yapılır.

Çizelge 3.4 Orta Detayda COCOMO Çaba Formülleri

Problem	Çaba
Organik	Çaba = 3.2 $(KLOC)^{1.05}$ x ÇAK
Yarı ayrık	Çaba = 3.0 $(KLOC)^{1.12}$ x ÇAK
Gömülü	Çaba = 2.8 $(KLOC)^{1.20}$ x ÇAK

Çizelge 3.5 Orta Detayda COCOMO Ölçütleri

Ürün	Donanım	İnsan	Proje
Güvenilirlik	Hız	Analist yeteneği	Yazılım aracı kullanımı
Veri tabanı büyüklüğü	Bellek yeterliliği	Uygulama Deneyimi	Zamanlandırma
Ürün Karmaşıklığı	Sanal Makine Değişkenliği	Geliştirme Ortamı Deneyimi	Yeni Programlama Teknikleri
	Kullanılabilme Süresi	Yazılım Geliştirici Yeteneği	
		Programlama Dili Deneyimi	

Çizelge 3.6 Yazılım Geliştirme Çaba Çarpanı

Ölçüt	Açıklama	Değerlendirme					
		Çok Az	Az	Normal	Yüksek	Çok Yüksek	Aşırı Yüksek
Product							
RELY	Yazılımın Güvenilirliği	0.75	0.88	1.00	1.15	1.40	-
DATA	Veritabanı Hacmi	-	0.94	1.00	1.08	1.16	-
CPLX	Ürün Karmaşıklığı	0.70	0.85	1.00	1.15	1.30	1.65
Bilgisayar							
TIME	Çalışma zamanı kıstı	-	-	1.00	1.11	1.30	1.66
STOR	Ana depolama kıstı	-	-	1.00	1.06	1.21	1.56
VIRT	Sanal makine uçuculuğu	-	0.87	1.00	1.15	1.30	-
TURN	Bilgisayar değişim (dönme) zamanı	-	0.87	1.00	1.07	1.15	-
Personel							
ACAP	Analistin yetenekleri	1.46	1.19	1.00	0.86	0.71	-
AEXP	Uygulama Tecrübesi	1.29	1.13	1.00	0.91	0.82	-
PCAP	Programcı kapasitesi	1.42	1.17	1.00	0.86	0.70	-
VEXP	Sanal Makine Tecrübesi	1.21	1.10	1.00	0.90	-	-
LEXP	Dil Tecrübesi	1.14	1.07	1.00	0.95	-	-
Project							
MODP	Modern programlama deneyimi	1.24	1.10	1.00	0.91	0.82	-
TOOL	Yazılım Araçları	1.24	1.10	1.00	0.91	0.83	-
SCED	Geliştirme Zaman Planı	1.23	1.08	1.00	1.04	1.10	-

Detaylı COCOMO modeli ise projenin evrelerine bağlı olarak süreç içinde değişiklikleri hesaba katarak arada bir kestirim hesaplamasını önerir (Fenton, N.E. ve Pfleeger S., 1998). Bu modelde zamana bağlılık temel değişikliktir. Projenin farklı evrelerinde çaba yoğunluğu ve yapılacak işin karmaşıklığı değişecektir. Benzer bir anlayış, yazılım eğrisi denilen bir sonucu yansıtan Putnam modelinde de kendisini gösterir. Evrelere bağlılığın bir sonucu olarak Raleigh adam-ay eğrileri ortaya çıkmıştır. Proje başlangıcındaki az iş gücü ile gereksinimlerin ilk çabalarını düşük düzeylerle temsil edildiğini bu eğrilerde görebiliriz. Hızla tırmanan eğri, geliştirmenin analiz, tasarım ve uygulama gibi evrelerinde yükseklerdedir. Daha sonra geliştirme sonrası faaliyet azalır. İleride bakım ve onarım yine bazı geçici yükselmeler yapabilir.

3.2.8 COCOMO II

1995 yılında COCOMO II'yi tanıtan makaleler yayınlandı. 1997 yılında COCOMO II için ilk kalibrasyonlar yapıldı. 1998 yılında COCOMO II için ikinci kalibrasyonlar yapıldı. COCOMO II.1998 ismi COCOMO II.1999 olarak sonra COCOMO II.2000 olarak değiştirildi. Aslında üçü de tamamen aynıdır.

COCOMO 81, yazılımın şelale sürecine göre geliştirildiği düşünülerek geliştirilmiş bir modeldir. Bu başlangıç versiyonunun önerilmesinden sonra yazılım geliştirmede büyük değişikler meydana geldi. Yazılım, yeniden kullanılabilir bileşenlerin birleştirilmesi ve bunların bir betik bir dil aracılığıyla geliştirilmesi yoluyla yaratılabilir. Prototip oluşturarak ve artırımsal geliştirme çoğunlukla kullanılan bir süreç modelleridir. Birçok durumda var olan alt sistemler, kütüphaneler kullanılır. Yeni yazılım oluşturmak için var olan yazılım üzerine yeniden mühendislik yapılabilinmektedir.

Bu değişiklikleri hesaba katmak için COCOMO II prototip oluşturarak, bileşenleri birleştirerek geliştirme, dördüncü nesil programlama dillerini kullanma gibi farklı yaklaşımlar kabul etmiştir. Model seviyeleri, artan bir şekilde karmaşık ve detaylı tahminleri yansıtmamaktadır. Seviyeler, yazılım sürecindeki aktivitelerle ilişkilendirildiğinden, başlangıç tahminleri, sistem mimarisi tamamlandıktan sonra gerçekleştirilen daha detaylı tahminle birlikte süreçte erken yapılırlar. COCOMO II'de tanımlanan seviyeler:

1. Erken prototip oluşturma seviyesi: Büyüklük tahminleri nesne noktalarını temel alır ve gerekli çabayı tahminlemek için basit büyüklük/üretkenlik formülü kullanılır.

2. Erken tasarım seviyesi: Bu seviye, sistem ihtiyaçlarını bazı başlangıç tasarımıyla tamamlamaya karşılık gelir. Tahminler, kaynak kodun satır sayılarına dönüştürülecek işlev puanını temel alır. Formülde standart formüle ek olarak ona ilişkilendirilmiş bir çarpan kümesi mevcuttur.

3. Mimari sonrası seviyesi: Sistem mimarisi tamamlandığında, yazılım büyüklüğü hakkında mantıklı, doğru tahmin yapılabilir. Bu seviyedeki tahmin, personelin yeteneklerini, ürün ve proje özelliklerini yansıtan daha kapsamlı çarpan kümesi kullanır.

3.2.8.1 Erken prototip oluşturma seviyesi

Erken prototip oluşturma seviyesi, prototip oluşturularak gerçekleştirilen projeler ve var olan bileşenlerin birleştirilmesiyle oluşturulan projeler için gerekli çabayı tahminlemek amacıyla COCOMO'nun içinde mevcuttur (Fenton, N.E. ve Pfleeger S., 1998). Tahmin edilen

üretkenlik için standart sayıya bölünen ağırlıklı işlev puanları temel alınır. Programcı üretkenliği; geliştiricinin deneyimine ve yeteneğine ve geliştirmeyi desteklemek için kullanılan bilgisayar destekli yazılım geliştirme araçlarının yetenekleri ile ilişkilidir. Çizelge 3.7'de üretkenliğin farklı seviyelerini gösterilmektedir (Boehm B., Bradford C., Horowitz E., Madachy R., Shelby R., Westland C. ,1995).

Çizelge 3.7 COCOMO modelinde üretkenlik (Boehm B., Bradford C., Horowitz E., Madachy R., Shelby R., Westland C., 1995)

Yazılım Geliştiricinin deneyimi ve yeteneği	**Bilgisayar Destekli yazılım mühendisliği aracının olgunluğu ve yeteneği**	**Üretkenlik**
Çok Az	Çok Az	4
Az	Az	7
Normal	Normal	13
Yüksek	Yüksek	25
Çok Yüksek	Çok Yüksek	50

Bu seviyede, yeniden kullanım oldukça yaygındır. Bu yüzden plan tahmininde kullanılan işlev puanları beklenen yeniden kullanım yüzdesini hesaba katarak ayarlanmalıdır. Plan hesaplama için (3.18)'deki gibi formülize edilebilinir:

Çaba=(İşlev puan sayısı . (1 - %Yeniden Kullanım/100)) / üretkenlik (3.18)

3.2.8.2 Erken tasarım seviyesi

Erken tasarım seviyesinde üretilen tahminler, algoritmik modeller için standart formülü temel alırlar. Erken tasarım seviyesinde çaba (3.19)'daki şekilde hesaplanabilmektedir.

$$\text{Çaba} = A \times \text{Büyüklük}^{B} \times M \quad (3.19)$$

Boehm, A katsayısının bu seviyede yapılan tahminler için 2.5 olmasını önermiştir. Sistemin büyüklüğü kaynak kodun binler cinsinden satır sayısı olarak ifade edilir. Bu yazılımdaki fonksiyon noktalarının sayısını tahminleyerek ve bunu, farklı programlama dilleri için yazılım büyüklüğü ile fonksiyon noktalarını ilişkilendirici standart tabloyu kullanarak KSLOC'a (binler cinsinden satır sayısı) çevirerek hesaplanır. Bu büyüklük tahmini, üreteçlerle yaratılan veya yeniden kullanılan kod yerine el ile gerçekleştirilen kodu ifade eder.

B üssü, projenin büyüklüğü arttıkça, artan gerekli çabayı yansıtır. Bu COCOMO'nun ilk versiyonundaki gibi sabit değildir ama projenin yeniliğine, geliştirme esnekliğine, kullanılan

risk çözünürlük süreçlerine, geliştirme takımının bütünlüğüne ve organizasyonun süreç olgunluk seviyesine bağlı olarak 1.1 ile 1.24 arasında değişebilmektedir.

M çarpanı, ürün güvenilirlik ve karmaşıklığı, gerekli olan yeniden kullanım, platform zorluğu (PDIF), personel yeteneği (PERS), personel deneyimi (PREX), plan (SCED) ve destek araçları (FCIL)'nı içeren yedi proje ve süreç sürücülerini içeren basitleştirilmiş kümeyi temel alır. Bunlar, 1'in düşük değerlere karşılık geldiği, 6'nın yüksek değerlere karşılık geldiği 6 noktalı ölçekle tahminlenirler. Alternatif olarak, daha ayrıntılı çarpanların kullanıldığı mimari sonrası seviyesindeki değerler birleştirilerek de hesaplanabilir. Çaba hesaplama (3.20) ve (3.21)'deki gibi yapılabilir.

$$\text{Çaba}=A \times \text{Büyüklük}^{B} \cdot M + PMm \tag{3.20}$$

$$M = PERS \cdot RCPX \cdot RUSE \cdot PDIF \cdot PREX \cdot FCIL \cdot SCED \tag{3.21}$$

(3.21)'de ki PMm, kodun otomatik olarak yaratılırsa kullanılan faktördür. Bu yüzden, gerekli çaba (PMm) eşitlik (3.22) kullanılarak ayrıca hesaplanır ve el ile geliştirilen kod için hesaplanan çabaya eklenir.

$$PMm= (ASLOC \cdot (AT/100)) / ATPROD \tag{3.22}$$

ASLOC, otomatik olarak yaratılmış kaynak kodun satır sayısıdır. ATPROD, bu tipteki kod üretiminin üretkenlik seviyesidir. Üreteçlerle yaratılan kodun sistemin kalan kısmıyla arayüzünü oluşturmak için ek olarak çaba gerekmektedir. Bu, otomatik olarak oluşturulan kodun (AT), toplam sistem koduna yüzdesine bağlıdır. Gerçek üretkenlik, üreteçlerle yaratılan modül sayılarına bağlıdır. Üreteçlerle yaratılan kodun miktarı ne kadar azsa, bunu sistemin içindeki diğer kodlarla uyuşturmada ve tümleştirmede daha fazla çaba gerekmektedir.

3.2.8.3 Mimari sonrası seviyesi

Mimari sonrası seviyesinde üretilen tahminler, erken tasarım tahminlerinde kullanılan aynı temel formülü temel alırlar. Bununla birlikte, yazılım için büyüklük tahmini tahminleme sürecinde bu evre ile daha doğrudur ve başlangıç çaba hesabı yedi özellik yerine 17 özellik kullanılarak arttılmıştır.

Kaynak koddaki toplam satır sayısının tahmini iki önemli proje faktörünü hesaba katarak ayarlanmıştır:

- Mümkün olabilecek yeniden kullanımın büyüklüğü: Kapsamlı yeniden kullanım, geliştirilen kaynak kodun satırlarının sayısının değiştirilmesinin gerekli olduğu anlamına gelir. Bununla birlikte (Selby M., 1988), yeniden kullanım maliyetlerinin doğrusal olmadığını keşfetti. Bunun nedeni, gerekli olan bileşenleri keşfetmek ve seçmek için harcanan başlangıç çabası ve yeniden kullanılabilir bileşenlerin değiştirilmeye ihtiyacı varsa, onların ve arayüzlerinin anlaşılması için gerekli çabadır.

- İhtiyaçların değişkenliği: Tahmin, sistem ihtiyaçlarındaki değişiklikleri uzlaştırmak için gerekli yeniden çalışmadan oluşur. Bu tahmin, değiştirilmesi gerekli kaynak kodun satırlarının sayısı cinsinden ifade edilir ve bu sayı başlangıç büyüklük tahminine eklenir.

Yeniden kullanımın etkileri COCOMO II'de büyüklük çabasını ayarlayarak eşitlik (3.23)'e göre hesaba katılmıştır.

$$ESLOC=ASLOC \cdot (AA + SU + 0.4DM + 0.3CM + 0.3IM) / 100 \quad (3.23)$$

ESLOC, yeni kodun denk satırlarının sayısıdır. ASLOC, değiştirilmesi gerekli yeniden kullanılabilir kodun satırlarının sayısıdır. DM, değiştirilen tasarımın yüzdesidir. CM, değiştirilen kodun yüzdesidir. IM, yeniden kullanılabilir yazılımı tümleştirmek için gerekli orijinal tümleştirme çabasının yüzdesidir. SU, yazılım anlaşılabilirliğini temel alan bir değerdir. 50 (karmaşık, yapısal olmayan kodlar için) ile 10 (iyi yazılmış, nesneye dayalı kod) arasında değerler alabilir. AA, yazılımın yeniden kullanılabilir olabileceği hakkında verilen karar için başlangıç değer biçme maliyetlerini yansıtan bir değerdir. O, yapılan test miktarına ve gerekli olan değerlendirme miktarına bağlıdır. Değeri 0'dan 8'e kadar değişebilir.

Üstel terim, COCOMO'nun ilk sürümünde, çaba hesaplama formülünde 3 farklı değere sahip olabilmekteydi. Bunlar farklı seviyelerdeki proje karmaşıklığıyla ilişkilendirilmişti. Projeler karmaşıklaştıkça, artan sistem büyüklüğünün etkileri daha önemli hale gelmektedir. Bununla birlikte, organizasyonel uygulamalar ve prosedürler bu ekonomik olmayan ölçeği kontrol edebilirler ve bu COCOMO II'de kabul edilmişti. Üs, aşağıdaki şekilde gösterildiği gibi beş adet ölçüm faktörü göz önüne alınarak tahminlenir.

1. Sürecin olgunluğu
2. Geliştirme esnekliği
3. Öncüllük
4. Mimari/risk çözünürlüğü
5. Takım bütünlüğü

Bu faktörler, altı dereceli ölçekte sınıflandırılırlar. Üstel terimi elde etmek için, sonuçlanan

oranlar toplanır, 100'e bölünür ve sonuca 1.01 eklenir.

Mimari sonrası modelindeki başlangıç tahminlerini ayarlamak için kullanılan özellikler dört sınıfa karşılık gelirler:

1.Ürün özellikleri:

Ürün özellikleri, geliştirilen yazılım ürününün gerekli özellikleri ile (Çizelge 3.8) ilgilidir.

Çizelge 3.8 COCOMO II Ürün özellikleri

Ölçüt	Açıklama
RELY	Sistem güvenilirliği
DATA	Veritabanının büyüklüğü
CPLX	Modüllerinin karmaşıklığı
DOCU	Gerekli belgelenmenin büyüklüğü
RUSE	Yeniden kullanılabilir bileşenlerin yüzdesi

2. Bilgisayar özellikleri

Bilgisayar özellikleri (Çizelge 3.9), donanım platformu tarafından yazılıma verilen kısıtlar.

Çizelge 3.9 COCOMO II Bilgisayar özellikleri

Ölçüt	Açıklama
TIME	Çalışma zamanı kısıtları
STOR	Bellek kısıtları
PVOL	Platformun uçuculuğu

3.Personel özellikleri

Personel özellikleri (Çizelge 3.10), projede çalışanların deneyimlerini ve yeteneklerini hesaba katan çarpanlardır.

Çizelge 3.10 COCOMO II Personel özellikleri

Ölçüt	Açıklama
PCAP	Programcı yeteneği

ACAP	Proje analistinin yeteneği
PCON	Personel devamlılığı
AEXP	Proje sahasında analist deneyimi
PEXP	Proje sahasında programcı deneyimi
LTEX	Dil ve araç deneyimi

4.Proje özellikleri

Proje özellikleri (Çizelge 3.11), yazılım geliştirme projesinin belirli özellikleri ile ilgilidir.

Çizelge 3.11 COCOMO II Proje özellikleri

Ölçüt	Açıklama
TOOL	Yazılım araçlarının kullanımı
SITE	Çok siteli çalışma büyüklüğü ve site haberleşmelerinin kalitesi
SCED	Geliştirme planı sıkıştırma

COCOMO II modelinin geliştiricileri tarafından önerilen bu formül, onların deneyimlerini ve verisini yansıtır ama pratik kullanım için çok karmaşık olmaktadır. Onların değerlerini tahminlemek için çok fazla özellik ve çok fazla saha vardır. Prensip olarak, modelin her kullanıcısı modeli ve modeli etkileyen yerel olayları yansıtan kendi tarihsel proje verisine göre özellik değerlerini ayarlamalıdır. Ancak çok az sayıda organizasyon, model ayarlamasını destekleyen formda geçmiş projelerinden yeterli veri toplamıştır.

3.3 Ölçüt karşılaştırması

Tahminleme çalışmalarda genelde COCOMO 81 kullanılmıştır. COCOMO 81 bir ölçüt bazdır. Ancak bunun yanında önemli ölçüt setlerini karşılaştırmaları çizelge 3.12'de gösterilmektedir .

Çizelge 3.12 Önemli ölçüt kümelerinin karşılaştırması

Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
Personel						
ACAP	Analist Yeteneği (*Analyst Capability*)	Evet	Hayır	Evet	Evet	Evet
APEX AEXP	Uygulama Tecrübesi (*Applications Experience*)	Evet	Hayır	Evet	Evet	Evet
PCAP	Programcı Yeteneği (*Programmer Capability*)	Evet	Hayır	Evet	Evet	Evet
LEXP	Programlama Dili Tecrübei (*Programming Language Experience*)	Hayır	Hayır	Evet	Evet	Evet
VEXP	Sanal Makine Tecrübesi (*Virtual Machine Experience*)	Hayır	Hayır	Evet	Evet	Evet
CPLX	Yazılım Ürünü Karmaşıklığı (*Software Product Complexity*)	Hayır	Evet	Hayır	Hayır	Hayır
PREX	Personel Tecrübesi (*Personnel Experience*)	Hayır	Evet	Hayır	Hayır	Hayır
PCON	Personel Sürekliliği (*Personnel Continuity*)	Evet	Hayır	Hayır	Hayır	Hayır
PLEX PEXP	Platform Tecrübesi (*Platform Experience*)	Evet	Hayır	Hayır	Hayır	Hayır
LTEX	Dil ve Aracı Tecrübesi (*Language and Tool Experience*)	Evet	Hayır	Hayır	Hayır	Hayır
Ürün						
RELY	Gereken Yazılım Güvenilirliği (*Required Software Reliability*)	Evet	Hayır	Evet	Evet	Evet
DATA	Veritabanı Büyüklüğü (*Database Size*)	Evet	Hayır	Evet	Evet	Evet

CPLX	Yazılım Ürünü Karmaşıklığı (*Software Product Complexity*)	Evet	Hayır	Evet	Evet	Evet
RUSE	Gereken Yeniden Kullanılabilinirlik (*Required Reusability*)	Evet	Evet	Evet	Evet	Hayır
DOCU	Yaşam Döngüsü ihtiyacına uygun dökümantasyon (*Documentation Match to Life-Cycle Needs*)	Evet	Hayır	Hayır	Hayır	Hayır
RCPX	Ürün Doğruluğu ve Karmalıklığı (*Product Reliability and Complexity*)	Hayır	Evet	Hayır	Hayır	Hayır
Platform						
TIME	Yürütme Zamanı Kısıtı (*Execution Time Constraint*)	Evet	Hayır	Evet	Evet	Evet
STOR	Ana Depolama Kısıtı (*Main Storage Constraint*)	Evet	Hayır	Evet	Evet	Evet
TURN	Bilgisayar İş Bitirme Zamanı (*Computer Turnaround Time*)	Hayır	Hayır	Evet	Evet	Evet
VIRT	Sanal Makine Uçuculuğu (*Virtual Machine Volatility*)	Hayır	Hayır	Evet	Hayır	Evet
VMVH	Sanal Makine Uçuculuğu : Ana Bilgisayar (*Virtual Machine Volatility: Host*)	Hayır	Hayır	Hayır	Evet	Hayır
VMVT	Sanal Makine Uçuculuğu : Hedef (*Virtual Machine Volatility: Target*)	Hayır	Hayır	Hayır	Evet	Hayır
PVOL	Platform Uçuculuğu (*Platform Volatility*)	Evet	Hayır	Hayır	Hayır	Hayır
PDIF	Platform Zorluğu (*Platform Difficulty*)	Hayır	Evet	Hayır	Hayır	Hayır
PLAT	Platform	Hayır	Hayır	Evet	Hayır	Hayır
Proje						
TOOL	Yazılım Araçlarının Kullanımı (*Use of Software Tools*)	Evet	Hayır	Evet	Evet	Evet

MODP	Modern Programlama Pratiği (*Modern Programming Practices*)	Hayır	Hayır	Evet	Evet	Evet
SCED	Gereken Geliştirme Zaman Planı (*Required Development Schedule*)	Evet	Evet	Evet	Evet	Evet
SECU	Sınıflandırılmış Güvenlik Uygulaması (*Classified Security Application*)	Hayır	Hayır	Evet	Evet	Hayır
SITE	Çokyerli Geliştirme (*Multisite Development*)	Evet	Hayır	Hayır	Hayır	Hayır
FCIL	Araç Gereç (*Facilities*)	Hayır	Evet	Hayır	Hayır	Hayır
RVOL	Gereken Uçuculuk (*Requirements Volatility*)	Hayır	Hayır	Evet	Hayır	Hayır

3.4 Karşılaştırma

Yazılım maliyet tahminleme yöntemleri iki temel çatıda ele alınmaktadır: algoritmik modeller ve algoritmik olmayan modeller. Çokça kullanılan modelleri değerlendirdiğimizde:

- Hiç biri tüm projelere uygun olamıyor.
- Parkinson ve kazanmak için fiyat (Price-to-win) metotları gelişmek, büyümek isteyen firmalar için uygun değildir.
- Bu yöntemlerin kombinasyonunu kullanmak daha iyi bir sonuç verebilir. Örneğin uzman görüşü ile yukarıdan aşağı model ile benzeşim bazlı yöntemlerin birleştirilmesi iyi bir sonuç verebilir.

Algoritmik modeller sınıflandırıldığında aşağıdaki gibi bir tablo çıkmaktadır.

Çizelge 3.13 Algoritmik modellerin sınıflandırılması

Algoritmik Modeller					
	Lineer	Çarpanlı	Üssel	Ayrık	Diğer
Gözlemsel	Nelson	Walston-Felix	COCOMO	Aron,Boeing, Wolverton	Prise-S
Analitik			Putnam		Softcost

Çokça kullanılan bu modellerin birbirlerine göre güçlü ve zayıf yanları vardır. Bunlar çizelge 3.14'te gösterilmiştir.

Çizelge 3.14 Karşılaştırma

	Metot	**Güçlü Yanları**	**Zayıf Yanları**
Algoritmik olmayan	Uzman Görüşü	İyi bir uzman bulunabilirse göreceli olarak iyi bir tahminde bulunabilir	Uzman'a göre değişir
		Hızlıdır	Tutarlı olamayabilir
			Eksik veriden dolayı yeniden tahminlemeler yapılır
	Benzeşim	Gerçek projeler ve geçmiş tecrübe temeli üzerine kurulur	Benzer proje olmayabilir
			Tarihsel veri doğru olmayabilir
	Parkinson	Kontrat'ı genellikle kazanır	Zayıf alıştırma/uygulama
			Çok fazla bitiş süresini/sınırı aşar
	Yukardan-Aşağı (Tsuneo Y. ve Tohru K., 1999)	Sistem seviyesinde bir odaklanma	Düzenlemeler konusunda az detay vardır
		Aşağıdan-yukarıya'ya nazaran daha kolay ve daha hızlıdır	Diğer metotlara nazaran tutarlılığı azdır
		Çok az detay gerektirir	
	Aşağıdan-Yukarı	Detaylı analize ihtiyaç duyar	Maliyet faktörlerine daha fazla bakmak gerektirir
		Proje izlemede diğerlerine göre avantaj sağlar ve alt seviyede maliyetleri bulmakta olanak sağlar	Daha fazla çaba gerektirir
			Başlarda tahminleme yapmak zordur
Algoritmik	Algoritmik	Objektiftir. Tekrarlanabilir sonuçları vardır	Girdiler objektif değildir
		Tahminleme yöntemini anlamakta ve açıklamakta daha iyi bir yoldur	Geçmiş projelerin kullanılması mevcut duruma (ortama) uygun olmayabilir
			Genellemeler yapmak zordur

4. YAZILIM MALİYET TAHMİNLEMEDE YAPAY SİNİR AĞLARININ KULLANIMI

Yapay Zekâ kullanarak maliyet tahminleme aslında oldukça karmaşık bir problemdir. Ancak bugüne kadar gereken ilgiyi görmemiştir . Araştırmacılar değişik yaklaşımlar getirmeye çalışmışlardır. Son zamanlarda yeni yapay zekâ yaklaşımları düşünülmeye başlanmıştır.

