T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Use-Case Tabanlı Yazılım Emek Kestirim Modeli

Fatih YÜCALAR

Doktora Tezi

Bilgisayar Mühendisliği Anabilim Dalı
I. Danışman: Prof. Dr. Fuat İNCE
II. Danışman: Yrd. Doç. Dr. Erdem UÇAR
2011 – EDİRNE

T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

USE-CASE TABANLI YAZILIM EMEK KESTİRİM MODELİ

Fatih YÜCALAR

DOKTORA TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 01 / 07 / 2011 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

Prof. Dr. Fuat İNCE
(I. Danışman)

Yrd. Doç. Dr. Erdem UÇAR
(II. Danışman)

Prof. Dr. Şaban EREN
(Üye)

Doç. Dr. Yılmaz KILIÇASLAN
(Üye)

Doç. Dr. Tahir ALTINBALIK
(Üye)

Yrd. Doç. Dr. Aydın CARUS
(Üye)

Doktora Tezi

Trakya Üniversitesi

Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

ÖZET

USE-CASE TABANLI YAZILIM EMEK KESTİRİM MODELİ

Başarılı bir yazılım projesinin temel hedefi, müşteri gereksinimlerine ilişkin beklentileri karşılayan, önceden belirlenmiş bütçe ve zaman içinde tamamlanan bir yazılım projesi geliştirmektir. Ne yazık ki, birçok durumda bu üç hedefi bir araya getirmek mümkün olmamaktadır. Bu başarı kıstaslarını sağlamak için, yazılım projelerinde yazılım ölçüm yöntemleri kullanılmaktadır.

Bir yazılım projesinin bütçesini ve süresini proje tamamlanmadan belirleyebilmek için harcanacak emeğin tahmin edilmesi gerekmektedir. Yazılım emek kestirimi, başarılı bir proje yönetiminin kilit unsurlarından biridir. Emeği tahmin etmek önemlidir çünkü bir yazılım projesine fazladan insan atamak gelir kaybına yol açabileceği gibi, gereğinden az insan atamak yazılım ürününün bitirilmesinde gecikmeye yol açacaktır. Zaman planı ve bütçeyi dengelemek için emek değerinin önceden belirlenmesi gerekir.

Harcanan emeği tahmin edebilmek için ise projenin büyüklüğü kestirilmelidir. Ne yazık ki, proje yöneticileri, yazılım büyüklüğü ve emek ile ilgili tahminleri doğru yapamamaktadırlar. Proje yöneticileri genelde gerekli emeği tahmin etmek için uzman yargısını kullanırlar, ancak tatmin edici sonuçlar almaktan uzaktadırlar.

Yazılım projelerinin büyüklüğünü ve iş gücünü tahmin etmek için birçok metot bulunmaktadır. Bu yöntemlerden en fazla bilinen ve yaygın olarak kullanılanları; İşlev Puanı Analizi, COSMIC Tam İşlev Puanı ve Geliştirici Maliyet Modeli II'dir. Ancak bu yöntemler birtakım ciddi kısıtlamalara sahiptir. Örneğin, işlev puanlarının sayılması uzman gerektirir. Bu metotların dışında özellikle nesne-tabanlı yazılım geliştiren organizasyonların kullandığı Use-Case Puanı (UCP) yöntemi vardır. UCP yöntemi, bir projenin teknik ve çevresel karmaşıklığına ilişkin iki ayarlama faktörü ile use-case modelini temel alan bir yazılım proje emek kestirim tekniğidir.

Bu çalışmanın amacı bütçe ve süre aşımlarına bağlı problemlerin üstesinden gelmek için yeni bir emek kestirim yöntemi ortaya koymak ve yazılım emek tahmininin bir analizini gerçekleştirmektir. Bu bağlamda, UCP yöntemi ayrıntılı olarak ele alınmıştır. UCP yönteminden yola çıkarak, yazılım emek kestirimi için çoklu doğrusal regresyon analizi tabanlı yeni bir emek kestirim yöntemi ortaya konulmuştur. Önerilen çözüm hem yazılım emek kestirim yöntemlerine farklı bir bakış açısı getirmekte hem de yazılım emek kestirim sürecini geliştirmeye çalışmaktadır. Bu yöntem Türkiye'deki dört farklı yazılım firmasından toplanmış on yazılım projesi üzerinde denenmiştir ve elde edilen sonuçlar oldukça başarılıdır.

Bu tez 2011 yılında yapılmıştır ve 118 sayfadan oluşmaktadır.

Anahtar Kelimeler: Yazılım büyüklük kestirimi, yazılım emek kestirimi, use-case puanı yöntemi, yazılım kestirim teknikleri.

Doctorate Thesis

Trakya University

Graduate School of Natural and Applied Sciences

Department of Computer Engineering

ABSTRACT

USE-CASE BASED SOFTWARE EFFORT ESTIMATION MODEL

The main objectives of a successful software project are developing it to meet the expectations of the customer's needs, completing it within a planned time and within the planned budget. Unfortunately, it is impossible to fulfill all of these three objectives in most cases. To ensure the success criteria, software measurement methodologies have been used in software projects.

It is necessary to estimate the effort needed to determine the cost and the time of a software project before it is completed. Software effort estimation is one of key elements of project management. Estimating effort is crucial since hiring more people than actually needed leads to loss of income and likewise hiring less people than actually needed leads to delay in software product delivery. To balance schedule and budget, the effort needs to be correctly predetermined.

Project size should be estimated to determine the effort. Unfortunately, project managers are not very successful at predicting the related to software size and effort. Software managers usually use expert judgment to estimate the required effort; however, they are far from getting satisfactory results.

There are many of methods to estimate project size and effort. The most common and known methods are Function Point Analysis (FPA), COSMIC Full Function Point and Cost Constructive Model II (COCOMO II). But these methods have some serious limitations. For example, counting function points requires experts. Apart from these methods there is Use-Case Points (UCP) method. The UCP method is a software project effort estimation technique based on a use-case model and two sets of adjustment factors related to the environmental and technical complexity of a project.

The objectives of this study are making an analysis of software effort estimation techniques and presenting a new software effort estimation method to overcome problems related to budget and time overruns. In this context, UCP method is discussed in detail. Using the UCP method as a base, new effort estimation method based on multiple linear regression analysis is proposed for