Bu çalışmada yapay sinir ağları bir araç olarak kullanılmıştır. Bu bölümde yapay sinir ağlarından kısaca bahsedilmiş, genel olarak yapay sinir ağları ve kullandığımız modellere değinilmiştir.

4.1 Yapay Sinir Ağları

Yapay Sinir Ağları, beynin fizyolojisinden yararlanılarak oluşturulan bilgi işleme modelleridir. Sinir sistemi ile yapay sinir ağı benzerliklerin çizelge 4.1'de gösterilmiştir. 100'den fazla yapay sinir ağı modeli vardır (Heaton J.,2005). Bazı bilim adamları, beynimizin güçlü düşünme, hatırlama ve problem çözme yeteneklerini bilgisayara aktarmaya çalışmışlardır. Diğer bir grup ise, beynin fonksiyonlarını kısmen yerine getiren birçok modelleri oluşturmaya çalışmışlardır.

Yapay Sinir ağlarının öğrenme özelliği, araştırmacıların dikkatini çeken en önemli özelliklerden birisidir. Çünkü herhangi bir olay hakkında girdi ve çıktılar arasındaki ilişkiyi, doğrusal olsun veya olmasın, elde bulunan mevcut örneklerden öğrenerek daha önce hiç görülmemiş olayları, önceki örneklerden çağrışım yaparak ilgili olaya çözümler üretebilme özelliği Yapay Sinir Ağlardaki zeki davranışın da temelini teşkil eder.

Çizelge 4.1 Sinir sistemi ile Yapay Sinir Ağı benzerlikleri

Sinir Sistemi	Yapay Sinir Ağı Sistemi
Nöron	İşlem elemanı
Dendrit	Toplama fonksiyonu
Hücre gövdesi	Transfer fonksiyonu
Aksonlar	Eleman çıkışı
Sinapslar	Ağırlıklar

Bu çalışmada daha çok MLP ve *Elman* kullanıldığından bu konulara kısaca 4.1.1 ,4.1.2 ve 4.1.3'te değinilmiştir.

4.1.1 Çok Katmanlı Algılayıcılar

Çok katmanlı algılayıcılar (*MLP: Multi-Layer perceptron)* sinir ağı modeli özellikle mühendislik uygulamalarında en çok kullanılan sinir ağı modeli olmuştur (Heaton J., 2005). Birçok öğrenme algoritmasının bu ağı eğitmede kullanılabilir olması, bu modelin yaygın kullanılmasının sebebidir. Bir MLP modeli, bir giriş, bir veya daha fazla ara ve bir de çıkış katmanından oluşur. Bir katmandaki bütün işlem elemanları bir üst katmandaki bütün işlem elemanlarına bağlıdır. Bilgi akışı ileri doğru olup geri besleme yoktur. Bu yüzden ileri beslemeli sinir ağı modeli olarak adlandırılır. Giriş katmanında herhangi bir bilgi işleme yapılmaz. Buradaki işlem elemanı sayısı tamamen uygulanan giriş sayısına bağlıdır. Ara katman sayısı ve ara katmanlardaki işlem elemanı sayısı ise genellikle deneme ve yanılma yolu ile bulunmaktadır. Çıkış katmanındaki eleman sayısı ise yine uygulanan probleme dayanılarak belirlenir. Biz çalışmalarımızda 1 olarak aldık.

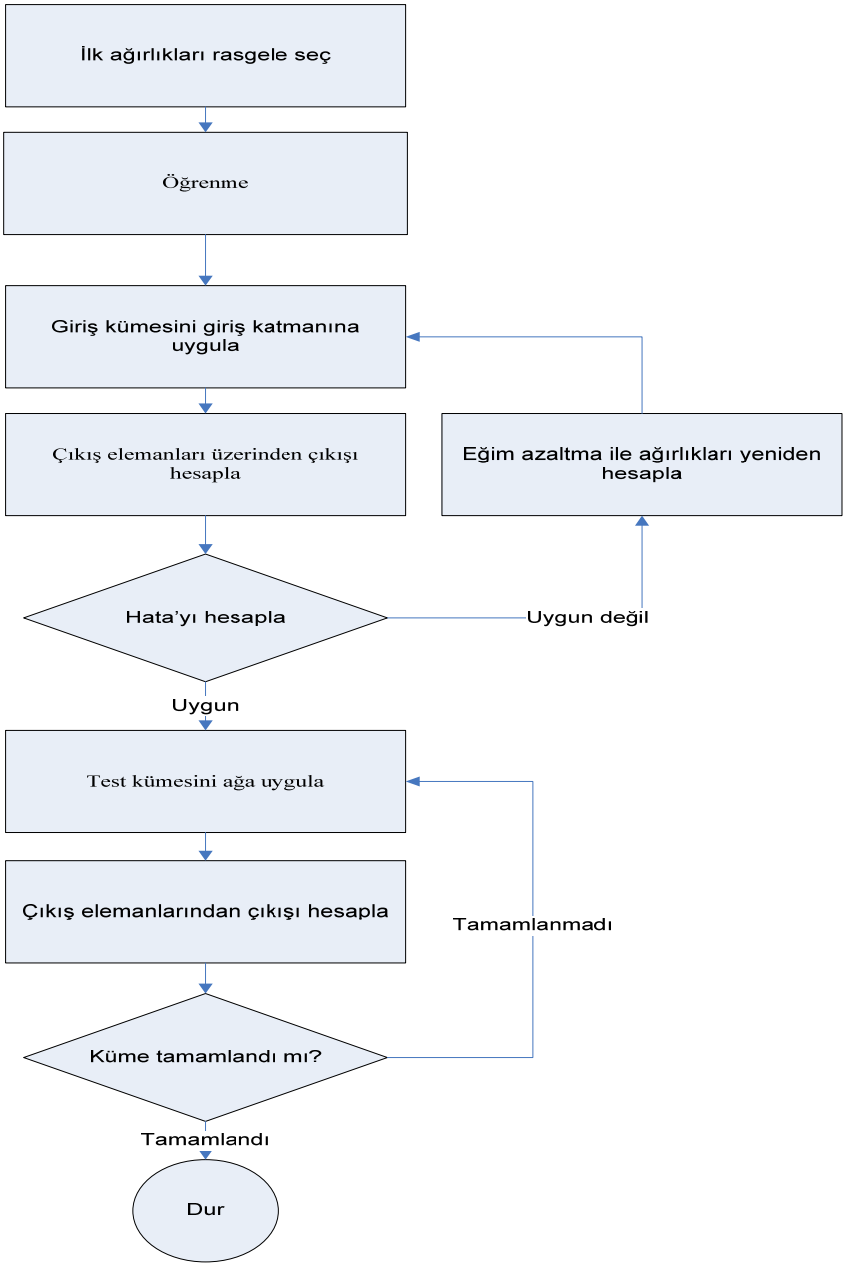
MLP ağlarında, ağa bir örnek gösterilir ve örnek neticesinde nasıl bir sonuç üreteceği de bildirilir. Örnekler giriş katmanına uygulanır, ara katmanlarda işlenir ve çıkış katmanından da çıkışlar elde edilir. Kullanılan eğitme algoritmasına göre, ağın çıkışı ile beklenen çıkış arasındaki hata tekrar geriye doğru yayılarak hata minimuma düşünceye kadar ağın ağırlıkları değiştirilir.

İleri beslemeli ağlar, en genel anlamıyla giriş uzayıyla çıkış uzayı arasında statik haritalama yapar. Bir andaki çıkış, sadece o andaki girişin bir fonksiyonudur.

4.1.2 Geri Yayılım Algoritması

Birçok uygulamalarda kullanılmış en yaygın öğrenme algoritmasıdır. Anlaşılması kolay ve matematiksel olarak ispatlanabilir olmasından dolayı en çok tercih edilen öğretme algoritmasıdır. Bu algoritma, hataları geriye doğru çıkıştan girişe azaltmaya çalışmasından dolayı geri yayılım ismini almıştır.

Tipik çok katlı geri yayılım ağı bir giriş tabakası, bir çıkış katmanı ve en az bir saklı katmana sahiptir. Saklı katmanların sayısında teorik olarak bir sınırlama yoktur. Geri yayılım algoritması, eğim azalan ve MLP'leri eğitmede en çok kullanılan temel bir algoritmadır. MLP'nin genel algoritması şekil 4.1'de gösterilmektedir.

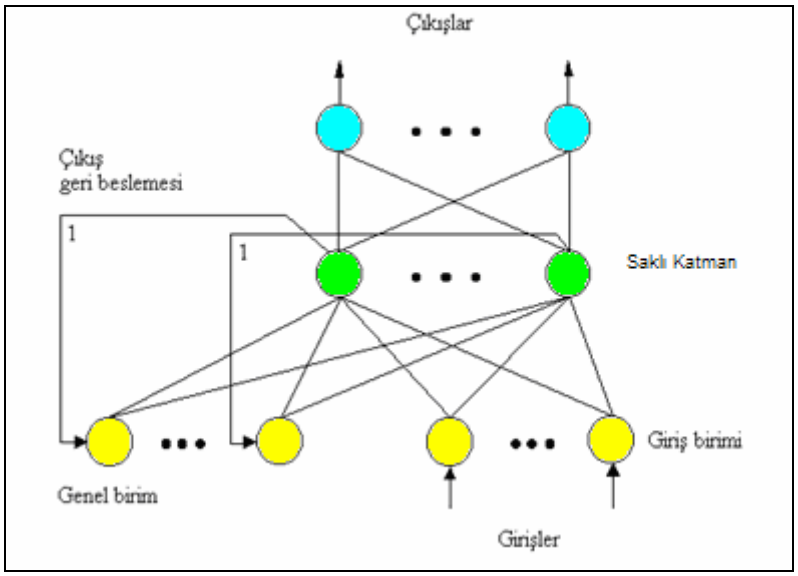


Şekil 4.1 MLP (Çok katmanlı bir algılayıcı) geri yayılım akış şeması

4.1.3 Elman Ağı

Elman ağı MLP'lere benzer bir yapıdadır ve çok katlıdırlar (Heaton J., 2005). *Elman*'da saklı katmana ek olarak diğer bir "durum" katmanı denilen özel bir saklı katman daha vardır (Şekil 4.2). Bu katman saklı katmandan veya çıkış katmanından geri besleme işaretleri alır. Durum katmanındaki işlemci elemanların çıkışları ileriye doğru saklı katmana verilmektedir Eğer sadece ileri doğru bağlantılar göz önüne alınır ve geri besleme bağlantılarına sabit değerler

verilirse, bu ağlar sıradan ileri beslemeli ağlar haline gelirler.



Şekil 4.2 Elman ağı

4.2 Yazılım Maliyet Tahminlemede Yapay Sinir Ağları Kullanılarak Yapılan Uygulamalar

Yazılım maliyeti için yapay sinir ağı kullanımı son 10 yıldır var olan ancak çok yavaş ilerleyen bir konudur. Bu bölümde bu alanda yapılmış bazı çalışmalar hakkında bilgi verilecektir. Yapılan önemli çalışmaları gösteren özet durum bu bölümün sonunda gösterilmektedir.

4.2.1 Wittig ve Finnie Çalışması

Bu konuda yapılan en ünlü çalışma Wittig ve Finnie'nin yaptığı çalışmadır (Wittig, G. ve G. Finnie, 1997). Avusturya Yazılım Ölçüt Birliğinin (ASMA) verileri kullanılmıştır. Yapay sinir ağı kullanılmıştır. Back-propagation yapısı seçilmiştir. Bu tez çalışmasında çıkan sonuç bu en başarılı çalışma ile karşılaştırılmıştır. Wittig ve Finnie'nin yaptığı bu çalışmada 7 girdi ve 1 çıktı kullanmışlardır. 136 örnek üzerinde çalışmışlardır. Öğrenme oranı olarak 0.3 ve momentum katsayısı olarak 0.6 alınmıştır. Sadece bir ara katman kullanmışlardır. Gaussian ve Sigmoid fonksiyonları kullanmışlardır. Ancak en iyi sonucu verebilecek katman sayısı bu olmayabilir. Hata oranı olarak MMRE (*Median of Magnitude of Relative Error*) 0.17'a ulaşmıştır. Diğer bir deyişle tahmin edilenden gerçekleşenin çıkarılmasının gerçekleşene

bölünmesinin ortalaması 0.17'dır. Yani %17 hata oranı mevcuttur. Bu çalışmanın detaylarını inceledik. Benzer bir ağı oluşturduk. Girdi sayısını artırarak, ara katman sayısını artırarak ve kullanılan yöntemler üzerinde çalışarak bu oranı düşürmenin mümkün olabileceğini biz düşündük.

4.2.2 Gary D. Boettiche Çalışması

Bu konuda yapılmış ikinci önemli çalışma Gary D. Boetticher'in yaptığı çalışmadır (Boetticher G.D, 2003). GUI bazlı ölçüt'ler ele alınmıştır. Klasik bir yapay sinir ağıdır ve geri-yayılım (*back-propagation*) kullanılmıştır. Yapıda 12 giriş ve sadece bir çıkış mevcuttur. 4 ara katman kullanılmıştır. Bu katmanlarda 5,11,3,5 düğüm mevcuttur. Baştan öğrenme oranı olarak 1 ve momentum sabiti olarak 1 verilmiştir. Bu çalışmada sadece kullanıcı arayüzü ölçütleri kullanılmıştır. Sadece bu ölçütlerin kullanılması yetersiz olabilir. Bu başarımı düşürmektedir. Kullanıcı arayüzü bazlı ölçütler kullanılabilineceği gibi bunun yanında başka ölçütlerinde kullanılması başarımı artıracaktır.

4.2.3 Ali Idri Çalışması

Ali Idri, Taghi M. Khoshgoftaar ve Alain Abran'ın yaptığı çalışma bu konuda yapılmış diğer bir çalışmadır (Idri A, Khoshgoftaar T.M. ve Abran A.,2002). Ölçüt olarak COCOMO '81 veri kümesini kullanmışlardır. 17 giriş ve 1 çıkış mevcuttur. Bir ara katman kullanılmıştır. 40 proje için eğitildiğinde MMRE %92.53 ve 62 proje için eğitildiğinde %53.67 oranlarını oranı elde edilmiştir. Ancak bu çalışmanın eksik gördüğümüz noktaları:

- Topolojinin seçimi rasgele yapılmıştır.
- Ölçüt veritabanı olarak hiç araştırma yapılmamış ve doğrudan COCOMO '81 alınmıştır. Test verisi olarakta gerçek hayattan projeler değil 63 taneCOCOMO '81 verileri kullanılmıştır.
- Başarım oranı düşüktür. Artırma çalışmaları yapılmamıştır.
- Hiç araştırma yapmadan sigmoid fonksiyonunu kullanmıştır, alternatifler incelenmemiştir.
- Çıkarım metodu olarak Benitez metodu kullanılmış, alternatifler araştırılıp kıyaslama yapılmamıştır

4.2.4 COBRA çalışması

Öğrenen bir sistemin avantajı parametrik olmaması ve adapte olabilirliğidir. Briand, El Eman ve Bomarius ismi COBRA hibrit bir model ortaya koymuşlardır. Cost Estimation,

Benchmarking and Risk Analysis'ın kısaltılmış halidir. Küçük bir veri seti kullanmışlardır (L. C. Briand, K. El Eman, F. Bomarius, 1998). Uzman bilgi ve nicel projeler üzerinde çalışmışlardır.

4.2.5 Kirsten Çalışması

Bir tezde (Kirsten R., 2001) nesne tabanlı yazılımla projelerinde kullanım senaryolarının kullanımı ile ilgili bir tahminleme çalışmasında iki tür faktör ortaya konulmuştur. Bunlar çizelge 4.2 ve 4.3'te gösterilmiştir. Ölçüt kümemizi oluştururken bu ölçütlerden yararlanmaya çalışılmıştır.

Çizelge 4.2 Teknik Karmaşıklık Faktörleri

Faktör	Tanım	Ağırlık
T1	Dağıtık sistem	2
T2	Yanıt durumu	2
T3	Son kullanıcı etkinliği	1
T4	Karmaşık işleme	1
T5	Tekrar kullanılabilinir kod	1
T6	Kolay kurulum	0.5
T7	Kolay kullanım	0.5
T8	Taşınabilir	2
T9	Değişiklik yapabilme kolaylığı	1
T10	Eş zamanlı	1
T11	Güvenlik Özellikleri	1
T12	Üçüncü partiler için erişim	1
T13	Özel eğitim gereksinimi	1

Çizelge 4.3 Çevre Faktörleri

Faktör	Tanım	Ağırlık
F1	RUP bilinirliği	1.5
F2	Uygulama tecrübesi	0.5
F3	Nesne tabanlı tecrübe	1
F4	Analist yetkinliği	0.5
F5	Motivasyon	1

F6	Kararlı gereksinimler	2
F7	Yarı zamanlı çalışanlar	-1
F8	Zor programlama dili	2

- Eşitlik (4.1) ve (4.2)'den gösterildiği gibi teknik karmaşıklık faktörü (TF) ve çevresel faktör (ÇF) hesaplanmaktadır.

$$TF=0.6+(0.01 \times \text{Teknik Faktörler}) \quad (4.1)$$

$$ÇF= 1.4+(-0.03 \times \text{Çevresel Faktörler}) \quad (4.2)$$

- Daha sonra projede çalışan aktörleri saptamak için toplam ayarlanmamış aktör ağırlığı (AAA) hesaplanıyor. Basit, Ortalama, Karmaşık olarak sınıflandırılan gruplarda her gruptaki eleman sayısı ile basit için 1, ortalama için 2 ve karmaşık için 3 ile çarpılıp toplanır.

- Daha sonra benzer şekilde Ayarlanmamış kullanım durumu ağırlığı (AAKDA) hesaplanır. Üç tür mevcuttur

  - Basit: 3 veya daha az işlem (Ağırlık çarpanı = 5)
  - Ortalama: 4 ile 7 arası işlem (Ağırlık çarpanı = 10)
  - Karmaşık: 7'den daha fazla işlem (Ağırlık Çarpanı = 15)

  Bu türlere göre her kullanım senaryosuna puan verilir ve puanlar toplanır.

- Ayarlanmamış kullanım durumu puanı (AAKDP) eşitlik (4.3)'teki gibi hesaplanır.

$$AAKDP = AAA + AAKDA \quad (4.3)$$

- Ayarlanmış kullanım durumu puanı (AKDA)eşitlik (4.4)'teki gibi hesaplanır.

$$AKDP = AAKDP \cdot TF \cdot ÇF \quad (4.4)$$

- En sonunda AKDA'den gerçek maliyet çıkarılır.

Bir yazılım geliştirme işi aslında programlama ile tamamlanmaz, sonrasında operasyonel bir kısmı da en az programlama kadar bir iş gücü gerektirir. Net Mevcut Değer ile ilgili yayınlanmış bir makale (Erdoğmus H. , 1999) buna değinmiştir.

T : Programlama için gereken zaman

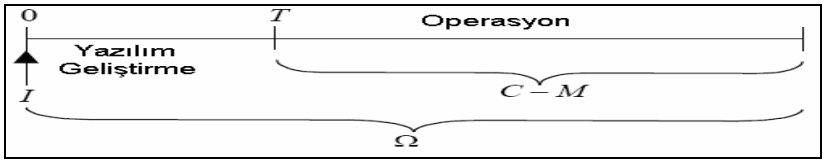
I : Programlama maliyeti

C : T anında bu programlamadan beklenen mali durum (Ciro)

M : Operasyonel Maliyet

Ω : Esneklik değeri

d : Risk



Şekil 4.3 Net Mevcut Değer

Net Mevcut Değer (NMD) (Şekil 4.3) eşitlik (4.5)'te belirtildiği şekilde hesaplanabilinmektedir.

$$NMD = (C - M) / (1 + d)^{T} - I + \Omega \quad (4.5)$$

Diğer bir ifadeyle (4.6)'da belirtilen eşitlikle hesaplanabilinmektedir.

NMD = (Operasyon maliyeti çıktıktan sonra kalan yani kar) / (1 + risk) $^{\text{DevelopmentZamanı}}$

– Geliştirme Maliyeti + Esneklik (4.6)

NMD'i doğrudan ölçüt olarak kullanamayız çünkü yazılım bittikten sonra hesaplanabilinecektir. Ancak Tahmini/Beklenen NMD önemli bir ölçüt olabilir. NMD'in yüksek olması bitirilme/başarım arasında bir korelâsyon olduğunu düşünülebilinir.

4.2.6 Yukarıdan-Aşağı Maliyet Tahmini

Yukarıdan-Aşağı maliyet tahmini yöntemi önemli tahminleme yöntemlerinden biridir. Bu konuda yayınlanmış olan bir makalede (Tsuneo Y., Tohru K.,1999) kategorizasyondan bahsedilmektedir. Benzerlik ile maliyet tahminlemesi ve dolayısıyla adam x saat hesaplaması oldukça kullanılan bir yöntemdir. İşlem adımları da oldukça basittir. Geçmişte yapılmış benzer bir projeyi baz alıp KLOC ve adam x saat'i tahmin etmek üzerine kurulmuştur. Bunun için tecrübe şarttır. Aşağıdan yukarıya (*Bottom-Up*) yöntemlerde ise (Örneğin İşlev Puan Metodu) fonksiyonelitelerle ilgilenir ve matematikseldir.

Örneğin

FP sayısı = f(fonksiyonların toplamı)

Tahmin edilen maliyet = f(FP, ortam)

şeklindedir.

Ancak FPM hedeflenen yazılım için derinlemesine bilgi gerektirmektedir. Bir uzman için bile FP saymak oldukça zaman almaktadır. Ve bir başka nokta ise hedeflenen sistem tamamlanmış olmalıdır.

TCE yaklaşımında (Top-Down Cost Estimation) temel adımları (Tsuneo Y., Tohru K.,1999) :

1. Yazılımlar için fonksiyoneliteye göre bir çizelgenin araştırmasının yapılmalıdır. Bu çizelgede maliyet, adam x saat bulunmalıdır.

2. Birinci maddede bulunan değeri öncelik ve kalite kriterine göre ayarlamalıdır.

3. Yazılım geliştirme ortamına göre yeniden ayarlamak

şeklindedir.

TCE'nin iki temel kabulü vardır.

1. Her yazılımın kendine has karakteristiği vardır. Örneğin fonksiyonel karmaşıklığı, performans gerekliliği, kullanıcı arayüzü gelişmişliği, vs vardır.

2. Yazılım geliştirme maliyeti ve adam x saat ihtiyacı bu yazılım karakteristiğinden, yazılım geliştirme ortamından etkilenir.

Benzeşimle kestirme yöntemlerinde (örneğin COCOMO ve İP metot) benzer maliyette gerçekleşeceğini düşünülür. Ancak her yazılımın kendi karmaşıklığı vardır.

Maliyeti etkileyen 3 temel faktör vardır:

1. Yazılım karakteristiği (Fonksiyonel karmaşıklık, performans gerekliliği)

2. Stratejik karakteristikler ("Kullanıma alınması sonra hataların giderilmesi" , "Hataların şimdi giderilmesi sonra devreye alınması" gibi)

3. Yazılım geliştirme ortamı karakteristikleri (Kullanılan yazılım geliştirme araçları ve donanım)

- İnsan kaynakları (çalışanların sayısı, yetenekleri, …).
- Donanım kaynakları (makineler, disk, ağ bant genişliği, ...).
- Yazılım kaynakları (yazılım geliştirme araçları, tekrar kullanılabilinirlik durumu, …).
- Yazılım geliştirme stratejisi (Spiral, Şelale, Prototipleme, …).

Buna göre

Standart Maliyet = f1(karakteristikler)

Tahmin edilen maliyet = f2(standart maliyet, strateji, ortam)

Adım 1:

TCE'ye göre öncelikle her türlü grubu kapsayan bir sınıflandırma (taksonomi) tablosu yapılmalıdır.

i.) İşletim sistemi: İş yönetimi, veri yönetimi, aygıt sürücüleri

ii.) Sistem araçları: Güvenlik, dosya yönetimi, kütüphane yönetimi

iii.) Ağ: Internet, C/S, ağ yönetimi, dağıtık objeler, ağ protokolü

iv.) Dil işleyiciler : Java, C, Fortran, …

v.) Veritabanı: Ağaç yapıları, İlişkisel veritabanı, dağıtık veritabanı

vi.) PC ile ilgili uygulamalar: Kelime işlemci, tablo işlemci,…

vii.) Uygulamalar : Bankacılık sistemi, rezervasyon sistemi, depo sistemi, elektronik ticaret,…

Adım 2:

Daha sonra her tür için standart maliyet çıkarılmalıdır ve amaca göre ağırlıklandırma yapılmalıdır (örneğin performans önemi…).

GUI amacına göre ağırlıklandırma yapılmalıdır.

Adım 3:

Düzeltme prosedürleri geliştirilmelidir. Ortama göre hangi durum gerçekleşirse hangi düzeltme katsayısı ile çarpılmalı şeklinde belirlenmelidir.

Adım 4:

TCE'nin tahminlemesi test edilmelidir.

Sonuç olarak yazılım hakkında detaya fazla girmeden kategorileri bulup ortam ve stratejik ağırlıklara göre sonuca ulaşmak bu yaklaşımın özetidir. Ortam ve stratejinin öneminin altını çizilmesi gereken noktalardır.

4.2.7 Yazılım Maliyet Tahminlemede Yapay Sinir Ağları Kullanılarak Yapılmış Çalışmaların Özeti

Bu konuda az sayıda çalışma mevcuttur. Karşılaştırabilmek için aralarından başarılı olanlar Çizelge 4.4'de verilmektedir.

Çizelge 4.4 Yapay sinir Ağı kullanarak Yazılım Maliyet Tahminlemesi yapma konusunda yapılmış çalışmalar

Çalışma	Kullanılan Algoritma	Ölçüt Kümesi	Proje Sayısı	Sonuç (MMRE)
Wittig & Finnie	Back-Propagation	ASMA ve Desharnais	136 ve 81	17%
Venkatachalm	Back-Propagation	COCOMO	63	
Jorgenson	Back-Propagation	Jorgensen	109	100%
Serluca	Back-Propagation	Mermaid-2	28	76%
Karunanithi	Cascade-Correlation			
Samson	Back-Propagation	COCOMO	63	428%
Srinivasan ve Fisher	Back-Propagation	Kemerer ve COCOMO	78	70%
Hughes	Back-Propagation	Hughes	33	55%

Genellikle geri yayılım yönteminin kullanıldığını görebiliyoruz. En başarılı olan uygulamaların geliştirilen ve görece daha büyük sayıda proje içeren ölçüt kümeleri kullandıkları görülmektedir.

5. YENİ ÖLÇÜT SETİ OLUŞTURMA ÇALIŞMASI

Ölçüt bir sistemin veya bir parçanın verilen bir özelliğinin nicel ölçüm derecesidir. Ölçüt hesaplanabilir veya birleşik bir gösterge olabilir. Bir yazılım ihtiyacının büyüklüğünün ölçülebileceği fikrini ilk ortaya koyan ve bir sistem kuran Allan Albrecht'tir. 1979'da IBM'de çalışan Albrecht işlev puan analizi adıyla bir yöntem ortaya koymuştur. İşlev puanın tanımlanmasından sonra California'da bir danışmanlık firması olan TRW'de bulunan birçok proje üzerinde çalışarak, Barry W. Boehm COCOMO'yu ortaya koymuştur. COCOMO göreceli olarak dosdoğru bir modeldir. Orijinal COCOMO modeli 1981'de yayınlanmıştır. Daha sonra bu geliştirilmiş ve COCOMO II adını almıştır (Samaraweera L.G., 1996).

Yapay sinir ağı kullanarak yazılım maliyet tahminleme konusunda yapılmış olan birçok çalışmada COCOMO 81 ölçüt kümesi kullanılmıştır. Ancak kullanılabilinecek seviyede iyi sonuçlar elde edilememiştir. Burada kritik nokta ölçüt kümesinin oluşturulmasına yeteri kadar önem verilmemesidir. Bu yüzden yapay sinir ağı modelinde yoğunlaşmadan önce bir ölçüt kümesi oluşturmaya karar verilmiştir.

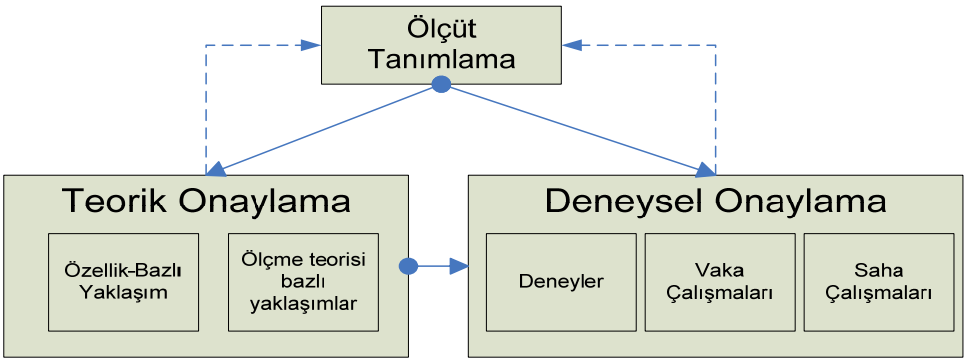
Oluşturduğumuz bu ölçüt kümesine YEEM (Yıldız Effort Estimation Metrics) ismini verdik. Birçok çalışmada COCOMO 81 ölçüt kümesinin kullanılmasından dolayı oluşturduğumuz ölçüt kümesinde COCOMO 81'i içermesi çalışma sonuçlarını bir nebze karşılaştırabilme şansı tanıyacağının önemi ihmal edilmemiştir. Oluşturulan ölçüt kümesi temel olarak COCOMO 81, COCOMO II, 21 adet gerçek yazılım projesini, toplam 90 yıllık yazılım geliştirme tecrübesi olan 28 yazılım geliştirme proje yöneticisi öneri ve fikirleri üzerinde çalışılarak hazırlanmıştır.

Bu çalışmada sadece bir yazılım maliyet tahmini yapay sinir ağı değil aynı zamanda yazılım maliyet tahminlemesinde kullanılabilinecek bir ölçüt kümesi oluşturulmuştur.

5.1 Ölçüt Kümesi Belirleme Metodolojisi

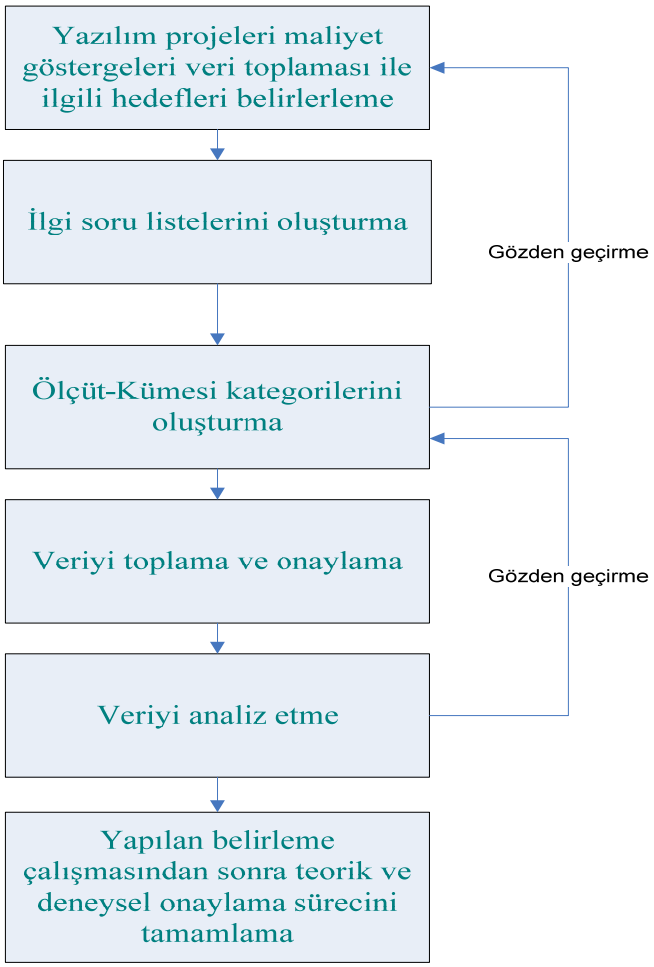
Literatürde ölçüt belirleme konusunda oldukça az çalışma vardır. Yazılım mühendisliği ölçütlerini belirlemek oldukça zordur. İyi tanımlanmamış hedefler, iyi tanımlanmamış hipotezler ve gözlemsel onaylama süreçleri eksikliği temel nedenleridir.

Ölçüt belirleme çalışması metodolojik olarak tanımlanmalıdır. Şekil 5.1'de kullandığımız ölçüt belirleme çalışmasının temel görevleri gösterilmiştir.



Şekil.5.1 Yazılım ölçütlerinin tanımlama ve onaylama metodu

Net ölçüm hedefleri üzerinde bir ölçüt kümesi oluşturulmaya çalışılmıştır. Bu yüzden hedefe yönelik ölçüm (*GQM (Goals/Questions/Metrics) : Goal Oriented Measure*) yaklaşımı ölçüt seçme amacıyla kullanılmıştır. Basili ve Weiss veri toplama metodolojisi YEEM'i oluşturmak için kullanılmıştır. Bu metodolojinin geri beslemeli 6 adımı vardır. Bu altı adım Şekil 5.2'de gösterilmiştir.



Şekil 5.2 Ölçüt tanımlama adımları

Bu ölçüt tanımlama adımları gerçekleştirilmiştir. Burada teorik onaylama süreci temelde iki türlüdür:

- Özellik bazlı yaklaşımlar (Weyuker E.J., 1988; Briand L., Morasca S. ve Basili V., 2002)
- Ölçüm teorisi bazlı yaklaşımlar (Whitmire S. ,1997; Reed, R. D. ve Marks, R. J. ,1999; Poels G. ve Dedene G. ,2000)

Bu çalışmada YEEM'in teorik onaylama süreci için "distance framework" modeli kullanılmıştır. Bu model Poels ve Dedene (Poels G. ve Dedene G., 2000) tarafından ortaya konulmuş ölçme teorisi bazlı yazılım ölçüt onaylama için kavramsal bir çatıdır. Bu çatının 5 aktivitesi vardır. Bunlar:

1. Bir ölçüm soyutlaması bulunur. Ölçüt belirlenir.
2. Uzaklık tanımlanır. Girdi ve çıktıya uygun olarak uygun transform fonksiyonu oluşturulur.
3. Ölçüm soyutlamaları arasındaki uzaklıklar nitelendirilir
4. Referans bir soyutlama bulunur
5. Bu özellik için bir ölçme tanımlanır.

Deneysel onaylama süreci bu tür çalışmalar için aslında bir kanıttır. YEEM ölçüt kümesi için saha çalışmalar ve vaka çalışmaları gerçekleştirilmiştir.

Ölçüt belirleme çalışmamızda dikkat edilen temel noktalar aşağıda maddeler halinde verilmiştir.

a. Ölçüt bulma, çıkarma süreci sistematik ve düzenli olmalıdır. Sürekli iyileştirme yapmak gerekir.
b. Süreç, kişi, ürün bazında mümkün olduğunca çok ölçüt bulup daha sonra yapılacak bir analiz ile en yararlıların bulunmasının sağlanması faydalıdır.
c. Bir yazılım mühendisliği ölçütünün ayrık ve yalıtılmış olması avantajlıdır.
d. Ölçütlerin doğruluğu gerçek hayat ile kontrol edilebilmelidir.
e. Ölçütlerin birbirleriyle ilişkili olmaması gerekmektedir. Birbirleri üzerinde etkisi olmamalıdır veya çok az olmalıdır. Bu sürekli izlenmelidir.
f. İstatistiksel ve başka yöntemlerle ölçütlerin hata analizleri yapılmalıdır. Böylece gereksiz hatta zararlı ölçütler temizlenmelidir.
g. Ölçüt bazında mantıklı ve doğru karşılaştırmalar yapabilmek için kişi, süreç ve ürün benzerlik ve farklılıkları iyi bilinmelidir.

h. Ölçütleri toplarken ölçütlere bulaşanların farkında olunulmalıdır. Burada çok ünlü iki "H" vardır (Heisenberg etkisi ve Hawthorne etkisi).

   Hawthorne etkisi bir araştırma esnasında gözlendiklerinin farkında olan deneklerin doğal davranmayışları olarak açıklanabilir. Heisenberg etikisi ise aynı zamanda gözlenen iki durumdan, birinin ne kadar gözlemlenirse öbürünün o kadar belirsizliğinin artacağını öne süren bir teoridir.

i. Ölçütler zararlı olabilir, daha doğrusu ölçütler yanlış kullanılıyor olabilirler. Buna sürekli dikkat edilmelidir.

5.2 Daskalantonakis'in Ölçüt Kümesi

Motorola'dan M.K.Daskalantonakis tarafından (Daskalantonakis, M.K.,1992) yapılan çalışma iyi bir örnek olduğundan inceleniştir. Bu çalışmada Hedef/Soru/Ölçüt (*Goal/Question/Metric*) yöntemi kullanılmıştır. Hedefler tanımlanmış ve buna karşılık sorular oluşturulmuş, en sonundada bu soruların ışığında ölçütler oluşturulmuştur. Hedefler ve ölçüm alanları Motorola yazılım geliştirme kalite politikasında tanımlanmıştır. Bu hedefler:

Hedef 1 : Proje planı geliştir
Hedef 2 : Hataları kontrol altına almayı yükselt
Hedef 3 : Yazılım güvenilirliğini artır
Hedef 4 : Yazılım hata yoğunluğunu düşür
Hedef 5 : Müşteri servisini geliştir
Hedef 6 : Uyuşmazlık maliyetini düşür
Hedef 7 : Yazılım verimliliğini artır

Bu hedeflerin yanısıra ölçüm alanları belirlenmişti. Bunlar :

- Teslim edilen hata sayısı ve büyüklük bazında hata oranı
- Sürecin başından sonuna kadar toplam etkinlilik
- Zaman planına uyum
- Tahminlerin doğruluk oranları
- Açık olan müşteri problemleri sayısı
- Bu problemlerin açık kaldıkları süre
- Uyuşmazlığın maliyeti
- Yazılım güvenilirliği

Bu hedeflerde kullanılan hata, eksiklik, kusur ve arıza kavramları birbirlerine genellikle karışmaktadır. Hata, kusur, eksiklik ve arıza konusunda inceleme çalışmalarında birbirleriyle karıştıkları görülmüştür. Bu kavramların İngilizce karşılıklarını Türkçe'de netleştirmek anlamayı artıracaktır. Bu kavramlar :

1. Hata (*Error*) : Hata tanım olarak istemeyerek ve bilmeyerek yapılan yanlış anlamında kullanılmaktadır. Yazılım mühendisliğinde, hata doğru olamayan işlem veya hesaplama amacıyla kullanılmaktadır. Hata (*Error*) uygulamanın gözden geçirilmesi sırasında bulunan bir problemdir.

2. Kusur (*Fault*) : Kusur tanım olarak beklenen durumu veya özelliği taşımama ve bilerek veya bilmeyerek bir işi gereği gibi yapmama durumunu belirtmek için kullanılmaktadır. Yazılım mühendisliğinde, kusur uygulamanın beklenildiği gibi çalışmama durumunu belirtmek amacıyla kullanılmaktadır.

3. Arıza (*Failure*) : Arıza beklenen başarı kriterinde olmama durumunu belirtmek için kullanılmaktadır.

4. Eksiklik (*Defect*) : Yazılım mühendisliğinde eksiklik beklenen fonksiyonelitenin tam olarak olmaması,sağlanmaması durumunu belirtmek için kullanılmaktadır. Eksiklik (Defect) uygulamanın gözden geçirilmesinden sonra bulunan bir problemdir.

Metodoloji gereğince daha sonra belirlenen her hedef için ölçütleri oluşturacak soru listeleri oluşturuldu.

5.2.1 Proje planı geliştirilmesi

Hedef 1 : Proje planı geliştir

Soru 1.1 : Projenin hesaplanan zaman planının doğruluğu nedir?

Ölçüt 1.1 : Zaman Planı Tahmini Doğruluğu (ZPTH)

$$\text{ZPTH} = \frac{\text{Projenin Sürdüğü Gerçek Zaman}}{\text{Projenin Tahmini Zamanı}} \tag{5.1}$$

Soru 1.2 : Proje için harcanan çaba tahmininin doğruluğu nedir?

Ölçüt 1.2 : Çaba Tahmini Doğruluğu (ÇTD)

$$\text{ÇTD} = \frac{\text{Proje için harcanan çaba}}{\text{Proje için tahmin edilen çaba}} \tag{5.2}$$

Burada çabadan kastedilen proje için gereken iş gücü diğer bir ifadeyle adam x saat'tir.

Dolayısıyla çaba gereken çalışan sayısı ve bu çalışanların ilgili projeye ayırdıkları zamanın çarpılmasından oluşmaktadır.

5.2.2 Hataları kontrol altına alma

Hedef 2 : Hataları kontrol altına almayı yükselt

Soru 2.1 : Önceki sürüme göre şu an ki hata yakalama sürecinin etkinliği nedir?

Ölçüt 2.1 : Toplam Eksiklik (*Defect*) Kontrolü Etkinliği (TEKE)

$$\text{TEKE}=\frac{\text{Sürüm Öncesi Eksiklik Sayısı}}{\text{Sürüm Öncesi Eksiklik Sayısı + Sürüm Sonrası Eksiklik Sayısı}} \quad (5.3)$$

Soru 2.2 : Özel bir yazılım ürününü geliştirmenin her yapısal fazında mevcut hata kontrol altına alma oranı etkinliği nedir?

Ölçüt 2.2 : Faz *i* için Faz hata (*Error*) kontrolü etkinliği (FHKE)

$$\text{FHKEi}=\frac{\text{Faz i'de sürüme hazırlanırken bulunan hata sayısı}}{\text{Faz i'de sürüme hazırlanırken bulunan hata sayısı+Faz}_i\text{'de sürüm sonrası çıkan hata sayısı}} \quad (5.4)$$

5.2.3 Yazılım güvenilirliğini artırma

Hedef 3 : Yazılım güvenilirliğini artır

Soru 3.1 : Yazılım arıza oranı nedir ve zaman içinde nasıl değişimektedir?

Ölçüt 3.1 : Arıza (*Failure*) Oranı (AO)

$$\text{AO} = \frac{\text{Arıza Süresi}}{\text{Çalışma Süresi}} \quad (5.5)$$

5.2.4 Hata yoğunluğunu düşürme

Hedef 4 : Yazılım hata yoğunluğunu düşür

Soru 4.1 : Normalize edilmiş olarak devam eden arıza sayısı ve devam eden hata sayısı ile nasıl karşılaştırabilirsiniz?

Ölçüt 4.1.a : Devam eden kusurlar (*Fault*) (DEK)

$$\text{DEK}=\frac{\text{Artışsal yazılım geliştirmeden kaynaklanan devam eden kusur (Fault) sayısı}}{\text{Kaynak kodun assembly eşitinde olanın değişimi}} \quad (5.6)$$

Kaynak kodun assembly dilinde eşiti olan kod miktarı farklı kişilerin kodlamalarında farklı çıkabilmektedir. Burada bir insan yerine belirli standartları barındıran ve tüm uygulama süresince aynen kullanılacak olan bir araç kullanılarak elde edilen assembly kodu kullanılabilinir.

Ölçüt 4.1.b : Devam eden eksiklikler (*Defect*) (DEE)

$$\text{DEE}=\frac{\text{Artışsal yazılım geliştirmeden kaynaklanan devam eden eksiklik (Defect) sayısı}}{\text{Kaynak kodun assembly eşitinde olanın değişimi}} \quad (5.7)$$

Soru 4.2 : Sürüme verilen yazılımda bilinen bozukluk sayısının kodun assembly eşitine oranı nedir?

Ölçüt 4.2.a : Sürüme Verilen Toplam eksiklikler (*Defect*) (SVE)

$$\text{SVE}_{(\text{Toplam})}=\frac{\text{Sürüme verilen eksiklik (Defect) sayısı}}{\text{Kaynak kodun assembly eşitinde olanın değişimi}} \quad (5.8)$$

Ölçüt 4.2.b : Sürüme Verilen Toplam eksiklik (*Defect*) Değişimi (SVE)

$$\text{SVE}_{(\text{Değişim})}=\frac{\text{Artışsal yazılım geliştirmeden kaynaklanan ve sürüme verilen eksiklik (Defect) sayısı}}{\text{Kaynak kodun assembly eşitinde olanın değişimi}} \quad (5.9)$$

Soru 4.3 : Müşteriye verilen sürümde müşterinin bulduğu eksiklik (*Defect*) ne kadardır?

Ölçüt 4.3.a : Müşterinin Bulduğu eksiklik (*Defect*) (MBE)

$$\text{MBE}_{(\text{Toplam})}=\frac{\text{Müşterinin bulduğu eksiklik (defect) sayısı}}{\text{Kodun asssembly eşitinin büyüklüğü}} \quad (5.10)$$

Ölçüt 4.3.b : Müşterinin Bulduğu Eksiklik (*Defect*) Değişimi (MBE)

$$\text{MBE}_{(\text{Değişim})}=\frac{\text{Artışsal yazılım geliştirmeden kaynaklanan ve müşterinin bulduğu eksiklik (defect) sayısı}}{\text{Kodun asssembly eşitinin büyüklüğü}} \quad (5.11)$$

5.2.5 Müşteri servisini geliştirme

Hedef 5 : Müşteri servisini geliştir

Soru 5.1 : Bu ay açılan yeni problem sayısı nedir?

Ölçüt 5.1 : Yeni açılan Problem (YAP)

YAP = Bu ay açılan sürüm öncesi problemleri (5.12)

Soru 5.2 : Bu ay sonu itibariyle açık problemlerin sayısı nedir?

Ölçüt 5.2 : Açık Problem Sayısı (APS) (5.13)

APS = Bu ay sonu itibariyle çözülmemiş olan sürüm öncesi problemlerin sayısı

Soru 5.3 : Bu ay sonu itibariyle açık olan problemlerin ortalama bekleme süresi nedir?

Ölçüt 5.3 : Açık Problemlerin Ortalama Bekleme Süresi (APOBS)

$$\text{APOBS} = \frac{\text{Bu ay sonu itibariyle sürüm öncesi açık problemlerin toplam bekleme süresi}}{\text{Bu ay sonu itibariyle sürüm öncesi açık problemlerin toplam sayısı}} \quad (5.14)$$

Soru 5.4 : Bu ay kapanan problemlerin ortalama bekleme zamanı ne idi?

Ölçüt 5.4 : Kapanan Problemlerin Ortalama Bekleme Süresi (KPOBS)

$$\text{KPOBS} = \frac{\text{Bu ay sürüm öncesi kapanan problemlerin toplam bekleme süresi}}{\text{Bu ay sürüm öncesi kapanan problemlerin toplam sayısı}} \quad (5.15)$$

5.2.6 Uyuşmazlık maliyetini düşürme

Hedef 6 : Uyuşmazlık maliyetini düşür

Soru 6 : Bu ay için sürümde olan yazılımın onarım maliyeti nedir?

Ölçüt 6 : Bu ay için harcanan Yazılım Onarım Maliyeti

5.2.7 Yazılım verimliliğini artırma

Hedef 7 : Yazılım verimliliğini artır

Soru 7.1 : Yazılım geliştirme projesinin verimliliği kaynak kod büyüklüğü bazında ne idi?

Ölçüt 7.1.a : Verimlilik

$$V_{(\text{Toplam})} = \frac{\text{Toplam kaynak kodunun assembly eşitinin büyüklüğü}}{\text{Yazılım Geliştirme Çabası}} \quad (5.16)$$

Ölçüt 7.1.b : Verimlilik Değişimi

$$V_{(\text{Değişim})} = \frac{\text{Toplam kaynak kodunun assembly eşitinin büyüklüğü değişimi}}{\text{Yazılım Geliştirme Çabası}} \quad (5.17)$$

5.3 Ölçüm Verisinde Gürültü Kaynakları

Bir grubun bir üyesinin belirli bir özelliğini nicelendirdiğimizde, bu özelliğin gerçek değerinden emin olunamaktadır (Munson J.C., 2003). Sadece doğa bu özelliğin gerçek değerini bilmektedir. Burada ölçüm yapanın bu durumda yapabileceği, bu değer için sağlayabileceği ancak bir kestirim olabilmektedir. Doğa, özelliklerin gerçek değerlerinin yanına gürültü eklemektedir. Örneğin diyelim ki bir kişinin boyunun uzunluğu ölçmeye çalışalım. Bu görece basit bir problemdir. Üzerinde çalıştıkça problem boyutu artmaktadır. Öncelikle bu kişi yaşayan bir organizmanıdır. Bu kişinin fiziksel durumu sürekli değişmektedir. Ölçmeyi yaptığımız zaman sadece o an ki durum için bir değer elde etmiş oluruz. Kişinin boyu ölçüm zamanındaki duruşuna, omurgasının anlık durumuna, moral durumuna, omuzlarının eğilme durumuna bağlı olarak değişebilmektedir. Bu durumda yapabilinecek, ne kadar kesinlikte bir sonuç ihtiyacımızın olduğunu öncelikle ortaya

koymaktır. Doğanın bizim ölçüm değerinin üzeine eklediği veya çıkardığı değere gürültü denmektedir. Bazen gürültü miktarı çok yüksek ölçümler sorun olamayacağı gibi bazı durumlarda olabilecek en küçük gürültülü ölçüme ihtiyacımız olabilmektedir. Diyelim ki bir yazılım geliştirme uzmanının belirlenmiş bir uygulama parçasını ne kadar zamanda tamamlayacağı bilgisine ihtiyacımız var olsun. Bir çok organizasyon bu işe başlama zamanını ile bitirme zamanı arasında geçen zamanı bularak bunu ölçmeye çalışmaktadırlar. Ancak aslında yazılım geliştirme uzmanı bu iki zaman arasında sadece yazılım geliştirme yapmamış, bunun yanında ilintisiz veya ilintli işler de yapmış olabilmektedir. Bir örnek olarak bir yazılım geliştirme uzmanı zamanını şu şekilde kullanmış olabilir:

- Zamanın %40'ını internette sörf yaparak
- Zamanın %10'unu e-postalarını okuyarak ve cevap yazarak
- Zamanın %20'sini arkadaşlarıyla konu dışı iletişim kurarak
- Zamanın %5'inde dinlenerek
- Zamanın %2'sinde arkadaşlarıyla konu dışı bir konuyu tartışarak
- Zamanın %5'inde çevresindekilerle konuşarak
- Zamanın %5'inde gözden geçirme toplantısıyla
- Zamanın %2'sinde idari toplantılarla
- Zamanın %11'inde yazılım geliştirme

şeklinde değerlendirmiş olabilir (Munson J.C., 2003). Dolayısıyla bu durumda yazılım geliştirme uzmanının verimliliği de önemli bir parametre haline gelmiş oluyor. Benzer durumlar düşünüldüğünde bunun yanında yazılım geliştirme konusunda becerisi, baskı miktarı, sahiplenme gibi ölçütlerde önemli hale gelmiş olmaktadır. Diğer bir deyişle tek tek ölçütler yanıltıcı olabilmekte ancak dengeleyici ölçüt kümeleri kullanarak daha doğru sonuçlar elde etmek mümkün olabilmektedir.

5.4 Ölçüt Seti oluşturma

Yazılım mühendisliğinde ölçütleri 3 ana grupta toplayabiliriz:

1. Yazılım mühendisliği ürünü: Örneğin tasarım, kaynak kod, test senaryoları
2. Yazılım mühendisliği süreci: Örneğin analiz, tasarım, kodlama ve test aktiviteleri
3. Yazılım mühendisliği çalışanları: Örneğin her bir yazılım geliştiricinin, analistin, test çalışanının verimliliği

Bu 3 grup bu çalışmada ana gruplar olarak belirlenmiştir. Ölçüt belirleme çalışmaların bir çoğunda bu 3 ana grup içerilmiştir. Ana gruplar birbirlerine oldukça yakındır. Önemli bir

kümeleme çalışmalarında da bu 3 gruba oldukça yakın bir şekilde 4 ana grupta toplanmıştır (Munson, J.C., 2003):

1. Çalışanlar
2. Yazılım Geliştirme Süreci
3. Ürün
4. Ortam

Bu 4 ana gruplu ölçüt kümesinde (Munson C.J., 2003) çalışanlar için yaş, cinsiyet, yazılım dilindeki tecrübe, şu an görev aldığı kurumdaki tecrübesi gibi; süreçle ilgili olarak yazılım geliştirme uzmanı hatası oranı, yazılımda değişilik oranı, maliyet bilgisi, yatırımın geri dönüşümü gibi; ürünle ilgili olarak yazılım gereklilik spesifikasyonu, yüksek ve alt seviye tasarımı, kaynak kodu, test durumları ve dokumantasyon gibi; ortamla ilgili olarak işletim sistemi, geliştirme ortamı, idari değişim, makinelerin düzgün çalışabilirliği, ofiste yaşanan kesilmeler, kütüphane olanakları gibi ölçütler mevcuttur.

Bu 3 veya 4 ana grup birçok çalışmada ve yayında mevcuttur. Bu tez çalışmasında Hedef/Soru/Ölçüt metodolojisiyle bu görüş desteklenmiştir. Ancak 3 grup daha eklenerek 6 grup oluşturulmuştur.

Bu çalışmada genel bir ölçüt kümesi çıkarması hedeflenmiştir. Nesneye dayalı yazılım mühendisliği ölçütlerini de inceleyerek genel bir yapı kurulmuştur. Nesneye dayalı yazılım mühendisliği ölçütlerinde genellikle kullanılan ölçütler (Chidamber S.R ve Kemerer CF , 1994) :

- Nesne başına düşen metot sayısı
- Kalıtım ağacı derinliği
- Çocuk nesne sayısı
- Nesneler arasındaki bağlaşım

Aslında nesneye yönelik ölçütlerde esas olarak ürünün ne kadar büyük ve ne kadar karmaşık olduğu anlaşılmaya çalışılmıştır. Nesneye yönelik olmayan ortamlara kıyasla ürünün büyüklüğünü tanımlamada daha başarılı olabilecek ölçütler yapı itibariyle mevcuttur.

Yazılım maliyet tahminleme çalışmalarında çoğunlukla COCOMO 81 ölçüt kümesi kullanılmıştır. Bu yüzden oluşturacağımız ölçüt kümesi mümkün olduğunca COCOMO 81'i içermesine çalıştık. Ölçüt belirleme çalışması yaparken 21 adet gerçek proje, COCOMO 81, yapılmış çalışmalar ve 28 tane 3 yıldan fazla proje yönetim tecrübesi olmuş proje yöneticisiyle birebir görüşüldü. Bu görüşmeler oldukça zaman almıştır. Önceden randevu

alınarak gerçekleştirilmiş bu formal görüşmeler 5 adımda gerçekleşmiştir (Ayyıldız M., Kalıpsız O, 2005).

1. Bu çalışmanın yapılmasının amacı ve beklenen fayda iletilmiştir.
2. Proje yöneticilerinden temel grupları belirlemelerini istedik. Burada açık uçlu olarak belirlemelerini istedik. Bir liste getirip seçmeleri bir sınırlandırma yaratacağı için kullanmadık.
3. Proje yöneticilerinden belirledikleri grupların yanı sıra bu grupların ağırlıklarını da belirtmelerini istedik. Ellerinde bulundan %100'lük bir pastayı bu gruplar arasında paylaştırmalarını istedik. Proje yöneticileri bunları oluşturduktan sonra biz bu grupların isimleri aynı olanları veya anlamları aynı olup farklı isimlendirilenleri aynı gruba dahil edip ortalamalarını aldık. Bu sayede çizelge 5.1 elde edilmiştir.
4. Oluşan sonuç verileri üzerinde mutabık kalınılmıştır.
5. Projenin ara adımlarda ve birleştirilmelerde bilgi verileceği iletilerek görüşme sonlandırılmıştır.

Net olmayan durumlar için iki tur olarak görüşmeler tekrarlanmıştır.

Çizelge 5.1 Ana Grup Sonuçları

Ana Grup	Önem Ortalaması
Yazılım geliştirme ortamı	11,10
Kullanılan teknoloji	10,12
Risk	9,97
Ayrılan kaynak	9,78
İşin Büyüklüğü / Ürünün büyüklüğü	8,15
Planlar	7,08
Ürünün karmaşıklığı	6,66
Tahminler	6,55
Bütçe	5,78
Çalışanların yetkinlikleri	4,34
Fiziksel koşullar	3,44
Proje kapsamında değişiklik oranları	3,32
Tecrübe	3,31
Proje planında değişiklik oranları	3,13
Diğer	2,62

Takımdaşlık	2,54
Motivasyon	2,11

Çizelge 5.1'de adı geçen etkenleri kısaca açıklamakta fayda var. Buna göre :

- Yazılım geliştirme ortamı : Yazılım geliştirme ortamındaki ekosistem, etken faktörler, yazılım geliştirme atmosferi kastedilmektedir.
- Kullanılan teknoloji : Kullanılan yazılım geliştirme araçları, dili, metodolojileri kastedilmektedir.
- Risk : İlgili yazılım geliştirme projesinde oluşan riskler kastedilmektedir.
- Ayrılan kaynak : İlgili yazılıma ayrılan iş gücü kastedilmektedir.
- İşin Büyüklüğü / Ürünün büyüklüğü : İlgili yazılım geliştirme projesinin sonucunda oluşacak yazılımın hacmi kastedilmektedir.
- Planlar : İlgili yazılım projesinin proje planı kastedilmektedir.
- Ürünün karmaşıklığı : Geliştirilecek olan yazılımın karmaşıklığını ifade etmektedir.
- Tahminler : İlgili yazılım projesinin proje planının proje yöneticisi tarafında gerçekçi tahmini kastedilmektedir.
- Bütçe : İlgili yazılım geliştirme projesi için ayrılan bütçe miktarıdır.
- Çalışanların yetkinlikleri : İlgili yazılım geliştirme projesine atanan, çalışacak olan personelin projedeki görevini yapabilme yetkinliği, muktedirliğidir.
- Fiziksel koşullar : Fiziksel ortam koşulları kastedilmektedir. Ortam sıcaklığı, araç gereç durumu, gürültü, ışık oranı, ulaşım durumu, imkanlar bu fiziksel koşullara birkaç örnek olarak verilebilinir.
- Proje kapsamında değişiklik oranları : İlgili projede proje ilerlerken kapsamda meydana gelebilecek kapsam değişiklikleri oranıdır.
- Tecrübe : İlgili yazılım geliştirme projesine atanan, çalışacak olan personelin projedeki görevine benzer işlerde daha önceki tecrübe oranlarıdır. Tecrübe çalışan yetkinliği ile kısmen ilintilidir ancak tecrübeli olmasına rağmen aynı oranda yetkin

olamama veya tecrübesiz olmasına rağmen aynı oranda yetkin olma durumları olabilir. Bu yüzden ayrı ayrı maddeler olarak geçmektedir.

- Proje planında değişiklik oranları : Proje zaman planında, projede alt iş parçacıklarının atanan kişilerde meydana gelebilecek değişiklikler.
- Takımdaşlık : İlgili projede görev alan çalışanların oluşturduğu takımda bulunan takımdaşlık durumu, beraber çalışabilirlik ve oluşan sinerji durumu kastedilmektedir.
- Motivasyon : Projede görev olan çalışanların motivasyon durumu, takımın genel motivasyon durumu kastedilmektedir.

Çizelge 5.1'de görülen durumun %95'ini oluşturacak 6 temel grupta toparlayabileceğimizi görülmüştür. Buna göre

- Ürünün karmaşıklığını ürün içinde değerlendirebileceğimize
- Bütçe'yi ürünün içinde değerlendirebileceğimize
- Ayrılan kaynağı ve tecrübeyi ve çalışanın yetkinliklerini kaynak başlığı altında toplayabileceğimizi
- Değişikliklerle ilgili kısımların risk konusunda ele alınabileceğini
- Planlar ve tahminlerin tek başlık altında toplanabileceği gözlemlemiş olduk.

Ana gruplarımızın işin büyüklüğü (ürün), kaynak, risk, teknoloji, ortam ile planlar ve tahmin grupları olabileceğini bu çalışmada gözlemledik.

COCOMO'da belirtilen tüm ölçütler bu önerilen ölçüt kümesinde kullanılmıştır. Bununla bir yenilik olarak bir hiyerarşik yapı getirilmiştir. Var olan tahminleme yöntemlerinden de gördüğümüz gibi melez bir çözüm başarımı artıracaktır. Bu yüzden uzman görüşünü de bu modele dahil etmek istedik. Bu çalışma aynı zamanda daha iyi sonuç alabilmek için bir melez model önermektedir. Birçok ölçütün yanı sıra işlev puanı kullanımı ve baştan yapılan tahminlerinde girdi olarak kullanımı melez bir durum yaratmaktadır ve başarı ihtimalini artırmaktadır. Dolayısıyla COCOMO'da kullanılmayan bazı ölçütlerde eklenmiştir.

5.5 YEEM Ölçüt Seti

Elde ettiğimiz ölçüt kümesi Çizelge 5.2'de gösterilmiştir. 6 ana ölçüt grubu ve bunlara bağlı 24 ölçütü bu tablodan görebilirsiniz. Bu 24 tane ölçütün bazılarının da alt ölçütleri vardır. Bunlarla beraber aslında toplam 56 tane ölçüt mevcuttur. Bu alt ölçütleri bu tabloda gösterip çizelgeyi karmaşık hale getirmemek için daha sonra açıklamaya çalıştık.

Buna göre oluşturduğumuz ölçütler kümesinin 6 temel grubu şu şekildedir (Ayyıldız M., Kalıpsız O, 2005):

A. İşin Büyüklüğü (Ürün)

B. Kaynak

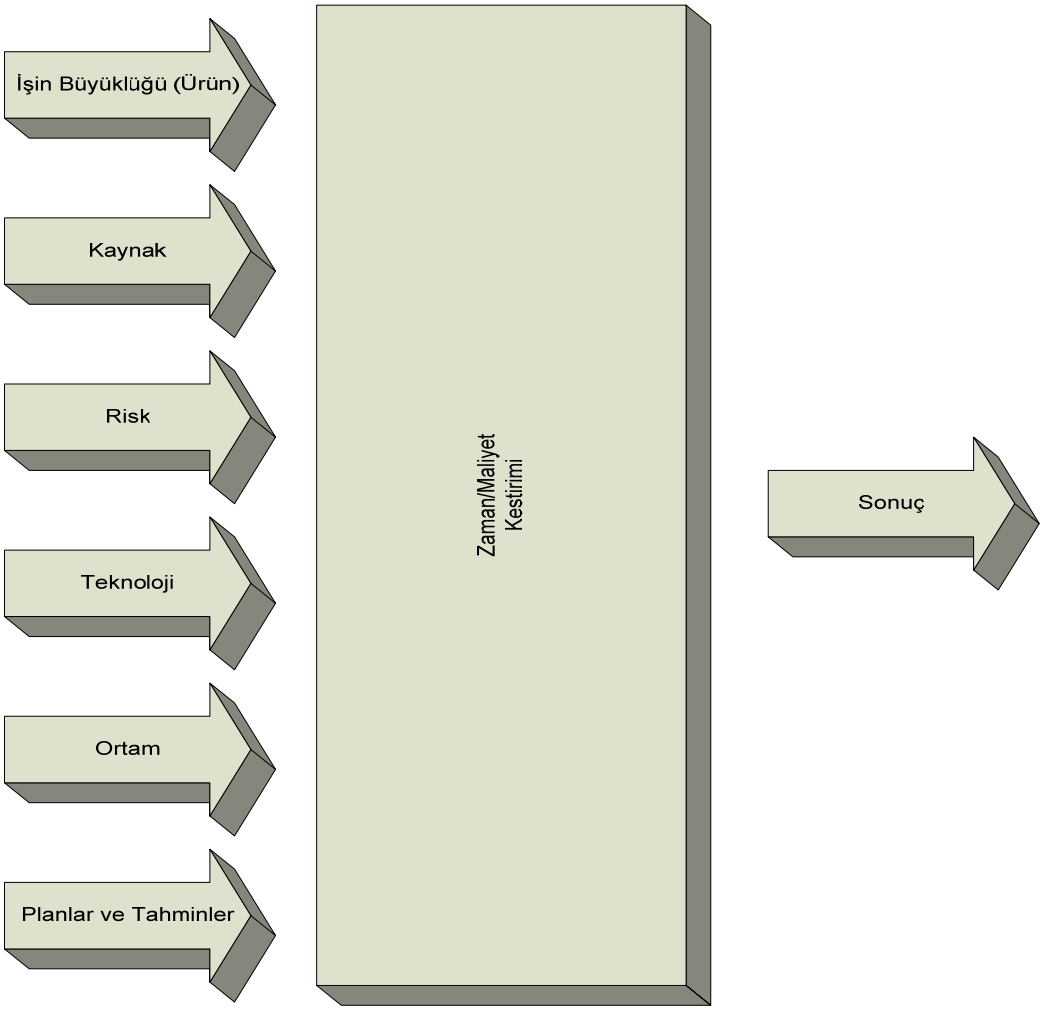
C. Risk

D. Teknoloji

E. Ortam

F. Planlar ve Tahminler

Amacımız bu 6 ana grubu girdi olarak kullanıp (Şekil 5.3) sonuca ulaşmaktır.



Şekil 5.3 Ana gruplar

Çizelge 5.2 YEEM Ölçüt Seti

Gruplar	Ölçüt
İşin Büyüklüğü (Ürün)	1. Karmaşıklık
	2.İşlev puan (*Function Point*)
	3.Önem
	4. Ayrılan bütçe
	5. Ürünün beklenen özellikleri
Kaynak	6. Çalışanın yeterlilikleri
	7 Çalışanların projeye katılım oranları
	8. Çalışan kişi sayıları
	9. Donanım durumu
Risk	10.Bütçe değişme riski
	11. Çalışan riski
	12.Donanım riski
	13.Ürünün tanımının ve kapsamının değişme riski
Teknoloji	14.Yazılım geliştirme araçlarının kullanım kolaylığı
	15.Yazılım geliştirme araçlarının kullanım tecrübesi
	16. Yazılım geliştirme araçlarının Kullanımı
	17. Modern Programlama Teknikleri
Ortam	18.Ortamın genel özellikleri
	19.Sahiplenilme (Her bir paydaş türünün projeyi sahiplenmesi)
	20.Baskı
	21.Zaman kullanım durumu
	22.Verimlilik durumu
Planlar ve Tahminler	23. Tahmin
	24. Planlar

5.5.1 İşin Büyüklüğü

6 temel gruptan biri olan işin büyüklüğünü bulabilmek için işlev puanı kullanmanın yanı sıra ürünün karmaşıklığı, önemi, beklenen özellikleri almak büyüklük temel ölçütünün daha doğru olmasını sağlayacaktır. Karmaşıklığın 3 alt ölçütü olarak Veritabanı büyüklüğü, Ürün karmaşıklığı ve yeniden kullanılabilinirlik olarak belirledik. Bunların üçü de COCOMO'da DATA, CPLX ve RUSE olarak geçmektedir.

İşlev puanı bir yazılım uygulamasının büyüklüğünün ölçütüdür. Bu büyüklük işlevsellik ölçümüdür. Yazılım dilinden, metodolojiden, proje takımının yeterliliğinden bağımsızdır. İşlev puanın nerede zorlandığını bilmek önemlidir. İşlev puanı bir uygulama için gereken çabayı ölçmek için mükemmel değildir. Ancak işin büyüklüğünü belirleme anlamında önemlidir. Çoğunlukla gerçek hayatta işin büyüklüğünü anlamak için benzetim kullanılır.

Örneğin 1000 m^2 bir ev yapmak genellikle 2000 m^2 bir ev yapmaktan daha ucuza mal olmaktadır. Ancak mermer banyo, karo fayans gibi birçok parametrenin aslında durumu tamamen değiştirmesi mümkündür. Dolayısıyla 1000 m^2 bir evin yapımının 2000 m^2 bir ev yapmaktan daha ucuza mal olabileceği gibi daha pahalıya mal olma ihtimalide vardır. Dolayısıyla sadece evin büyük olması çaba ile ilgili bir fikir verse bile bunun üzerine maliyetinin hesaplanması pek mümkün gözükmemektedir. İşlev puan'da bunun gibi ürünün büyüklüğünü vermesine karşın tek başına maliyetin bulunması mümkün değildir. İşlev puanı kullanmanın iki avantajı vardır. Bize ürünün büyüklüğünü ve ürünün görünen büyüklüğünü verir. Bunu girdi olarak kullanmak faydalı olacaktır. Dolayısıyla önereceğimiz ölçüt kümesine işlev puanı da bir ölçüt olarak eklenmiştir.

Bir projeye ayrılan bütçe o projenin temel parçalarından biridir. Dolayısıyla projenin büyüklüğü ve önemi ile ilgili genel bir fikir verir. Bu açıdan ayrılan bütçe büyüklüğünü bir ölçüt olarak kullanmak faydalı olmaktadır. Bununla beraber nihai ürünün önemi de doğrudan gereken çabayı etkileyecektir. Bu bize toplam baskıyı vermenin yanı sıra o projeye atanacak çalışanların yetkinlikleri ile ilgili bir bilgi de vermiş olacaktır. Hatta yönetimsel destek ile ilgili önemli bir parametre olması dolaysıyla projenin planlanan zamanda bitmesini etkileyecektir.

Ürünün beklenen özellikleri ölçütünün, alt ölçütlerini de belirledik. Bunlar esneklik, güvenilirlik, kullanım kolaylığı, bakım yapılabilirlik, güvenlik ve dokümantasyon'dur.

5.5.2 Kaynak

Kaynağın büyüklüğü ve niteliği doğrudan olarak sonucu etkilemektedir. Basit olarak bir işin maliyetini bulmak istersek ürünün büyüklüğünü ve dolayısıyla gereken kaynağın büyüklüğünün ne olduğunu bilmeye ihtiyacımız vardır.

Çalışanın yetkinliklerini ise gruplara göre alt ölçütlerini oluşturduk. Bunlar proje yöneticisin yetkinliği, yazılım geliştiricilerin yetkinliği, sistem analistlerin yetkinliği ve test edenlerin yetkinlikleridir.

Çalışan kişi sayılarını gruplara göre alt ölçütlerde kapsamak faydalı olacaktır. Yazılım geliştirici sayısı, analist sayısı, test yapanların sayısı ve genelde bir olmasına rağmen olası durumlar için proje yöneticisi sayısı alt ölçütlerdir.

Donanım durumu ölçütünün de alt ölçütleri vardır. Bunlar zaman kısıtı, depolama kısıtı, platform değişkenliğidir. Bunların üçü de COCOMO'da mevcuttur ve adları TIME, STOR ve

PVOL'dur.

5.5.3 Risk

Risk, projenin zamanında ve eksiksiz bir şekilde tamamlanmasına engel olabilecek her türlü etmendir. Projede ileride ortaya çıkabilecek problemlerin önceden belirlenmesi, erken önlem alınması önemli bir konudur. Herhangi bir projede ilk tanımlanması sırasında, karşılaşılması olası risklerin belirlenmesi gereklidir. Bu yüzden bir risk yönetimi yürütülerek risklerin belirlenmesi gerekir. Riskler, nasıl giderilebileceği, alınacak önlemler ve etkiler öngörülmeli ve planlanmalıdır.

Risk etkisi riskin oluşma olasılığı ve bunun maliyetinin çarpımı olarak tanımlanabilir. Dolayısıyla ürün ve kaynak risklerini ve etkilerine ihtiyacımız vardır ve her birinin alt ölçütlerini belirledik. Her birinin iki alt ölçütü vardır. Bunlar olma olasılığı ve olunca yaratacağı etkidir.

5.5.4 Teknoloji

Kullanılan yazılım geliştirme araçları ve bu araçlarda ki tecrübe oldukça önemlidir. Sonuçta oluşacak maliyete önemli oranda etkilemektedir. Yazılım geliştirme araçlarının kullanım kolaylığı yanında yer alan yazılım geliştirme araçlarındaki tecrübenin de alt ölçütünü belirledik. Bunlar platform tecrübesi, uygulama tecrübesi, yazılım dili ve çoklu ortam geliştirme durumu tecrübesidir. Bunlar COCOMO'da PEXP, AEXP ve LTEX, SITE olarak geçmektedir.

5.5.5 Ortam

Yazılım geliştirilen kurumun yapısı, maliyetleri doğrudan etkilemektedir. Bu kurumdaki ortalama gecikme, ortalama sapma, baskı, sahiplenme ve sorumluluk gibi durumlar sonucu doğrudan etkilemektedir. Ortamın genel özellikleri ölçütünün alt ölçütlerini geciken proje oranı, kurumdaki ortalama gecikme oranı, ortalama gecikme miktarı, gecikmenin standart sapma miktarı oluşturmaktadır.

Sahiplenme çok önemli bir ölçüttür. Bunun alt ölçütleri proje yöneticisinin, yönetimin, müşterinin, proje çalışanlarının sahiplenme durumları alt ölçütlerdir.

Baskı miktarı ölçütü projenin gecikmesine ve proje maliyetine etki eden önemli bir ölçüttür. Bunun 3 alt ölçütü vardır. Bunlar pazar baskısı, müşteri baskısı, yönetim baskısıdır.

Zaman yönetimin alt ölçütleri bir çalışanın ortalama gün içinde kesilme (*interrupt)* sayısı, bir kesilmenin ortalama süresi, kesilme sonrası önceki duruma geçme ortalama süresi, bu kurumda ortalama yapılan fazla mesai süresidir.

5.5.6 Planlar ve Tahminler

Uzman görüşü çoğu yerde en çok kullanılan yazılım maliyeti tahmin etme yöntemidir. Bu yöntem maalesef mevcut yöntemler arasında hala en başarılı olandır. Bu görüşü de bir ölçüt olarak almak faydalı olacaktır. Bu uzmanların tahmin sapma oranlarını da bir ölçüt olarak aldığımız takdirde daha doğru sonuçlar alabileceğiz. Tahmin ölçütünün alt ölçütleri tahmini zaman planı, Tahmini yapan kişi veya kişilerin otalama sapma oranıdır. Plan ölçütünün alt ölçütü planlanan zaman, yanılma oranı ve hedeflenen zamanda bitmemesi durumunda katlanılabilinirlik şeklindedir.

5.6 Gerçek projelerden veri toplama

Veri toplaması oldukça zaman alan ve dikkat gerektiren bir süreçtir. Birçok kurum rekabet güçlerini olumsuz etkileyebileceği endişesiyle ellerindeki ölçütleri paylaşmak istememektedirler. Bu yüzden veri sayısını artırmak oldukça zordur.

Türkiye'de bulunan lider bir uluslararası kurumdan birçok konuda yapılmış 109 proje için bu veriler toplandı. Veri toplamak için çeşitli yöntemler vardır. Mevcut altyapıdan çekilebilinecek veriler olmasının yanı sıra, çalışanların doldurduğu formlar ve doğrudan görüşmelerle alınabilinecek bilgiler vardır.

Çizelge 5.3 Veri toplama formu

Proje No	
Proje Adı	
Proje Yöneticisi	
Proje Sponsoru	
Proje Müşterisi	
Geliştiren Bölüm	
Müşteri	
Amaç	
Kurum Adı	
Proje Başlangıç Tarihi	
Proje Kapanış Tarihi	
Harcanan adam x saat	

Ana Ölçütler	Ölçütler	Alt Ölçütler	Değer	Açıklama
A. İşin Büyüklüğü (Ürün)	1.Karmaşıklık	a.Veritabanı büyüklüğü [DATA]		
		b.Modüllerin karmaşıklığı [CPLX]		
		c.Yeniden kullanılabilinirlik [RUSE]		
	2.Function Point	2.Function Point		
	3.Önemi	3.Önemi		
	4.Ayrılan Bütçe	4.Ayrılan Bütçe		
	5.Ürün beklenen özellikleri	a.Esneklik		
		b.Doğruluk		
		c.Kolay kullanım		
		d.Dokumantasyon		
		e.Bakım Yapılabilirlik		
		f.Güvenilirlik		
B. Kaynak	1.Çalışanların yeterlilikleri	a.Proje yöneticisinin Yeterlilikleri		
		b.Yazılım Geliştiricilerin Yeterlilikleri		
		c.Analistlerin Yeterlilikleri		
	2.Çalışanların projeye katılım oranları [PCON]	2.Çalışanların projeye katılım oranları [PCON]		
	3.Çalışan kişi sayıları	a.Çalışan yönetici sayısı		
		b.Çalışan yazılım geliştirici sayısı		
		c.Çalışan analist sayısı		
		d.Çalışan test eden sayısı		
	4.Donanım durumu	a.Zaman Kısıtı [TIME]		
		b.Ana bellek kısıtı [STOR]		
		c.Bilgisayar platform değişim olasılığı [VIRT]		
		d.Bilgisayar iş geri dönüş zamanı [TURN]		
C. Risk	1.Kaynak riskleri ve	a.Bütçe Riski Gerçekleşme Olasılığı		
		b.Bütçe Riski Etkisi		

	gerçekleşme oranı	c.Çalışan Riski Gerçekleşme olasılığı		
		d.Çalışan Riski Etkisi		
		e.Donanım Riski Gerçekleşme olasılığı		
		f.Donanım Riski Etkisi		
		g.Ürün tanımının veya kapsamın değişiklik gerçekleşme olasılığı		
		h.Ürün tanımının veya kapsamın değişiklik etkisi		
D. Teknoloji	1.Yazılım Geliştirme Araçlarının kolaylığı	1.Yazılım Geliştirme Araçlarının kolaylığı		
	2.Yazılım Geliştirme Araçlarının Developer'larca bilinirliği	2.Yazılım Geliştirme Araçlarının Developer'larca bilinirliği		
	3.Yazılım Geliştirme araçları kullanımı *(Proje ekibi tarafından bilgisayar destekli yazılım geliştirme araçlarının kullanım oranı)*	3.Yazılım Geliştirme araçları kullanımı (Proje ekibi tarafından bilgisayar destekli yazılım geliştirme araçlarının kullanım oranı)		
	4.Modern Programlama Teknikleri [MODP]	4.Modern Programlama Teknikleri [MODP]		
E. Ortam	1.Ortamın genel özellikleri (Ivme)	a.Yazılımın gerçekleştiği firmanın projelerin gecikenlerin oranı		
		b.Ortalama gecikme oranı (Yazılımın gerçekleştiği firmanın proje uzunluğu baz alınarak yüzde olarak ortalama gecikme miktarı)		
	2..Sahiplenilme	a. Yönetimin projeye sahiplenme oranı		
		b. Proje yöneticisinin projeye sahiplenme oranı		
		c. Analistlerin projeye sahiplenme oranı		
		d. Developerların projeye sahiplenme oranı		
		e. Müşterilerin projeye sahiplenme oranı		
	3.Baskı	a.Üstten/Yönetimden gelen baskı		
		b.Müşteriden gelen baskı		
		c.Pazardan gelen baskı		
	4.Zaman kullanım durumu (1 gundeki gunluk kayıp)	a.Çalışanların günde ortalama kaç defa kesildiği (adet)		
		b.Ortalama kesilme süresi (dk)		
		c.Kesilmeden sonra normala dönüş ortama süresi (dk)		
		d.Ortalama yapılan fazla mesai saati (saat)		
	5.Verimlilik durumu	5.Verimlilik durumu		
F. Planlar ve Tahminler	Planlar	a Hedeflenen/Planlanan Zaman (Kaç Gün)		

		b.Hedeflenen zaman planı yanılma oranı (% kac)		
		c. Hedeflenen zamanda bitmemesi durumunda katlanılabilinirlik - Tolerans (%)		
	Tahminler	a.Gerçekçi Tahmini zaman		
		b.Tahmini yapan kişi veya kişilerin tahmin tutturma oranı		
Açıklama				

Temel olarak elimizde proje planları, bütçe, kaynaklar, katılım oranlarını temel olarak barındıran bir veri kümesive bu veri kümesinden bir takım ölçütleri alabilmek mümkündü. Ancak alınamayacak bilgiler için formlara ihtiyaç duyulmaktadır. Birçok veri toplama formu mevcuttur. Bu formlar daha çok elektronik ortamdadır. Bunlara bir takım eklemeler yapılıp YEEM ölçüt kümesini kapsayacak uygun bir form haline getirildi (Çizelge 5.3). Yeni gelen projelerde bu sayede bu ölçütler toplanmaya başlandı. Ancak çoğunluğu oluşturan tamamlanmış, eski projelerde bu bilgilere ulaşma ihtiyacını çözebilmek için ilgili proje yöneticilerinden geçmişe dönük olarak bu bilgileri kâğıt formlarda ve elektronik formlarda doldurması istendi. Girilenler sayısal olmayan 5 ölçekli verilerek sayısallaştırıldı, ölçeklendirildi. Nihai olarak kişi, adet olmayan ölçekler 0-1 aralığında ölçeklendirilmiş oldu.

Oluşturulan nihai listenin ilgili kısımları proje yöneticileriyle paylaşıldı. Proje yöneticileri kendilerinin yönettiği projeler arasında kıyas yaparak yanlış değerlendirildiğini düşündüğü ölçekleri gerekçeleriyle beraber ilettiler. Uygun görülenler düzenlendi.

Bu 109 proje tek bir konu değil, çeşitli konulardadır. Dolayısıyla çeşitlilik zenginliği göstermektedir. Bu 109 projenin temel konuları şu şekildedir :

- İş Akış Uygulamaları
- Harita Uygulamaları
- Muhaberat
- Depo/Stok
- Hukuk
- Elektirik Ödemeleri
- Performans

- Portal
- Transmisyon
- Santral Planlama
- Tesis
- Servisler
- Alarm İzleme
- Optimizasyon
- Hücre Planlama
- Yatırım Planlama

Veri toplama süreci içinde proje yöneticilerinden ölçütler için katsayı önerileri de alındı. Bunların dağılımı, ortalaması, minimum ve maksimum değerleri incelendi. Bu katsayıların yapay sinir ağında bağlantıların ağırlıkları olarak kullanılması düşünülmüştür. Ancak çok çeşitlilikte proje olmasından dolayı grup bazında ortalamaları almanın daha doğru bir yaklaşım olabileceği, ortalamaları doğrudan ağırlık olarak kullanmanın fazla avantaj sağlamayacağı düşünüldü. Bu çalışmanın devamı olarak geliştirme noktalarından biri proje grupları oluşturmak ve proje grupları bazında bu ağırlıkları kullanmak olabilir.

5.7 Ölçütlerin Karşılaştırılması

Oluşturduğumuz ölçüt kümesinin mevcut ölçüt kümeleriyle karşılaştırmasında anlaşılması ve kapsamı göstermesi açısından fayda vardır. YEEM teorik kapsam olarak neredeyse konu olan tüm ölçütleri kapsayan bir kümedir. Bazı ölçütleri doğrudan bazılarını ise dolaylı olarak kapsamaktadır. Genel bir karşılaştırma çizelgesi yerine anlaşılırlığı artırmak için gruplar bazında verilmiştir.

5.7.1 İşin Büyüklüğü Grubunun Ölçütleri Karşılaştırması

İşin büyüklüğü grubunda bulunan ölçütler, bu konuda en kapsamlı olan COCOMO II Post Mimari ölçütlerinin tümünü kapsıyor. Tüm ölçütleri kapsamanın yanı sıra işlev puan, projenin bütçesi ve önemini de yeni ölçütler olduğu görülmektedir. Ürünün beklenen özellikleri kısmında ise kolay kullanım ve bakım yapılabilinirlik ölçütleri de önemli ve yeni ölçütlerdir. İşin büyüklüğü ölçütlerinin karşılaştırılması Çizelge 5.4'te gösterilmektedir. Çizelge 5.4'te "Hayır$_1$" ve "Hayır$_2$" olarak iki "Hayır" görülmektedir. "Hayır$_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "Hayır$_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi

gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "Hayır$_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.4 İşin Büyüklüğü Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
Ana Ölçütler	Ölçütler	Alt Ölçütler							
A. İşin Büyüklüğü (Ürün)	1.Karmaşıklık	a.Veritabanı büyüklüğü [DATA]	DATA	Veritabanı Büyüklüğü *(Database Size)*	Evet	Hayır$_1$	Evet	Evet	Evet
		b.Modüllerin karmaşıklığı [CPLX]	CPLX	Yazılım Ürünü Karmaşıklığı *(Software Product Complexity)*	Evet	Hayır$_1$	Evet	Evet	Evet
		c.Yeniden kullanılabilinirlik [RUSE]	RUSE	Gereken Yeniden Kullanılabilinirlik *(Required Reusability)*	Evet	Evet	Evet	Evet	Hayır$_1$
	2.Function Point	2.Function Point			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	3.Önemi	3.Önemi			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	4.Ayrılan Bütçe	4.Ayrılan Bütçe			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	5.Ürün beklenen özellikleri	a.Esneklik			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Doğruluk	RCPX	Ürün Doğruluğu ve Karmalıklığı (Product Reliability and Complexity)	Hayır$_1$	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$
		c.Kolay kullanım			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		d.Dokumantasyon	DOCU	Yaşam Döngüsü ihtiyacına uygun dokümantasyon *(Documentation Match to Life-Cycle Needs)*	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$

		e.Bakım Yapılabilirlik			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		f.Güvenilirlik	SECU	Sınıflandırılmış Güvenlik Uygulaması *(Classified Security Application)*	Hayır$_1$	Hayır$_1$	Evet	Evet	Hayır$_1$
			RELY	Gereken Yazılım Güvenilirliği *(Required Software Reliability)*	Evet	Hayır$_1$	Evet	Evet	Evet

5.7.2 Kaynak Grubunun Ölçütlerinin Karşılaştırması

Kaynak grubunda bulunan ölçütler bu konuda en kapsamlı ölçüt kümesi olan COCOMO 81'in ölçütlerinin tümünü kapsıyor. Tümünü kapsamanın yanı sıra proje yöneticisi ve sayılarıda ele alıyor. Kaynak grubunun ölçütlerinin karşılaştırılması Çizelge 5.5'te gösterilmektedir. Çizelge 5.5'te ve 5.6'da "Hayır$_1$" ve "Hayır$_2$" olarak iki "Hayır" görülmektedir. "Hayır$_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "Hayır$_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "Hayır$_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.5 Kaynak Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCO MO II Erken Tasarım	REVIC	COCO MO 87, AD A 87, APM 88	COCO MO 81, COCO MO 85
B. Kaynak	1.Çalışanların yeterlilikleri	a.Proje yöneticisinin Yeterlilikleri			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Yazılım Geliştiricilerin Yeterlilikleri	PCAP	Programcı Yeteneği (Programmer Capability)	Evet	Hayır$_1$	Evet	Evet	Evet
		c.Analistlerin Yeterlilikleri	ACAP	Analist Yeteneği (Analyst Capability)	Evet	Hayır$_1$	Evet	Evet	Evet

	2.Çalışanların projeye katılım oranları [PCON]	2.Çalışanların projeye katılım oranları [PCON]	PCON	Personel Sürekliliği (Personnel Continuity)	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$
	3.Çalışan kişi sayıları	a.Çalışan yönetici sayısı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Çalışan yazılım geliştirici sayısı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		c.Çalışan analist sayısı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		d.Çalışan test eden sayısı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	4.Donanım durumu	a.Zaman Kısıtı [TIME]	TIME	Yürütme Zamanı Kısıtı (Execution Time Constraint)	Evet	Hayır$_1$	Evet	Evet	Evet
		b.Ana bellek kısıtı [STOR]	STOR	Ana Depolama Kısıtı (Main Storage Constraint)	Evet	Hayır$_1$	Evet	Evet	Evet
		c.Bilgisayar platform değişim olasılığı [VIRT]	VIRT	Sanal Makine Uçuculuğu (Virtual Machine Volatility)	Hayır$_1$	Hayır$_1$	Evet	Hayır$_1$	Evet
		d.Bilgisayar iş geri dönüş zamanı [TURN]	TURN	Bilgisayar İş Bitirme Zamanı (Computer Turnaround Time)	Hayır$_1$	Hayır$_1$	Evet	Evet	Evet

Oluşturulan yeni ölçüt kümesinde çalışanların yetkinlikleri bir ölçüttür. Dolayısıyla yeterlilik ve tecrübe dolaylı olarak kapsanmaktadır.

Çizelge 5.6 Doğrudan kapsanmayan Kaynak Ölçütleri

Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
APEX AEXP	Uygulama Tecrübesi (Applications Experience)	Evet	Hayır$_1$	Evet	Evet	Evet
PREX	Personel Tecrübesi (Personnel Experience)	Hayır$_1$	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$

PERS	Personel Yeteneği (Personnel Capability)	$Hayır_1$	Evet	$Hayır_1$	$Hayır_1$	$Hayır_1$
VEXP	Sanal Makine Tecrübesi (Virtual Machine Experience)	$Hayır_1$	$Hayır_1$	Evet	Evet	Evet

5.7.3 Risk Grubunun Ölçütlerinin Karşılaştırması

Risk grubunda bulunan ölçütler konusunda YEEM oldukça yenilik getirmiştir. Önemine binaen ölçütlerin neredeyse tamamı yenidir. Risk grubunun ölçütlerinin karşılaştırılması Çizelge 5.7'de gösterilmektedir. Çizelge 5.7'de "$Hayır_1$" ve "$Hayır_2$" olarak iki "Hayır" görülmektedir. "$Hayır_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "$Hayır_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "$Hayır_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.7 Risk Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
C. Risk	1.Kaynak riskleri ve gerçekleşme oranı	a.Bütçe Riski Gerçekleşme Olasılığı			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		b.Bütçe Riski Etkisi			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		c.Çalışan Riski Gerçekleşme olasılığı			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		d.Çalışan Riski Etkisi			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		e.Donanım Riski Gerçekleşme olasılığı			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		f.Donanım Riski Etkisi			$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$	$Hayır_2$
		g.Ürün tanımının veya kapsamın değişiklik gerçekleşme olasılığı	RVOL	Gereken Uçuculuk (Requirements Volatility)	$Hayır_1$	$Hayır_1$	Evet	$Hayır_1$	$Hayır_1$

		h.Ürün tanımının veya kapsamın değişiklik etkisi			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$

5.7.4 Teknoloji Grubunun Ölçütlerinin Karşılaştırması

Teknoloji grubunda bulunan ölçütler bu konuda en kapsamlı ölçüt kümesi olan COCOMO 81'in ölçütlerinin tümünü kapsıyor. Tümünü kapsamasının yanı sıra uygulama tecrübesi ve çok taraflı geliştirme de araçların kolaylığı, bilinirliği ve modern programlama teknikleri ölçütlerinide dolaylı olarak içermektedir. Teknoloji grubunun ölçütlerinin karşılaştırılması Çizelge 5.8'de ve 5.9'da gösterilmektedir. Çizelge 5.8'de ve 5.9'da "Hayır$_1$" ve "Hayır$_2$" olarak iki "Hayır" görülmektedir. "Hayır$_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "Hayır$_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "Hayır$_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.8 Teknoloji Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCO MO II Erken Tasarım	REVIC	COCO MO 87, AD A 87, APM 88	COCO MO 81, COCO MO 85
D. Teknoloji	1.Yazılım Geliştirme Araçlarının kolaylığı	1.Yazılım Geliştirme Araçlarının kolaylığı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	2.Yazılım Geliştirme Araçlarının Developer'larca bilinirliği	2.Yazılım Geliştirme Araçlarının Developer'larca bilinirliği	LTEX	Dil ve Aracı Tecrübesi (Language and Tool Experience)	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$
			LEXP	Programlama Dili Tecrübei (Programming Language Experience)	Hayır$_1$	Hayır$_1$	Evet	Evet	Evet

	3.Yazılım Geliştirme araçları kullanımı *(Proje ekibi tarafından bilgisayar destekli yazılım geliştirme araçlarının kullanım oranı)*	3.Yazılım Geliştirme araçları kullanımı (Proje ekibi tarafından bilgisayar destekli yazılım geliştirme araçlarının kullanım oranı)	TOOL	Yazılım Araçlarının Kullanımı (Use of Software Tools)	Evet		Evet	Evet	Evet
			FCIL	Araç Gereç (Facilities)	Hayır$_1$	Evet			
	4.Modern Programlama Teknikleri [MODP]	4.Modern Programlama Teknikleri [MODP]	MODP	Modern Programlama Pratiği (Modern Programming Practices)	Hayır$_1$	Hayır$_1$	Evet	Evet	Evet

Çizelge 5.9 Doğrudan kapsanmayan Teknoloji Ölçütleri

Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
SITE	Çokyerli Geliştirme (Multisite Development)	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$
PLEX PEXP	Platform Tecrübesi (Platform Experience)	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$

5.7.5 Ortam Grubunun Ölçütlerinin Karşılaştırması

Ortam grubunda bulunan ölçütler konusunda YEEM oldukça yenilik getirmiştir. Önemli bir grup olmasının yanı sıra buradaki ölçütler de yenidir. Doğrudan ve dolaylı olarak konu olabilecek tüm ölçütleri içermektedir. Ortam grubunun ölçütlerinin karşılaştırılması Çizelge 5.10'da gösterilmektedir. Çizelge 5.10'da "Hayır$_1$" ve "Hayır$_2$" olarak iki "Hayır" görülmektedir. "Hayır$_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "Hayır$_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "Hayır$_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.10 Ortam Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
E. Ortam	1.Ortamın genel özellikleri (Ivme)	a.Yazılımın gerçekleştiği firmanın projelerin gecikenlerin oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Ortalama gecikme oranı (Yazılımın gerçekleştiği firmanın proje uzunluğu baz alınarak yüzde olarak ortalama gecikme miktarı)			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	2..Sahiplenilme	a. Yönetimin projeye sahiplenme oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b. Proje yöneticisinin projeye sahiplenme oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		c. Analistlerin projeye sahiplenme oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		d. Developerların projeye sahiplenme oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		e. Müşterilerin projeye sahiplenme oranı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	3.Baskı	a.Üstten/Yönetimden gelen baskı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Müşteriden gelen baskı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		c.Pazardan gelen baskı			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	4.Zaman kullanım durumu (1 gundeki gunluk kayıp)	a.Çalışanların günde ortalama kaç defa kesildiği (adet)			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		b.Ortalama kesilme süresi (dk)			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$

		c.Kesilmeden sonra normala dönüş ortama süresi (dk)			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
		d.Ortalama yapılan fazla mesai saati (saat)			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$
	5.Verimlilik durumu	5.Verimlilik durumu			Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$	Hayır$_2$

Çizelge 5.11 Doğrudan kapsanmayan Ortam Ölçütleri

Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
VMVH	Sanal Makine Uçuculuğu : Ana Bilgisayar (Virtual Machine Volatility: Host)	Hayır$_1$	Hayır$_1$	Hayır$_1$	Evet	Hayır$_1$
VMVT	Sanal Makine Uçuculuğu : Hedef (Virtual Machine Volatility: Target)	Hayır$_1$	Hayır$_1$	Hayır$_1$	Evet	Hayır$_1$
PVOL	Platform Uçuculuğu (Platform Volatility)	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$	Hayır$_1$
PDIF	Platform Zorluğu "(Platform Difficulty)	Hayır$_1$	Evet	Hayır$_1$	Hayır$_1$	Hayır$_1$
PLAT	Platform	Hayır$_1$	Hayır$_1$	Evet	Hayır$_1$	Hayır$_1$

5.7.6 Planlar ve Tahminler Grubunun Ölçütlerinin Karşılaştırması

Planlar ve Tahminler grubunda bulunan ölçütler konusunda YEEM oldukça yenilik getirmiştir. Buradaki ölçütler sonucu doğrudan etkileyen, kritik ölçütlerdir. Bunların ele alınmasıyla verimi artırılmıştır. Uzman görüşü modelinin bu şekilde ölçüt olarak ele alınmasıyla melez bir yapı oluşturmuş olmatadır. Bu sayede başarı ihtimali artmanın yanı sıra yeni olarak melezlik dışında plan ve tahminin oluşturduğu ortam da ele alınmış olunmaktadır. Planlar ve tahminler grubunun ölçütlerinin karşılaştırılması Çizelge 5.12'de gösterilmektedir. Çizelge 5.12'de "Hayır$_1$" ve "Hayır$_2$" olarak iki "Hayır" görülmektedir. "Hayır$_1$" ile ismi geçen ölçütün ilgili ölçüt kümesi için gündemde olduğu ancak ilgili ölçüt kümesinde mevcut olmadığını belirtmek için kullanılmıştır. "Hayır$_2$" ile ismi geçen ölçütün ilgili ölçüt kümesinde olmadığı gibi gündemde olmadığını belirtmek için kullanılmıştır. Dolayısıyla "Hayır$_2$" ile belirtilen ölçütler YEEM ile yeni olan ölçütlerdir.

Çizelge 5.12 Plan ve Tahmin Ölçütleri Karşılaştırması

YEEM			Ölçüt	Uzun İsmi	COCOMO II Post Mimari	COCOMO II Erken Tasarım	REVIC	COCOMO 87, ADA 87, APM 88	COCOMO 81, COCOMO 85
F. Planlar ve Tahminler	Planlar	a.Hedeflenen/Planlanan Zaman (Kaç Gün)	SCED	Gereken Geliştirme Zaman Planı (Required Development Schedule)	Evet	Evet	Evet	Evet	Evet
		b.Hedeflenen zaman planı yanılma oranı (% kac)			Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]
		c. Hedeflenen zamanda bitmemesi durumunda katlanılabilinirlik - Tolerans (%)			Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]
	Tahminler	a.Gerçekçi Tahmini zaman			Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]
		b.Tahmini yapan kişi veya kişilerin tahmin tutturma oranı			Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]	Hayır[2]

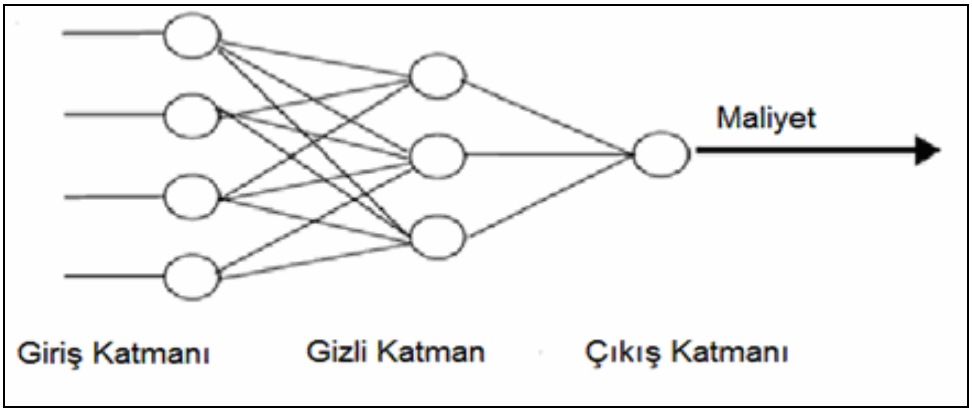
6. YAPAY SİNİR AĞI KULLANILARAK YAZILIM MALİYET TAHMİNLEME MODELİ OLUŞTURMA ÇALIŞMASI

Temelde iki tür maliyet tahmin yöntemi vardır: algoritmik ve algoritmik olmayan. Algoritmik modellerin bazıları istatistiği kullanarak basit aritmetik formüller kullanmaktadır. Algoritmik modeli bir takım değişkenleri kullanan bir fonksiyon (6.1) olarak düşünebiliriz.

$$\text{Maliyet} = f(x_1, x_2, \ldots, x_n) \qquad (6.1)$$

Burada $x_1,x_2,..x_n$ değişkenlerini maliyet faktörleri olarak düşünebiliriz. Birçok maliyet çalışmasında kullanılan COCOMO II modelinin 4 temel maliyet faktörü vardır: ürün faktörleri, bilgisayar faktörleri, personel faktörleri ve proje faktörleri. İkinci kısımda önerdiğimiz ölçüt kümesi COCOMO'nun tüm ölçütlerini kapsamaktadır. Ancak bu 4 temel faktör yerine 6 temel faktör getirilmiştir.

Maliyet tahminleme aslında oldukça karmaşık bir problemdir ancak bugüne kadar gereken ilgiyi görmemiştir. Araştırmacılar değişik yaklaşımlar getirmeye çalışmışlardır. Son zamanlarda yeni yapay zekâ yaklaşımları geliştirilmeye başlanmıştır. Temel ve basit gösterimle durum şekil 6.1'de gösterilmiştir. Yapay sinir ağları kullanılarak yapılan bazı çalışmalar vardır. Bu konu henüz yeni gelişen bir konudur. Bu konuda yapılmış çalışmaları şekil 6.2'de gösterilmiştir.



Şekil 6.1 Yapay Sinir ağı (YSA) Topolojisi

Yapılan çalışmalarda başarıyı artırmak için daha çok tahminleme üzerinde çalışılmış ve girdiler çok önemsenmemiştir. Bu çalışmada bir farklılık yaratarak önemli bir sonuç farkı olduğunu düşünmekteyiz. Ölçüt kümesinin yazılım maliyet tahmininde çok önemli bir rolü

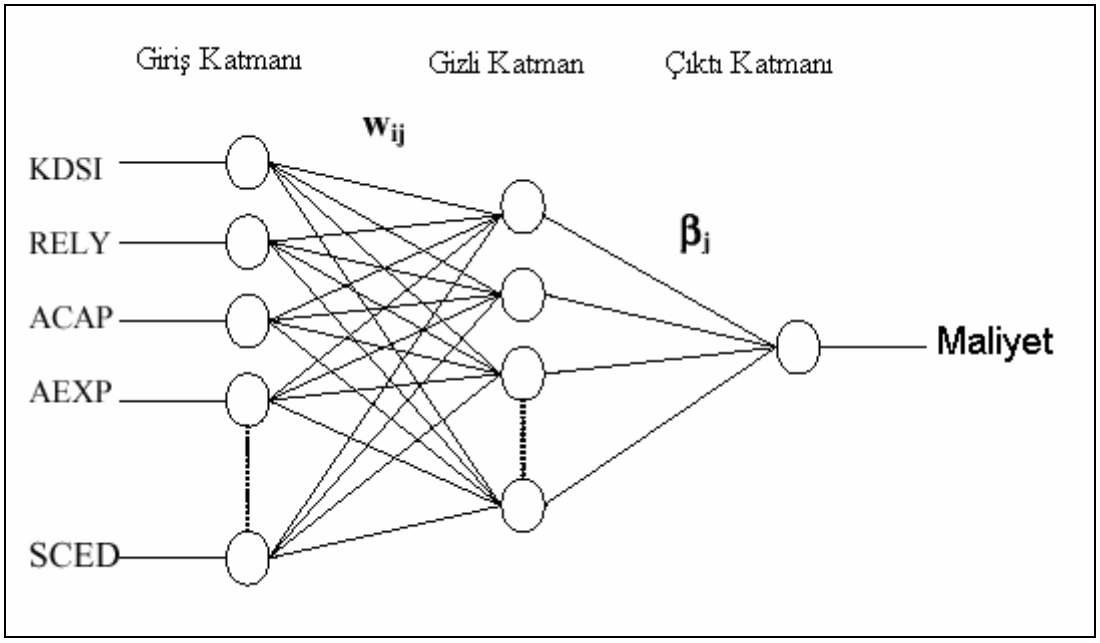
vardır. Birçok model ölçüt kümesi üzerinde çok çalışma yapmadan COCOMO'yu kullanmayı tercih edip çoğunlukla tahminleme yöntemi üzerinde çalışmayı tercih etmişlerdir. Ancak ölçüt kümesi sonucu doğrudan etkilemektedir. Dolayısıyla iyi bir tahminleme çalışması yapmadan önce iyi bir ölçüt kümesi ile işe başlamak önemlidir.

Yapay sinir ağlarında yazılım maliyet tahmininde kullanılabilinecek birçok model mevcuttur. Bunları iki temel grupta sınıflandırabiliriz.

1. İleri beslemeli ağlar (Feed-Forward networks) : Bunlarda ağ üzerinde döngü (loop) yoktur.

2. Geri yayılımlı birçok katmanlı ileri beslemeli ağlar (*Feed-Forward multi-layer perceptron with Back propagation*) : En çok kullanılan gruptur. Ağ üzerinde döngü vardır ve ağda düğümler katmanlar şeklinde düzenlenmiştir. Bu çalışma temel olarak bu yapıyı kullanmayı düşündük.

Temel mantığı bir örnekle açıklamakta fayda var. COCOMO ölçüt kümesini yazılım maliyet tahminlemesi için yapay sinir ağı çalışmasını örnek olarak ele alabiliriz. Bu ağ Şekil 6.2'de gösterilen bir yapay sinir ağı olacaktır.



Şekil 6.2 COCOMO kullanılarak yapılan ağ

Bu örnek ağ çeşitli parametreleri alarak bunları işleyerek maliyeti hesaplamaktadır. Ağdaki her düğüm lineer olmayan bir fonksiyon ile hesaplama yapar ve sonucu çıktı olarak verir. Yapay sinir ağlarında genel olarak en fazla kullanılan fonksiyon Sigmoid (6.2) fonksiyonudur.

Sigmoid fonksiyon:

$$f(x) = \frac{1}{1+e^{-x}} \tag{6.2}$$

Burada ağın yeni projeler için çalışmaya başlamadan ve sonuçlar vermeden önce elde mevcut olan çeşitli girdi ve çıktıları belli olan veri kümeleriyle eğitilmesi gerekmektedir. COCOMO 81 veri seti yazılım maliyet tahminleme çalışmalarının çoğunda kullanılmıştır. Bu çalışmada başlangıç olarak COCOMO 81 seti kullanmayı planladık. Daha sonra gerçek hayattan alınan 109 tane proje ile destekledik. COCOMO 81'de 63 tane proje mevcuttur. Her projede 17 özellik vardır. Bunlardan biri proje büyüklüğüdür. Proje büyüklüğünü KDSI (Kilo Delivered Sources Instructions) biriminde vermiştir. Bir diğer parametre ise proje modudur. Proje modu olarakta "organik", "yarı ayrık", "gömülü" sıfatları kullanılmıştır. Geri kalan 15 parametre "çok az", "az", "normal", "yüksek", "çok yüksek", "aşırı yüksek" sıfatları kullanılmıştır. Burada aslında bulanık mantık kullanımı yatkınlığı mevcuttur. Her seçilen özellik için oluşan bulanık küme için if-then bulanık kuralları basitçe gösterilen yapay sinir ağından çıkartılabilinir.

Yapay sinir ağı kullanılarak yapılacak tahminlemede kullanılacak mimari, öğrenme mimarisi, aktivasyon fonksiyonu için kesin kararlar ve seçimler yapmak gerekmiştir. Birçok projede mimari 3 katmanlı algılayıcı kullanılmıştır. Genelde sigmoid fonksiyonu kullanılmıştır. Çalışmanın birinci fazında ilk eğitim için COCOMO 81'in verilerini kullanacak olursak 17 girişi ve sadece tek çıkışı olan bir yapı kurulur. Çıkış ise maliyetdir (6.3). Ancak gerek bu kullanılacak COCOMO 81 veri kümesi olsun, gerek toplanacak olan veri kümesinde muhtemelen veriler bulanık olacaktır. Dolayısıyla bunları normalize etmek gerekecektir. Bu yüzden başka algoritmalar, başka fonksiyonlar kullanmaya çalıştık. Ağırlıklar başlangıçta raslantısal olarak verildi. Oldukça küçük değerler verilerek öğrenme süreci başlatılmıştır. Geri yayılım kullanılarak eğitime başlanır, ancak başarılı kabul edilebilinecek bir duruma gelemedik.

$$Çaba = \sum_{j=1}^{h} Z_j \beta_j \quad Z_j = f(\sum_{j=1}^{n} w_{ij} x_i) \tag{6.3}$$

Burada

f : Sigmoid fonksiyon

Wij : Ağırlık (giriş katmandan gizli (ara) katmana)

Bj : Ağırlık (gizli katmandan çıkış katmanına)

6.1 Normalizasyon Tekniği

YSA (Yapay Sinir Ağı) giriş ve çıkışlarında kullanılabilinecek birkaç normalizasyon tekniği kullanılıp, değerlendirildi. İlk olarak verilerin ortalamasını tüm verilerden çıkararak, verileri sıfır merkezli hale getirip, her kolon için standart sapmaya bölme tekniği kullanıldı. Böylece veriler -1 ile +1 arasında kümelenmektedir. Bu yöntemi kullandığınızda normalde [-1..+1] aralığının dışında bir değer elde edilmemesi gerekir, bu değerlerin dışında kalan verilerin yapay sinir ağının eğitimini kötü yönde etkilemesi de mümkündür, bu nedenle kontrol edilmelerini ve gerekiyorsa gözardı edilmelerinde fayda olabilir. Çıkışlar ise 0.1 ile 0.9 arasına sıkıştırıldığında, hata fonksiyonunun 0'a veya 1'e yakın değerlerde (tek taraftan yaklaştığı düşünüldüğünde) asimptotik olarak işaret değiştirmesi mümkün olduğundan, 0 ile 1 arasında sıkıştırıldıklarından daha iyi sonuçlar verebildiğini gördük. Bu yüzden normalizasyon yöntemi olarak maksimum değere bölme yönteminin daha başarılı sonuçlar verdiği görülmüştür.

6.2 Yapay Sinir ağlarının kullanımının zayıf yanları

Yapay sinir ağları kullanılarak yapılan yazılım maliyet tahminlemesinde aslında temelde iki zayıflık mevcuttur.

1. Bu bize önceki durumları ve verileri kullanma olanağı verir. Burada öğrenme kriteri maliyet belirleme için oldukça önemlidir. Çünkü yazılım geliştirme teknolojisi sürekli gelişmektedir.

2. Birbirine bağlı olan veriler ve birbirine bağlı olmayan veriler arasında karmaşık ilişkiler kurabilir. Birbirine bağlı olanlara örnek olarak “maliyet” ve “emek” verileri verilebilir. Birbirine bağlı olmayanlara örnek olarak maliyeti oluşturan parçacıklar verilebilir. Bunlar arasında aslında temel olarak herhangi bir ilişki yoktur. Ancak bu çalışma bize bunlar arasında bir ilişki çıkarabilir.

Bunlara ek olarak sayılabilecek zayıf yanlar:

1.Yapay sinir ağları “kapalı siyah kutu” olarak görülebilir. Dolayısıyla anlamak ve anladıktan sonra açıklamak kolay değildir.

2.Yapay sinir ağlarında kullanılacak topoloji için bir rehber yoktur. Kaç katman olacağı, her katmanda kaç düğüm olacağı, ilk değerlerin ne olacağı ile ilgili bir rehber olmamasından ötürü bu konuda tecrübe, önceki çalışmalar kullanılacaktır. Deneme ve yanılmalar ile başarılı

sonuç verebilecek bir yapı oluşturulmaktadır.

3. Yapay sinir ağlarının sınıflandırma ve kategorizasyon konusunda oldukça karmaşık problemleri çözdüğü görülmüş ve bu konularda kendini ispatlamıştır. Ancak maliyet belirleme sınıflandırmadan çok bir genelleme, genel kural çıkarma işlemidir. Dolayısıyla sınıflandırma yapılmasından ziyade tahminlemede kullanılacak olması çalışmayı zorlaştırmaktadır.

Birçok araştırmacı maliyet tahminlemede yapay sinir ağlarının kullanımını oldukça zor ve kapalı kutu olarak gördüklerinden kullanmak istememişlerdir. Buna neden olarak açıklanması ve anlatılması zor bir sistem olmasından dolayı basit sistemlere nazaran daha az güvenilir olduğunu iddia etmişlerdir. Ancak sınıflandırmada yapay sinir ağları oldukça güvenilir neticeler vermektedir. Çalışma tamamlandıktan sonra test kümeleri oluşturup deneyerek güvenilirliği test etmek mümkün olabilmektedir. Yapay sinir ağının mimarisinde, sinapslarda, ağırlıklarında saklı bilgiyi ve kullanılan algoritmanın açıklanması oldukça önemlidir.

6.3 Yapılan Ön Çalışmalar

Uzun süre C'de yapay sinir ağı çalışmasına baz oluşturacak kodlama yapıldı. Bu sürede yapay sinir ağı uygulamaları ile ilgili genel mantık anlaşıldı. C'de yazılan kodlamada düzenleme yapmak, değişiklik yapmak oldukça güç olmaktadır ve çok zaman almaktadır. Diğer bir deyişle bakım yapılabilirliği oldukça düşüktü. Bu kodlama genel özelliklerin anlaşılması açısından faydalı olmuştur.

Daha sonra bu konuda yapılan birçok çalışmada kullanıldığı tespit edilen MatLab kullanılmaya başlanıldı. MatLab'te kolaylıkla yapıda değişiklik yapılabilinmesi sayesinde çok deneme yapılabilinmesi sağlanmıştır. Bu sayede 138 adet yapay sinir ağı modeli (Çizelge 6.1) deneme fırsatı bulabildik (Ayyıldız M.,Kalıpsız O., Yavuz S, 2007).

Çizelge 6.1 Yapay Sinir ağları konusunda yapılan ön çalışma

Deneme No	Yöntem	Bağlama	Test Performansı	Test Hata	Eğitim Yöntemi
1	MLP	1:5-1-1:1	1	0	BP200,CG200,CG3b
2	MLP	1:5-3-4-1:1	1	0	BP200,CG200,CG11b
3	RBF	14:67-11-1:1	1	0	KM,KNN,PI
4	Linear	17:83-1:1	2,83E+13	1,70E+12	PI
5	GRNN	17:83-33-2-1:1	2	0	SS
6	MLP	1:5-1-1:1	1	0	BP200,CG200,CG3b
7	MLP	12:60-25-9-1:1	2	0	BP200,CG200,CG40b

8	RBF	14:67-5-1:1	2	0	KM,KNN,PI
9	Linear	16:78-1:1	2,82E+13	1,50E+12	PI
10	GRNN	17:83-33-2-1:1	2	0	SS
11	MLP	7:28-33-1:1	2	0	BP200,CG200,CG3b
12	MLP	4:19-16-4-1:1	6	0	BP200,CG200,CG102b
13	RBF	14:67-17-1:1	2	0	KM,KNN,PI
14	Linear	15:66-1:1	4,50E+13	2,20E+12	PI
15	GRNN	17:83-33-2-1:1	1	0	SS
16	MLP	1:5-1-1:1	1	0	BP200,CG200,CG3b
17	MLP	9:47-14-4-1:1	1	0	BP200,CG200,CG46b
18	RBF	14:67-13-1:1	1	0	KM,KNN,PI
19	Linear	14:62-1:1	6,54E+13	4,20E+12	PI
20	GRNN	17:83-33-2-1:1	1	0	SS
21	MLP	2:9-1-1:1	3	0	BP200,CG200,CG0b
22	MLP	17:83-26-6-1:1	3	0	BP200,CG200,CG81b
23	RBF	14:67-7-1:1	2	0	KM,KNN,PI
24	Linear	13:57-1:1	4,50E+13	2,30E+12	PI
25	GRNN	17:83-33-2-1:1	1	0	SS
26	MLP	13:58-3-1:1	2	0	BP200,CG200,CG13b
27	MLP	15:73-34-10-1:1	10	0	BP200,CG8c,CG8b
28	RBF	14:67-8-1:1	1	0	KM,KNN,PI
29	Linear	12:53-1:1	4,34E+13	2,10E+12	PI
30	GRNN	16:78-33-2-1:1	1	0	SS
31	MLP	12:60-8-1:1	6	0	BP200,CG5c,CG11b
32	MLP	16:80-34-12-1:1	2	0	BP200,CG7c,CG38b
33	RBF	14:67-9-1:1	1	0	KM,KNN,PI
34	Linear	11:45-1:1	7,75E+13	3,80E+12	PI
35	GRNN	15:75-33-2-1:1	1	0	SS
36	MLP	10:42-7-1:1	2	0	BP200,CG6c,CG5b
37	MLP	16:80-34-13-1:1	2	0	BP200,CG5c,CG148b
38	RBF	14:67-10-1:1	2	0	KM,KNN,PI
39	Linear	10:42-1:1	3,91E+14	2,20E+13	PI
40	GRNN	14:70-33-2-1:1	1	0	SS
41	MLP	17:83-9-1:1	2	0	BP200,CG5c,CG45b
42	MLP	12:65-34-10-1:1	3	0	BP200,CG18c,CG89b
43	RBF	14:67-12-1:1	1	0	KM,KNN,PI
44	Linear	9:39-1:1	1	4,80E+13	PI
45	GRNN	13:65-33-2-1:1	1	0	SS
46	MLP	14:73-6-1:1	5	0	BP200,CG5c,CG75b
47	MLP	11:58-34-7-1:1	7	0	BP200,CG7c,CG79b
48	RBF	14:67-11-1:1	2	0	KM,KNN,PI
49	Linear	8:34-1:1	2	PI	
50	GRNN	12:62-33-2-1:1	1	0	SS

51	MLP	17:83-12-1:1	5	0	BP200,CG5c,CG0b
52	MLP	12:53-34-4-1:1	4	0	BP200,CG8c,CG63b
53	RBF	14:67-11-1:1	2	0	KM,KNN,PI
54	Linear	7:28-1:1	3	1	PI
55	GRNN	11:59-33-2-1:1	1	0	SS
56	MLP	14:61-9-1:1	4	0	BP200,CG6c,CG136b
57	MLP	17:83-34-11-1:1	2	0	BP200,CG5c,CG21b
58	RBF	14:67-11-1:1	2	0	KM,KNN,PI
59	Linear	6:25-1:1	0		
60	GRNN	10:54-33-2-1:1	1	0	SS
61	MLP	10:56-11-1:1	5	0	BP200,CG6c,CG25b
62	MLP	13:56-34-9-1:1	3	0	BP200,CG5c,CG2b
63	RBF	14:67-11-1:1	1	0	KM,KNN,PI
64	Linear	5:21-1:1	1	PI	
65	GRNN	9:50-33-2-1:1	1	0	SS
66	MLP	16:79-9-1:1	4	0	BP200,CG6c,CG39b
67	MLP	16:78-34-7-1:1	1	0	BP200,CG8c,CG20b
68	RBF	14:67-11-1:1	1	0	KM,KNN,PI
69	Linear	4:18-1:1	2	PI	
70	GRNN	8:46-33-2-1:1	1	0	SS
71	MLP	10:55-8-1:1	12	0	BP200,CG6c,CG0b
72	MLP	12:60-34-13-1:1	14	0	BP200,CG8c,CG1b
73	RBF	14:67-1-1:1	1	0	KM,KNN,PI
74	Linear	3:13-1:1	4	0	PI
75	GRNN	7:43-33-2-1:1	1	0	SS
76	MLP	10:46-9-1:1	10	0	BP200,CG6c,CG0b
77	MLP	15:74-34-8-1:1	0	0	BP200,CG30c,CG2b
78	RBF	14:67-17-1:1	1	0	KM,KNN,PI
79	Linear	2:10-1:1	1	0	PI
80	GRNN	6:40-33-2-1:1	1	0	SS
81	MLP	13:67-5-1:1	4	0	BP200,CG6c,CG8b
82	MLP	12:50-34-10-1:1	1	0	BP200,CG5c,CG4b
83	RBF	14:67-4-1:1	2	0	KM,KNN,PI
84	Linear	1:5-1:1	2	0	PI
85	GRNN	5:36-33-2-1:1	1	0	SS
86	MLP	15:65-9-1:1	4	0	BP200,CG8c,CG32b
87	MLP	17:83-34-8-1:1	1	0	BP200,CG6c,CG41b
88	RBF	14:67-6-1:1	2	0	KM,KNN,PI
89	GRNN	4:28-33-2-1:1	1	0	SS
90	MLP	17:83-11-1:1	7	0	BP200,CG6c,CG9b
91	MLP	13:58-34-10-1:1	8	0	BP200,CG9c,CG1b
92	RBF	14:67-13-1:1	2	0	KM,KNN,PI
93	GRNN	3:16-33-2-1:1	4	0	SS

94	MLP	13:60-7-1:1	3	0	BP200,CG6c,CG23b
95	MLP	8:31-34-8-1:1	6	0	BP200,CG6c,CG0(En iyi)
96	RBF	14:67-7-1:1	1	0	KM,KNN,PI
97	GRNN	2:10-33-2-1:1	5	0	SS
98	MLP	14:64-10-1:1	4	0	BP200,CG6c,CG58(En iyi)
99	MLP	16:80-34-7-1:1	4	0	BP200,CG6c,CG0(En iyi)
100	RBF	14:67-8-1:1	1	0	KM,KNN,PI
101	GRNN	1:5-33-2-1:1	1	0	SS
102	MLP	15:77-14-1:1	12	0	BP200,CG6c,CG0(En iyi)
103	MLP	8:37-34-4-1:1	3	0	BP200,CG6c,CG5(En iyi)
104	RBF	14:67-9-1:1	2	0	KM,KNN,PI
105	MLP	15:73-8-1:1	8	0	BP200,CG5c,CG5(En iyi)
106	MLP	17:83-34-8-1:1	3	0	BP200,CG6c,CG0(En iyi)
107	RBF	14:67-10-1:1	1	0	KM,KNN,PI
108	MLP	14:61-13-1:1	4	0	BP200,CG6c,CG43(En iyi)
109	MLP	17:83-34-12-1:1	4	0	BP200,CG5c,CG19(En iyi)
110	RBF	14:67-12-1:1	2	0	KM,KNN,PI
111	MLP	11:56-8-1:1	4	0	BP200,CG6c,CG3(En iyi)
112	MLP	15:75-34-6-1:1	7	0	BP200,CG6c,CG3(En iyi)
113	RBF	16:78-11-1:1	2	0	KM,KNN,PI
114	MLP	11:48-5-1:1	1	0	BP200,CG5c,CG21(En iyi)
115	MLP	15:72-34-10-1:1	2	0	BP200,CG5c,CG0(En iyi)
116	RBF	16:78-11-1:1	1	0	KM,KNN,PI
117	MLP	16:75-12-1:1	7	0	BP200,CG5c,CG128(En iyi)
118	MLP	12:59-34-6-1:1	8	0	BP200,CG6c,CG2(En iyi)
119	RBF	16:78-11-1:1	1	0	KM,KNN,PI
120	MLP	10:50-5-1:1	2	0	BP200,CG5c,CG81b
121	MLP	15:72-34-8-1:1	4	0	BP200,CG8c,CG21b
122	RBF	15:74-11-1:1	2	0	KM,KNN,PI
123	MLP	15:74-10-1:1	6	0	BP200,CG5c,CG37b
124	MLP	11:54-34-10-1:1	5	0	BP200,CG7c,CG129b
125	RBF	14:71-11-1:1	2	0	KM,KNN,PI
126	MLP	12:56-8-1:1	5	0	BP200,CG5c,CG1b
127	MLP	14:63-34-17-1:1	3	0	BP200,CG6c,CG71b
128	RBF	13:68-11-1:1	2	0	KM,KNN,PI
129	MLP	15:72-9-1:1	3	0	BP200,CG6c,CG109b
130	MLP	12:60-34-6-1:1	4	0	BP200,CG5c,CG2b
131	RBF	12:64-11-1:1	2	0	KM,KNN,PI
132	MLP	12:60-11-1:1	2	0	BP200,CG6c,CG79b
133	MLP	14:65-34-9-1:1	7	0	BP200,CG21c,CG0b
134	RBF	11:52-11-1:1	2	0	KM,KNN,PI
135	MLP	7:39-7-1:1	3	0	BP200,CG6c,CG4b

136	MLP	14:70-34-10-1:1	1	0	BP200,CG5c,CG1b
137	RBF	10:49-11-1:1	1	0	KM,KNN,PI
138	MLP	17:83-34-1:1	13	0	BP5b

Çizelge 6.3'te son kolonda geçen kısaltmaların açıklaması :

- BP : Back Propagation
- CG : Conjugate Gradient Descent
- QN : Quasi-Newton
- LM : Levenberg-Marquardt
- QP : Quick Propagation
- DD : Delta-Bar-Delta
- SS : (sub)Sample
- KM : K-Means (Center Assignment)
- EX : Explicit (Deviation Assignment)
- IS : Isotropic (Deviation Assignment)
- KNN : K-Nearest Neighbour (Deviation Assignment)
- PI : Pseudo-Invert (Linear Least Squares Optimization)
- KO : Kohonen (Center Assignment)
- PN : Probabilistic Neural Network training
- GR : Generalised Regression Neural Network training
- PCA : Principal Components Analysis

şeklindedir. Burada geçen *b* harfi "en iyi" olmayı *c* harfi "yakınsamayı" ifade etmektedir. Kullanılan algoritmanın yanında yazan sayı adım sayısı (epoch) olup sonuca ne zaman ulaşıldığı bilgisini verir.

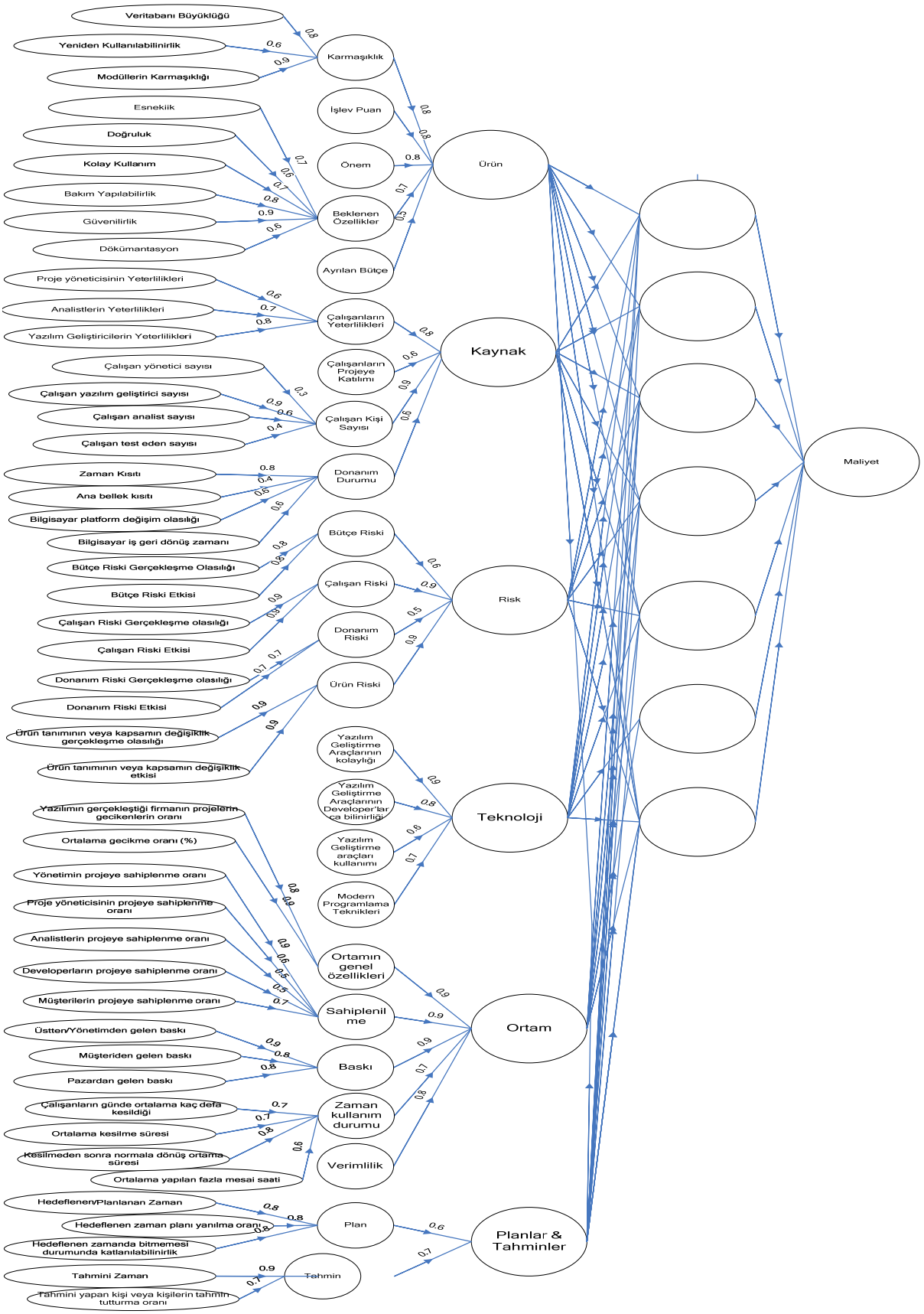
Performans ve hata sütunlarındaki değer kullanılan yönteme göre değişik anlamlar kazanmaktadır. Regresyon ağında standart sapmayı, sınıflandırma algoritmalarında kaç örneği yakaladığını belirtir. Pek fazla optimizasyon yapılmadan önceki durumlar içinde MLP'nin önce en başarılı yöntem olduğu burada görülmüştür.

6.4 Yapay Sinir Ağı topolojisini oluşturma

Bu konuda yapılmış çalışmalarda kullanılan ölçüt kümesi girdi katmanı olarak kullanılmıştır. Göreceli hatanın miktarının ortalaması (MMRE) göz önüne alınarak en başarılı olanın Wittig

ve Finnie çalışmasıdır. Bu çalışmada COCOMO ölçüt kümesini doğrudan girdi olarak kullanmıştır. Bu çalışmada baz alarak önerdiğimiz ölçüt kümesi bu yapı üzerine şekillendirilebilinir. Bu durumu şekil 6.2'de görebilirsiniz. Burada çok katmanlı algılayıcı ağ topolojisini kullanılmıştır (Ayyıldız M.,Kalıpsız O., Yavuz S, 2007).

Wittig ve Finnie'nin çalışmasındaki gibi ölçüt kümesini bu şekilde kullanmayı biz de düşündük. Oluşturmuş YEEM ölçüt kümesini bir yapay sinir ağı modeli üzerine (Ayyıldız M.,Kalıpsız O., Yavuz S, 2007) uyguladık (Şekil 6.4).



Şekil 6.4 Tez Çalışmasında geliştirilen YEEM Yapay Sinir Ağı Topolojisi

6.5 Ulaştığımız Yapay Sinir Ağı Modelleri ve Sonuçları

6.5.1 Algoritma

Bu çalışmada çok katmanlı ileri beslemeli ve geri yayılımlı yapay sinir ağı (MLP) ve *Elman* yapay sinir ağı modellerini kullandık. Birçok denemeden sonra lineer olmayan eğri uydurmada zaten başarısı sabitlenmiş olan Levenberg-Marquardt (Reed, R. D. ve Marks, R. J., 1999) eğitim algoritması seçilip uygulanmıştır.

Yapay sinir ağı kullanarak yapılan yazılım maliyet tahminleme çalışmalarında genellikle çok katmanlı ileri beslemeli (MLP) ağlar kullanılmıştır. Ancak dinamik sistemleri modellemede yenilenen yapay sinir ağları *(RNN : Recurrent Neural Networks)* daha iyi sonuç verebilmektedir. RNN kısmen veya tamamen bağlı olabilir. *Elman* ağı, bilinen en iyi kısmi-bağlı RNN'lerden biridir. Bu çalışmada geri yayılımlı *Elman* ağ modeli eğim iniş (Gradient Descent) yöntemi kullanarak eğitilmesi çalışılmıştır.

Her iki modelde hem COCOMO 81 hem de YEEM ile kullanmıştır. Ölçüt veri kümesi ise bu modele girdi olarak verilmiştir.

6.5.2 Veri İşleme

Eğitim yapılmadan önce rasgeleliğin incelenmesi, istisnaların durumu ve ön işlemeler yapılması eğitim kalitesini artırmaktadır (Pyle D., 1999). Bazı bozuk örnekler eğitimin başarısız olmasına veya çok uzun sürmesine neden olabilirler.

6.5.2.1 Rasgelelilik ve İstisnaların Kontrolü

Henüz herhangi bir işlem yapılmaksızın verinin rasgeleliğini ve istisnaların incelenmesi oldukça önemlidir. Literatürde rasgeleliği ve istisnaları bulmaya yarayan birçok test yöntemi vardır. Bu testler temel olarak teorik ve deneysel olarak 2 kategoriye ayrılabilinir. Bu çalışmada eğitim ve test veri kümesi rasgelelik kontrolü "*Run*" testi ile yapılmıştır (Pyle D., 1999). Yaptığımız çalışmalarda COCOMO 81'de 3 tane istisnai veri olduğu gözlemlenmiştir. COCOMO 81, 63 adet projeden oluşmaktadır. Bu 3 tane istisna verinin çıkarılmasıyla geriye kalan 60 tane veri ile çalışma yapılmıştır. YEEM için ise elimizde 109 adet veri mevcuttur. Bunlardan 9 tanesinin istisnai olduğu saptanmış ve çıkarılmıştır.

6.5.2.2 Verinin Ön-işleme ve Son-işlemesi

Teorik olarak yapay sinir ağları, ham giriş verisini, herhangi bir çıkış verisine eşleyebilir.

Ancak uygulamada, verilere bazı ön işlemler uygulanması faydalı ve çoğu zaman da gereklidir. Ön işlem olarak nitelendirilebilecek işlemler arasında, zaman serilerine uygulanan basit filtreleme işleminden, görüntü verilerinden özellik çıkarma gibi daha karmaşık işlemlere kadar pek çok işlem sayılabilir. Uygulanacak ön işlemler uygulamanın ve verilerin özelliğine göre farklılık ve çeşitlilik gösterir.

Verilerin içerdiği bilgi ile ilgili verilerin seçilmesi ve gürültü niteliğindeki verilerin ayıklanması da önemlidir.

Verileri, yapay sinir ağının girişlerine ve kullanılan aktivasyon fonksiyonlarına uygun hale dönüştürmek, ağın çalışmasını hızlandıracaktır. Bu dönüşümler, verilere bir fonksiyon uygulama veya ölçekleme yolu ile yapılabilir.

Bu çalışmada veri değerleri, çıktı katmanında kullanılan sigmoid fonksiyonu ve verilerin sayısal değerlerinin birbirlerinden çok farklı aralıklarda yer aldığı göz önüne alınarak, ondalık ölçekleme yöntemi ile [0,1] aralığına sıkıştırılmıştır.

Ondalık ölçekleme yöntemi kullanıldığında, k elemanlı bir seride yer alan s elemanının, ölçeklenmiş s' değeri şu şekilde ifade edilir (Pyle D., 1999):

$$s' = \frac{s - \min(s_k)}{\max(s_k) - \min(s_k)} \tag{6.1}$$

Bu çalışmada veri [0,1] aralığına çevrilerek kullanılmış ve bu aralığına getirmek için minimum-maksimum normalizasyon yöntemi (Yavuz S. , 2006) ve maksimum'a bölme normalizasyon yöntemleri kullanılmıştır. Minimum-maksimum normalizasyon yöntemi en küçük projeyi 0'a indirgediğinden ancak gerçek hiçbir projenin 0 zamanda bitmeyeceğinden, maksimuma bölme yönteminin daha başarılı olacağı düşünülmüştür.

6.5.2.3 Hata Hesabı

Yapay sinir ağı modellerinin, eğitim, doğrulama ve test başarıları, ortalama karesel hata (*mean squared error – mse)* fonksiyonu kullanılarak değerlendirilmiştir (Pyle D., 1999).

$$\text{Ortalama Karasel Hata} = \frac{1}{N}\sum_{k=1}^{N} e(k)^2 = \frac{1}{N}\sum_{k=1}^{N} (t(k) - a(k))^2 \tag{6.2}$$

Burada *t(k)* ve *a(k)*, sırasıyla, *k*'ıncı hedef değeri ve yapay sinir ağının tahmin ettiği çıkış değerini göstermektedir. N ise, veri kümesindeki eleman sayısıdır. Bağlantıların ağırlıkları bu

hatayı en küçük hale getirecek şekilde ayarlanmaktadır.

Ayrıca, her veri kümesi için ortalama yüzde hata ve standart sapma hesapları da yapılmıştır.

$$\text{Ortalama Hata (\%)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|\text{ölçülen değer}_i - \text{tahmin}_i\right|}{\text{ölçülen değer}_i} x100 \tag{6.3}$$

$$\text{Standart Sapma} = \sqrt{\frac{\sum_{i=1}^{N}(Hata(\%)_i - \text{Ortalama Hata}(\%))^2}{N-1}} \tag{6.4}$$

6.5.2.4 Veri Kalitesi

Tahmin işleminin mümkün olduğundan ve yapay sinir ağının gerektiği gibi eğitilebileceğinden emin olmak için öncelikle kullanılacak verilerin kalitesinin kontrol edilmesi gerekir. Verilerin kalitesi ile kastedilen, aykırı değerler içermemesi ve rasgele olmamasıdır.

Verilerin kalan kısmından son derece farklı veya uyumsuz olan örnekler, "*aykırı değerler"* olarak adlandırılır. Aykırı değerleri tespit etmek için, veri dağılımının 75. yüzdelik ve 25. yüzdelik değerleri kullanılır. (Yavuz S. , 2006)

Veri kümesini, her alt küme verilerin 25% ve 75%'ini içerecek şekilde, ikiye ayıran değer

Q_1 (alt çeyrek) = 25. yüzdelik,

Q_2 (medyan) = 50. yüzdelik,

Q_3 (üst çeyrek) = 75. yüzdelik olarak adlandırılır.

Veri kümesinde $Q_3 + 3(Q_3 - Q_1)$ değerinden büyük veya $Q_1 - 3(Q_3 - Q_1)$ değerinden küçük olan değerler "*uç aykırı değer"* olarak adlandırılır. Bu çalışmada kullanılan veriler herhangi bir aykırı değer içermemeleri için kontrol edilmiştir.

6.5.2.5 Veri Kümesinin Organizasyonu

Bu çalışmada hem COCOMO 81 hem de YEEM veri kümesi kullanılmıştır. COCOMO 81 ile YEEM'i daha iyi ve gerçekçi karşılaştırabilmek için aynı veri sayısı içinde karşılaştırmak gerekmektedir. Bu yüzden YEEM'i 60 ve 100 adet projeden oluşan iki küme olarak değerlendirmeye aldık. Dolayısıyla toplam 3 adet veri kümesi kullanılmıştır. Her veri kümesini eğitim ve test için iki gruba ayırdık. Bu çalışmada kullanılan veri kümeleri, veri sayısı, eğitim ve test için ayrılan veri sayısı çizelge 6.2'de gösterilmiştir. Üçüncü veri

kümesinin başarısı aslında bu çalışmada ortaya konan modelin gerçek değerini verecektir.

Çizelge 6.2 Veri Kümelerinin Organizasyonu

	Veri Kümesi 1	**Veri Kümesi 2**	**Veri Kümesi 3**
Ölçüt Kümesi	COCOMO	YEEM	YEEM
Veri Sayısı (Proje Sayısı)	60	60	100
Eğitim kümesi	45	45	75
Test kümesi	15	15	25

6.5.3 Uygulama Sonuçları

Bu bölümde uygulama sonuçları, değerlendirme kriteri ve karşılaştırmalar mevcuttur.

6.5.3.1 Değerlendirme Kriteri

Bir modeli test etmek için ve ne kadar iyi çalıştığını ölçmek için hata fonksiyonunu tanımlamak gerekiyor. Literatürde bu konuda az olan çalışmalarda genel kabul görmüş değerlendirme kriteri Ortalama bağıl hatadır (MMRE). Ortalama bağıl hata, bağıl hataların toplamının veri kümesinde bulunan toplam sayıya bölünmesi ile bulunabilir. Eşitlik (6.5)'te gösterilen hesaplamada "N" kullanılan veri kümesinin veri sayısıdır.

$$MMRE = \frac{1}{N}\sum_{i}\frac{\left|Gerçekle\,şen_i - Hesaplanan_{\ i}\right|}{Gerçekle\,şen_i} \tag{6.5}$$

MMRE yeteri kadar iyi olmasına rağmen insan algısı için yüzdesel gösterimide faydalı olacaktır. Bu yüzden MMRE'nın 100 katını da çizelgede göstermekteyiz.

6.5.3.2 Uygulama Sonuçları

3 veri kümesi için hem MLP hem de *Elman* ağları kullanmak suretiyle bu çalışma için 6 değerlendirme yapılmıştır. Çalışma sonuçları çizelge 6.3'te gösterilmiştir.

Çizelge 6.3 COCOMO ve YEEM veri kümeleri ile MLP ve Elman çalışmaları sonuçları

Yapay Sinir Ağı Modeli	Veri Kümesi	Veri Sayısı (Proje Sayısı)	MMRE	Ort. % Hata
MLP	COCOMO	60	1.805	180.50
MLP	YEEM	60	0.280	28.00
MLP	YEEM	100	0.207	20.70
ELMAN	COCOMO	60	1.312	131.20
ELMAN	YEEM	60	0.240	24.00
ELMAN	YEEM	100	0.191	19.10

Elman yapay sinir ağı MLP'ye nazaran daha başarılı olduğu görülmektedir. Bu başarıda *Elman* ağının saklı katmana ek olarak diğer bir "durum" katmanı denilen özel bir saklı katmanın varlığı önemli bir etkendir. Veri kümesi büyüdükçe hata oranının MLP'de %28'den %20.70'e düştüğünü, *Elman*'da %24 hata oranından %19.10'a düştüğünü görmekteyiz. 100 veri kümesinin büyümesiyle daha da başarılı sonuçlara ulaşmak mümkün olabilecektir. YEEM veri kümesinin COCOMO'ya nazaran hem *Elman*'da hem de MLP'de daha başarılı sonuç elde ettiği görülmüştür.

6.5.3.3 Çapraz Onaylama

Hata hesaplama teknikleri genel performansı göstermede yardımcı olmaktadır (Fenton N. ve Pfleeger S., 1997). Bu çalışmada k-kat çapraz onaylama (*k-fold cross validation*) kullanılmıştır. k-kat çapraz onaylama ile elimizdeki veriyi daha iyi kullanmam sağlamaktadır. Çapraz onaylamada veri kümemiz eşit büyüklükte veya yaklaşık eşit büyüklükte k adet alt kümeye bölünür. Ağı k defa eğitilmektedir ve her seferinde farklı bir alt kümeyi eğitim dışında bırakılmalıdır. Her eğitim sonrasında dışarıda bırakılan alt küme hata oranını bulmak için kullanılmaktadır. Dolayısıyla k defa eğitim ve sonrasında hata bulma sonucu elimizde k adet hata mevcut olur. Bu hataların ortalaması bize çapraz onaylama hata oranını vermektedir. Örneğin 15-kat çapraz onaylamada, verinin 1/15'ini ayırıp daha sonra modeli aynı prosedüre göre yeniden inşaası yapılmaktadır ve bu işlem 15 kez yapılmaktadır. Bu çalışmada 5,10 ve 15-kat çapraz onaylama yapılmıştır. Çizelge 6.4'te 5,10 ve 15-kat çapraz onaylama çalışmasının sonuçları verilmiştir (Ayyıldız M.,Kalıpsız O., Yavuz S, 2007).

Çizelge 6.4 MLP kullanılarak Yapılan Çapraz Onayla Sonuçları

	Veri Kümesi 1	Veri Kümesi 2	Veri Kümesi 3
Veri Kümesi	COCOMO	YEEM	YEEM
Veri Sayısı (Proje Sayısı)	60	60	100
Ort.% Hata (5-fold cv)	189.8	21.66	12.89
Ort.% Hata (10-fold cv)	189.0	22.26	12.01
Ort.% Hata (15-fold cv)	166.4	18.97	9.57

Bunun dışında ek olarak 3. veri kümesi (YEEM 100) kulannılarak 20-fold cv yaptık. Bunun sonuçları Çizelge 6.5'te gösterilmiştir.

Çizelge 6.5 20-fold cv MLP kullanılarak Yapılan Çapraz Onayla Sonuçları

Gerçekleşen	Tahmin	Hata
0.098	0.115	0.017
0.263	0.281	0.018
0.062	0.049	-0.013
0.066	0.019	-0.047
0.067	0.07	0.003
0.098	0.107	0.009
0.112	0.132	0.019
0.08	0.119	0.039
0.198	0.215	0.017
0.205	0.18	-0.025
0.07	0.088	0.018
0.413	0.338	-0.076
0.073	0.056	-0.017
0.49	0.534	0.044
0.152	0.137	-0.015
0.057	0.077	0.02
0.219	0.206	-0.013
0.047	0.057	0.01
0.556	0.548	-0.008
0.14	0.111	-0.029
0.096	0.084	-0.012
0.099	0.077	-0.022
0.121	0.125	0.004
0.119	0.107	-0.012
0.276	0.327	0.051
0.239	0.247	0.008
0.072	0.075	0.003
0.075	0.083	0.008

0.142	0.151	0.009
0.063	0.08	0.017
0.073	0.085	0.013
0.264	0.278	0.014
0.061	0.06	-0.001
0.103	0.101	-0.002
0.078	0.069	-0.009
0.055	0.066	0.011
0.06	0.07	0.01
0.136	0.113	-0.023
0.119	0.141	0.021
0.239	0.284	0.045
0.079	0.098	0.019
0.07	0.097	0.026
0.103	0.089	-0.014
0.118	0.092	-0.026
0.128	0.098	-0.03
0.217	0.169	-0.048
0.073	0.092	0.019
0.057	0.082	0.025
0.594	0.606	0.012
0.214	0.183	-0.031
0.102	0.097	-0.004
0.063	0.068	0.004
0.035	0.043	0.009
0.16	0.142	-0.018
0.576	0.644	0.068
0.098	0.102	0.004
0.159	0.173	0.014
0.037	0.043	0.006
0.08	0.09	0.01
0.305	0.298	-0.007
0.094	0.131	0.037
0.066	0.098	0.032
0.115	0.134	0.019
0.067	0.076	0.009
0.188	0.179	-0.009
0.122	0.1	-0.022
0.079	0.069	-0.01
0.148	0.16	0.012
0.091	0.098	0.007
0.326	0.252	-0.074
0.271	0.27	-0.001
0.118	0.097	-0.021
0.154	0.153	-0.001
0.157	0.156	0
0.401	0.332	-0.069
0.075	0.065	-0.01
0.2	0.212	0.012

0.355	0.314	-0.041
0.139	0.145	0.007
0.103	0.089	-0.014
0.198	0.179	-0.019
0.073	0.077	0.004
0.094	0.084	-0.011
0.065	0.059	-0.006
0.183	0.167	-0.016
0.098	0.102	0.004
0.092	0.086	-0.006
0.062	0.057	-0.005
0.128	0.111	-0.017
0.22	0.189	-0.031
0.094	0.09	-0.005
0.072	0.083	0.011
0.335	0.375	0.041
0.404	0.344	-0.059
0.211	0.188	-0.024
0.158	0.156	-0.002
0.072	0.093	0.022
0.602	0.564	-0.038
0.061	0.056	-0.005
0.061	0.063	0.002
0.09	0.076	-0.014
0.231	0.222	-0.009
0.264	0.244	-0.02
0.087	0.116	0.028
0.063	0.085	0.021
0.134	0.123	-0.01
0.114	0.115	0.001
0.155	0.154	-0.001

MLP ile *20-fold* çapraz onaylama kullanılarak yapılan uygulama sonuçları aşağıdaki şekilde yorumlanabilir :

- Bağlantı katsayısı (*Correlation coefficient)* : 0.98
- Mutlak ortalama hata (*Mean absolute error)* : 0.0184
- Hatanın karekökü (*Root mean squared error)* : 0.0245
- Göreli mutlak hata (*Relative absolute error)* : 20.4401 %

0.98 olan bağlantı katsayısı 1'e oldukça yakındır ve doğrusal bir bağlantı olduğu vermektedir. Bu sonuçlara göre 15-fold 20-fold'tan daha iyi bir sonuç vermektedir. Dolayısıyla göreli olarak %20.4 hata mevcuttur. Veri kümesini bu kadar parçalamak zaten iyi olmamaktadır. Bir kırılma noktası mevcuttur. Genelde k=10 ile çalışılmaktadır.

6.6 İlgili Çalışmalar İle Karşılaştırma

Bu çalışmada kullandığımız *Elman* oldukça başarılı sonuçlar vermiştir. Gözlemleyebildiğimiz kadarıyla yapılmış çalışmalar arasında en başarılı olarak görünen Wittig&Finnie çalışması kadar hatta 15-kat çapraz-onaylama ile daha da iyi bir sonuca ulaşılmıştır (Çizelge 6.6).

Çizelge 6.6 Yapay sinir Ağı kullanarak Yazılım Maliyet Tahminlemesi yapma konusunda yapılmış çalışmalar

Çalışma	Kullanılan Algoritma	Ölçüt Kümesi	Proje Sayısı	Sonuç (MMRE)
Wittig & Finnie	Back-Propagation	ASMA ve Desharnais	136 ve 81	%17
Venkatachalm	Back-Propagation	COCOMO	63	
Jorgenson	Back-Propagation	Jorgensen	109	%100
Serluca	Back-Propagation	Mermaid-2	28	%76
Karunanithi	Cascade-Correlation			
Samson	Back-Propagation	COCOMO	63	%428
Srinivasan ve Fisher	Back-Propagation	Kemerer ve COCOMO	78	%70
Hughes	Back-Propagation	Hughes	33	%55
Ayyıldız	Elman	YEEM	100	%9.57

Yapılmış olan diğer çalışmalarda da çok büyük sayıda proje mevcut değildir. Bunun nedenlerinden biri veri toplanmada yaşanan zorluktur. YEEM göreceli olarak küçük denemeyecek sayıda veri kümesine sahiptir. Dolayısıyla ölçüt kümesinin doğruluğu, kapsamı ve iyi bir yapay sinir ağının yanı sıra az olmayacak sayıda gerçek proje verisi de elde edilen başarıda önemli bir etkendir. YEEM veri kümesi yapay sinir ağı olarak yapılan denemeler ve incelemeler kapsamında *Elman*'la en yüksek başarıyı elde ettiği görülmüştür.

6.7 Model Kullanılarak Yapılan Bir Örnek

Eğitim ve test kümesi kullanarak ağımızı test edilmişti (6.5.3.2). Sonuçları belirtilmişti. Bunun yanında çapraz onaylama kullanılarakta test edilmiş ve sonuçlar gösterilmiş oldu. Ancak daha da somutlaştırma adına bir örnek eklemekte fayda vardır. Toplanan 109 proje dışında 110. proje için Çizelge 5.3'te belirtilen aynı veri toplama formu kullanılarak ölçüt değereri toplanmıştır. 110. proje 7 yazılım geliştirme uzmanının, 2 iş analistinin ve 3 test elemanının görev aldığı bir proje şeklindedir. Kapsam değişikliğinin düşük olduğu ancak çalışanların değişim riskinin yüksek olduğu bir projedir. Çalışanların projeye sahiplenmeleri yüksek olan bu projede uygulama karmaşık bir projedir. Bu proje 95 iş gününde

tamamlanmıştır. Projenin ölçütlerinin Çizelge 5.3'de gösterilen veri toplama formu ile toplandıktan sonra eğitilmiş olan *Elman* ağına uyguladığımızda sonuç olarak 0,127317164181600955 değerini vermiştir. Bu değeri denormalize ederek tahmini zamanı bulunmaktadır (6.6).

0,127317164181600955 . 837 = 106,564466419999999335 (6.6)

Bu proje gerçekte 95 iş günü sürmüştür. Dolayısıyla %10.8520 hata oranıyla oluşturmuş oldu.

6.8 Yazılım Geliştirme Projesinin Maddi Maliyetine Geçiş

Yazılım geliştirme maliyetinin temel gideri iş gücüdür. Dolayısıyla projenin ne kadar sürdüğünü, çalışanlar ve katılım oranlarını kullanarak temel maliyeti bulmak mümkündür. 6 adımda maliyete geçmek mümkündür.

1.Yapay sinir ağından çıkan tahmini zaman denormalize edilmelidir.

YSA'da çıkan sonucu (*t*) denormalize edilmelidir. Kullanılan normalizasyon yönteminin tersi uygulanmalıdır. Bu çalışmada kullandığımız normalizasyon yöntemi maksimum değere bölünme idi. Dolayısıyla denormalize etme eşitlik (6.7)'da gösterilmiştir.

Denormalize değeri = $değer_i$. max(değer) (6.7)

Oluşturduğumuz yapay sinir ağında max(değer) 837 idi. Dolayısıyla 837 ile çarpmak denormalize edecektir. Yeni proje için YSA'na uyguladıktan sonra çıkan değere t dersek, ağ sonucu tahmin edilen çalışma günü (6.8)'de hesaplanabilir.

Çalışma Günü (g) = t . 837 (6.8)

2. İş gücü maliyetini bulunmalıdır.

Yapay sinir ağına topladığımız ölçütleri girdi olarak verip tahmini zamanı elde ettikten sonra bunun maddi maliyeti bulunabilinir. Projenin başlangıç aşamalarında sadece atanan personel belli olmayabilmektedir. Projede çalışanlar sadece sayı bazında mevcut ise Çizelge 6.7'de gösterildiği gibi her bir iş tipi için ortalma maliyetten giderek katılım oranınca ve adet'le çarpılarak elde edilebilinir.

Çizelge 6.7 Çalışanlar henüz belli değilken maliyet

	Adet	**Ortalama Maliyeti**	**Katılım Oranı (%)**	**Çalışma Günü**	**Maliyet**
Proje Yöneticisi	x	MP	KP	g	x . MP . KP . g
Yazılım Geliştirici	y	MY	KY	g	y. MY . KY . g
Analist	z	MA	KA	g	z . MA . KA . g
Test Çalışanı	d	MT	KT	g	d . MT . KT . g
….	…	…	…	…	…
Toplam					g . (x.MP.KP + y.MY.KY + z.MA.KA + d.MT.KT +…)

Projede çalışacak personel netleşmişse, her personelin maliyeti elimizde mevcuttur. Dolayısıyla projede görev alan her personelin maliyeti, katılım oranı ve çalışma günü ile çarpılıp her çalışanın maliyeti hesaplanabilinir (Çizelge 6.8). Tüm personellerin maliyetlerinin toplanmasıyla toplam personel maliyetine ulaşılabilinir.

Çizelge 6.8 Çalışanlar belli iken maliyet

	Maliyeti	**Katılım Oranı (%)**	**Çalışma Günü**	**Maliyet**
1. Proje Yöneticisi	MP1	KP1	g	MP1 . KP1 . g
2. Proje Yöneticisi	MP2	KP2	g	MP2 . KP2 . g
…	…	…	…	…
1.Yazılım Geliştirici	MY1	KY1	g	MY1 . KY1 . g
2.Yazılım Geliştirici	MY2	KY2	g	MY2 . KY2 . g
…	…	…	…	…
1.Analist	MA1	KA1	g	MA1 . KA1 . g
2.Analist	MA2	KA2	g	MA2 . KA2 . g
…	…	…	…	…
1.Test Çalışanı	MT1	KT1	g	MT1 . KT1 . g
2.Test Çalışanı	MT2	KT2	g	MT2 . KT2 . g
….	…	…	…	…
Toplam				**Maliyetlerin Toplamı**

3. Donanım giderlerini eklenmelidir.

Çalışan giderlerinin hesaplanmasından sonra yazılım donanım giderleri eklenmelidir. İki tür

donanım kaleminden bahsedilebilinir.

a. Projenin özel donanım giderleri

b. Genel donanımları kullanım giderleri : Kurumda olan genel donanımların ilgili proje için kullanım oranları (6.8)'de gösterilmiştir.

Projenin genel donanım gideri = Genel donanım gideri . Projenin kullanım oranı (6.8)

4.Genel Giderleri eklenmeldir.

Projenin çalışan ve donanım giderleri dışında var olan genel giderler eklenmelidir. Proje için kullanılan ulaşım giderleri , kırtasiye giderleri, eğitim giderleri , danışmanlık giderleri ve bunun dışında var olan genel giderler eklenmelidir.

5. Kurum ortak giderleri eklenmelidir.

Projenin geliştirilmesinin yapıldığı kurumum proje geliştirme amacıyla ek giderleri mevcuttur. Bu giderleri geliştirme yapıldığı kurumun bu giderleri iş gücü orantılı olarak projelere dağıtabiliriz (6.9).

Ortam Gideri = (Lokasyon Kirası/Maliyeti + Elektirik + Telefon + Su + Destek +

Güvenlik + …) x (İş Gücü Maliyeti / Toplam Yazılım İş Gücü Maliyeti) (6.9)

6. Diğer giderleri eklenmelidir.

Projenin geliştirilmesinde iş gücü, donanım, genel ve ortak giderlerin dışında giderleri olacaksa veya olduysa bunlarında maliyete eklenmelidir.

6 adım sonucunda projenin maddi maliyetine geçilmiş olunmaktadır.

7. SONUÇ

Bu çalışmada yeni bir yazılım ölçüt kümesi oluşturma, oluşturulan yazılım ölçüt kümesi için veri toplama ve yeni veri kümesi ile yapay sinir ağı kullanılarak yazılım maliyet tahmin modelinin oluşturulması adımları olarak üç kısımdan oluşmaktadır.

Yazılım maliyet tahminleme çalışmalarında ölçüt kümesi seçiminin hayati bir rolü vardır. Yazılım maliyet tahminleme çalışmalarında ölçüt kümesinin öneminin ihmal edilerek genellikle mevcut kümelerin kullanılmasının, tahmin sonucunu ciddi biçimde etkilediği görülmüştür. Bu tez çalışmasında ile yeni bir ölçüt kümesi (YEEM) oluşturulmuştur. YEEM ölçüt kümesi reel sektörün görüşleri doğrultusunda oluşturulmuştur. Yeni hazırladığımız bu ölçüt kümesi için toplam 28 proje yöneticisinin katıldığı vaka çalışmaları yapılmıştır ve onaylama adımı gerçekleşmiştir. Oluşturulan ölçüt kümesinin mevcut olanlardan farklı olarak barındırdığı bir takım yenilikler mevcuttur. Hiyerarşik bir ölçüt kümesidir. Ana ölçütler, ölçütler ve alt ölçütler olarak 3 katmandır. İşlev puanı bir ölçüt olarak alınarak ve planlar ile tahminler de bir ölçüt olarak alınarak iki defa melezleştirilmiş bir yapı oluşturulmuştur. Planlar ve tahminlerin ölçüt olarak ele alınmasıyla, aynı zamanda bu plan ve tahminlemenin proje üzerinde yarattığı etki de dikkate alınmıştır.

Reel sektörden bu ölçüt kümesine uygun olarak 109 adet yazılım geliştirme projesinden veri toplanmıştır.

Yazılım maliyet tahminleme için yapay sinir ağları kullanımı konusunda literatürde bazı çalışmalar vardır. Bu çalışmalarda genellikle COCOMO 81 veri kümesinin kullanıldığını görülmektedir. Literatürde incelenen çalışmaların sonuçları günümüz için hala kabul edilebilinir, kullanılabilinir seviyelere henüz gelmemiştir.

Karşılaştırma yapabilmek için hem COCOMO 81 hem de yeni oluşturduğumuz YEEM ölçüt kümeleri ayrı ayrı kullanılmıştır. COCOMO 81 ile yaptığımız çalışmalarda yapay sinir ağı kullanarak yazılım maliyet tahminlemesi için bu ölçüt kümesinin yetersiz kaldığı görülmüştür. COCOMO 81 ile bir kaç değişik yapay sinir ağı modeli denememize rağmen sonuç olarak kullanılabilir seviyeye ulaşamamıştır.

Oluşturduğumuz ölçüt kümesi olan YEEM'i uyguladığımızda, kullanılabilinir seviyede iyi sonuçlar alınmaktadır. YEEM ve YEEM için toplanan veriler MLP ve *Elman* yapay sinir ağları kullanılarak test edilmiştir. Yapay sinir ağı kullanılarak yapılan yazılım maliyet tahmini modeli ile bu alanda kullanılmayan *Elman* yapay sinir ağı modeli uygulanmıştır. Bunun yanı sıra başarıyı artırmak için %5, %10 ve %15'lik çapraz onaylamalar kullanılmıştır. Yazılım

maliyet tahminlemede yapay sinir ağı kullanılarak yapılan çalışmalarda böylece yeni olarak *Elman* yapay sinir ağı modelini kullanmanın yanı sıra çapraz onaylama kullanılmıştır.

Elman yapay sinir ağı MLP'ye nazaran daha başarılı olduğu görülmektedir. Bu başarıda *Elman* ağının saklı katmana ek olarak diğer bir "durum" katmanı denilen özel bir saklı katmanın varlığı önemli bir etkendir. Veri kümesi büyüdükçe hata oranının MLP'de %28'den %20.70'e düştüğünü, *Elman*'da %24 hata oranından %19.10'a düştüğünü görmekteyiz. 100 veri kümesinin büyümesiyle daha da başarılı sonuçlara ulaşmak mümkün olabilecektir. YEEM veri kümesinin COCOMO'ya nazaran hem *Elman*'da hem de MLP'de daha başarılı sonuç elde ettiği görülmüştür.

MLP'de çapraz-onaylama (*cross-validation*) yaptığımızda hata oranını %15'in altına düştüğünü görmekteyiz. Değişik boyutta ölçüt kümesi ile çalışıldığından veri sayısının modelin başarısı için önemli bir etken olduğu da görülmektedir. Yapılmış olan diğer çalışmalarda da çok büyük sayıda proje mevcut değildir. Bunun nedenlerinden biri veri toplanmada yaşanan zorluktur. YEEM göreceli olarak küçük denemeyecek sayıda veri kümesine sahiptir. Dolayısıyla ölçüt kümesinin doğruluğu, kapsamı ve iyi bir yapay sinir ağının yanı sıra az olmayacak sayıda gerçek proje verisi de elde edilen başarıda önemli bir etkendir. YEEM veri kümesi yapay sinir ağı olarak yapılan denemeler ve incelemeler kapsamında *Elman*'la en yüksek başarıyı elde ettiği görülmüştür. Dolayısıyla doğru ve nitelikli bir ölçüt kümesi kullanıldığı sürece yapay sinir ağları yazılım maliyet tahminleme çalışmalarında başarıyla kullanılabilinmektedir.

Var olan çalışmaların temel eksikliğin, ölçüt kümesi seçimi olmasının yanı sıra çeşitli yapay sinir ağı modellerinin denenmemiş olması görülmektedir. Bunun yanında ölçüt kümesinin küçük olması ve dengeli olmaması problemleri mevcuttur. Kullanılan proje sayısının az olması başarıyı doğrudan etkileyen bir diğer faktördür.

Bu çalışmada oluşturulan ve kullanılan ölçüt seti için toplanılan yazılım geliştirme verisinin artırılmasıyla daha iyi sonuçlar elde edilebilir. Veri toplamak oldukça zahmetli ve çok zaman alan bir iş olması nedeniyle zor bir noktadır. Üstelik reel sektörde rekabet açısından dezavantaj oluşturma olasılığı nedeniyle birçok kurum bu konuda topladığı verileri eğitim amaçlı da olsa kullanıldırmayı uygun görmemeleri bir diğer zorluk noktasıdır.

Bu çalışmanın devamı olarak oluşturulan ölçüt kümesini başka yöntemler kullanılarak yine yazılım maliyet tahminleme çalışmalarında kullanılması sağlanabilir. Genetik algoritmalar, karar ağaçları ve Vaka bazlı usavurum kullanılarak iyi sonuçlar alınabilir. Genetik

algoritmalar kullanılarak yazılım maliyet tahminleme konusunda yapılan çalışma sayısı yapay sinir ağı kullanılarak yazılım maliyet tahminlemeye oranla daha fazla olmakla beraber gelişme noktaları vardır. YEEM ölçüt kümesini bu alanlarda kullanıp, karşılaştırma çalışmaları yapabilmek mümkündür.

KAYNAKLAR

Abran, A. ve Robillard, P. N. (1994), "Function Points:A Study of Their Measurement Processes and Scale Transformations", The Journal of Systems and Software.

Albrecht, A.J. (1979), "Function-Point Method (Measuring Applications Development Productivity), IBM Application Development Joint Share and Guide Symposium, Monterey.

Ayyıldız, M., Kalıpsız, O., "Yazılım Geliştirme Projelerinde Maliyet Tahminleme Çalışmalarında Kullanılacak Bir Maliyet Tahminleme Çalışmalarında Kullanılacak Bir Ölçev Kümesi ve Bir Yapay Sinir Ağı Topolojisi Önerisi", Ulusal Yazılım Mühendisliği Sempozyumu UYMS 05 22-24 Eylül, 2005, Ankara,Turkiye.

Ayyıldız, M.,Kalıpsız, O., Yavuz, S., "A Metric-Set and Model Suggestion for Better Software Project Cost Estimation" XVI. International Enformatika Conference November 24-26, 2006 Venice, Italy.

Ayyıldız, M.,Kalıpsız, O., Yavuz, S., "YEEM : Yazılım Projeleri Maliyet Tahminleme Ölçev Seti ve Modeli", Eleco 2006, Bursa, Turkiye.

Ayyıldız, M.,Kalıpsız, O., Yavuz, S.,"Software Development Cost Estimation based on Neural Networks", IFSA'2007,Cancun, Mexico, June 18-21, 2007.

Ayyıldız, M.,Kalıpsız, O., Yavuz, S.,"Yazılım Geliştirme Projelerinde Yapay Sinir Ağı Kullanarak Maliyet Tahmini", UYMS'2007, Ankara, Türkiye.

Bailey, C.T. ve Dingee W.L (1981), "A Software Study Using Halstead Metrics", Bell Laboratories Denver.

Ben-Arieh D. (2000), "Parametric Cost Estimation of Design Activities", Department of Industrial and Manufacturing Systems Engineering, Kansas State University.

Boehm, B.W.(1981), Software Engineering Economics, Prentice Hall.

Boehm B., Bradford C., Horowitz E., Madachy R., Shelby R., Westland C. (1995), "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0",Annals of Software Engineering Special Volume, Amsterdam, The Netherlands.

Boetticher G.D. (2003), "Applying Machine Learners to GUI Specification in Formulating Early Cycle Project Estimation".

Briand, L. C., El Eman K., Bomarius F. (1998), "COBRA: A hybrid method for software cost

estimation, benchmarking, and risk assessment", International conference on software engineering.

Briand, L., Morasca S. ve Basili V (2002), "An Operational process for goal-driven definition of measures", IEEE Transactions on Software Engineering 30.

Chidamber, S.R. ve Kemerer C.F. (1994), "A Metrics Suite for Object Oriented Design", IEEE Trans. Software Eng., vol 20, no 6.

Chulani S. (1999), "Bayesian Analysis of Software Cost and Qualıty models" , Doctor of Philosophy Thesis, University of Southern California,.

DeMarco, T. (1982), "Bang Metrics :Controlling Software Projects - Management, Measurement and Estimation", Englewood Cliffs, N.J.

Erdogmus, H. (1999), "Comparative evaluation of software development strategies based on Net Present Value" , Software Engineering Group National Research Council, Canada.

Ferens, D. V., ve R. B. Gumer (1992), "An evaluation of three function point models of estimation of software effort", IEEE National Aerospace and Electronics Conference, vol. 2.

Fenton, N.E. ve Pfleeger S. (1997), Software Metrics: A Rigorous Approach. , Chapman & Hall, London.

Fenton, N.E. ve Pfleeger S. (1998), S.L.,"Software Metrics: A Rigorous and Practical Approach", International Thomson Computer Press.

Gray, A. ve MacDonell S. (1997), "Applications of Fuzzy Logic to Software Metric Models for Development Effort Estimation" ,The North American Fuzzy Information Processing Society.

Gray, A.R. ve S.G. MacDonell (1996), "A Comparison of Alternatives to Regression Analysis as Model Building Techniques to Develop Predictive Equations for SW Metrics", University of Otago.

Heaton J., (2005), "Introduction to Neural Networks with Java", Heaton Research Inc.

Heemstra F. J. (1992), "Software cost estimation", Information and Software Technology, vol. 34, no.10.

Hihn, J. and Habib-Agahi H. (1991), "Cost estimation of software intensive projects: a survey of current practices", International Conference on Software Engineering.

Hughes, R.T. (1996), “An Evaluation of Machine Learning Techniques for Software Effort Estimation”, University of Brighton.

Humphrey, W.S (1989).,"Managing the Software Process", Addison-Wesley Publishing Company.

Idri, A., Khoshgoftaar T.M. ve Abran A. (2002), “Can Neural Networks be easily Interpreted in Software Estimation”,World Congress on Computational Intelligence, Honolulu,Hawaii.

Jeffery, D.R., Low GC ve Barnes M (1993), "A comparison of function point counting techniques", IEEE Trans. Software Eng, 19(5).

Jeffery, D.R. ve G. C., Low (1990), “Calibrating estimation tools for software development”, Software Engineering Journal.

Jones, C. (1996), "Applied Software Measurement", McGraw-Hill.

Jørgensen, M. ve Shepperd M. (2006), A Systematic Review of Software Development Cost Estimation Studies. IEEE Transactions on Software Engineering.

Jorgensen, M. (1995), Experience with the Accuracy of Software Main. Task Effort Prediction Models. IEEE Transactions on Software Engineering, vol 21(8).

Karunanithi, N., D. Whitley, and Y.K. Malaiya (1992), Using Neural Networks in Reliability Prediction. IEEE Software, volume 9.

Kan, S.H. (2002), “Metrics and Models in Software Quality Engineering, Second Edition”, Addison Wesley.

Kemerer, C.F (1987), "An Empirical Validation of Software Cost Estimation Models.", Communications of the ACM 30.

Kemerer, C.F. (1992),"Empirical studies of assumptions that underlie software cost estimation”, Information and Software. Technologies, Volume 34.

Kevin, S., Khaled, E., Madhavji, E. (2000), "Software Cost Estimation with Incloplete Data", National Research Council,Canada.

Kirsten, R. (2001), “Estimating Object-Oriented Software Projects with Use Cases”, University of Oslo, Master of Science Thesis, Oslo, Norway.

Krantz, D., Luce R.D., Suppes P. ve Tversky A. (1971), Foundations of Measurement. Vol. 1.

Academic Press, New York.

Leung, H., Fan, Z. (1981), In Handbook of Software Engineering and Knowledge Engineering , Addison-Wesley.

Lionel, C. B., Khaled E., Morasca S. (1995), "On the Application of Measurement Theory in Software Engineering",International Software Engineering Research Network technical report.

Lionel, C. B., Langley T., Wieczorek I. (2000), "A replicated Assessment and Comparison of Common Software Cost Modelling Techniques", 22nd International Conference on Software Engineering.

Lorenz, M. ve Kidd J. (1994), "Object-Oriented Software Metrics", Prentice Hall, New Jersey.

McCabe T.J. (1976) , "Complexity Measure", IEEE Transacions on Software Engineering, Volume 2, No 4,.

Miyazaki, Y., and Mori K. (1985), "COCOMO evaluation and tailoring", Eighth Int. Conf. Software Engineering.

Morris, K.L. (1989), "Metrics for Object-Oriented Software Development Environments." Master Thesis, M. I. T. School of Management.

Munson J.C. (2003), Software Engineering Measurement, Auerbach Publication.

Park, R. (1998), "Software Size Measurement: A Framework for Counting Source Lines of Code" Software Engineering Institute Technical Report.

Parr, F.N (1980), Parr Model :An Alternative to Rayleigh Norden Curve Model for Software Development Effort. IEEE Transactions on Software Engineering, SE-6, No. 3.

Pfleeger, S.L.; Fitzgerald, J.C.(1989), Pfleeger Model Software Metrics Tool Kit: Support for Selection, Collection and Analysis. Information and Software Technology, Volume 33, No. 7.

Pillai, K. and Sukumaran Nair, V.S. (1997), A Model for Software Development Effort and Cost Estimation. IEEE Transactions on Software Engineering 23.

Poels, G. ve Dedene, G. (2000), "Distance-based software measurement: necessary ve sufficient properties for software measures", Information and Software Technology: 42(1).

Pressman, R.S. (1997), “Software Engineering A Practitioner’s Approach”, McGraw-Hill.

Pressman, R.S. (2005), “Software Engineering A Practitioner’s Approach”, 6th Edition, McGraw-Hill International.

Putman, W.L., (1978), "A general empirical solution to the macro software sizing and estimating problem.", IEEE Trans. on Softw. Eng., Volume 4, No 4.

Pyle, D. (1999), Data Preparation for Data Mining, Morgan Kaufmann.

Reed, R. D. ve Marks, R. J. (1999), Supervised Learning in Feedforward Artificial Neural Networks, MIT Press.

Samaraweera, L.G. (1996), "A review of Approaches to the Measurement of Software Enginering", Dept. of Electronics and Computer Science, Univercity of Southampton.

Samson, B., Ellison,D., Dugard P. (1993), “Software Cost Estimation using an Albus Perceptron (CMAC)”, In Proc Eight International COCOMO Est. Meeting, Pittsburgh.

Serluca, C. (1995)., An Investigation Into Software Effort Estimation using a Back-Propogation Neural Network, M.Sc. Thesis, Bournemouth University.

Selby, R. (1988), “Empirically Analyzing Software Reuse in a Production Environment”, Software Reuse: Emerging Technology, W. Tracz , Computer Society Press.

Shepperd, M.and C. Schofield (1997), “Estimating software project effort using analogy”, IEEE Trans. Soft. Eng. SE-23,12.

Sommerville, I. (1992), "Software Engineering", Addison Wesley Publishing Company, Workingham, England.

Cockcroft, S. (1996), "Estimating CASE development size from outline specifications" University of Otago, Information and Software Technology

Srinivasan, K. ve D. Fisher (1995), Machine Learning Approaches to Estimating Software Dev. Effort. IEEE Transactions on Software Engineering, vol 21(2).

SWEBOK (2001), Guide to the Software Engineering Body of Knowledge, IEEE.

Symons, Charles, R (1988) ,MARK II Function Points.:Function Point Analysis: Difficulties and Improvements. IEEE Transactions on Software Engineering,Volume 14.

Tausworthe, R.C.(1981) : SOFTCOST JPL 1981 (Deep Space Network Software Cost

Estimation Model. Publication 81-7, Jet Propulsion Library, Pasadena,Canada.

The Standish Group (1995), CHAOS Chronicles, Standish Group Int. Report., USA

Tsuneo, Y., Tohru K. (1999), "A Framework for Top-down Cost Estimation of Software Development", Department of Informatics and Mathematical Sciences, Graduate School of Engineering Science, Osaka University, Japonya.

Turner C.R. (1999), "Feature Engineering of Software Systems", Doctor of Philosophy Thesis, Univeristy of Colorado.

Venkatachalam, A.R. (1997), "Software Cost Estimation Using Artificial Neural Networks", International Joint Conference on Neural Networks, Nagoya.

Vicinanza, S. S., Mukhopadhyay T., ve Prietula M. J. (1991), "Software-effort estimation: an exploratory study of expert performance", Information Systems Research, vol. 2, no. 4.

Yavuz, S.(2006) "Performans Artırmaya Yönelik Paralel Mimarilerin Yapay Sinir Ağları Yaklaşımı İle Değerlendirilmesi", Doktora Tezi, İstanbul.

Walston, C. E.; Felix, C.P. (1977), A Method of Programming Measurement and Estimation, IBM Systems Journal, Vol. 16, No. 1.

Weyuker, E.J. (1988), "Evaluating Software Complexity Measures", IEEE Transactions on Software Engineering. 14(9).

Whitmire, S. (1997), "Object Oriented Design Measurement", John Wiley and Sons. Inc.

Wittig, G. and G. Finnie (1997), "Estimating Software Development Effort with Connectionist Models", Information and Software Technology, vol 39.

Wolverton, R.W. (1974) , "The Cost of Developing Large-Scale Software", IEEE Transactionson Computer, Volume C-23.

ÖZGEÇMİŞ

Doğum tarihi	05.07.1976	
Doğum yeri	İstanbul	
Lise	1989-1992	İzmir Karşıyaka Gazi Lisesi
Lisans	1992-1997	Marmara Üniversitesi, Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü
Yüksek Lisans	1997-2001	Marmara Üniversitesi, Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı
Doktora	2001-2007	Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı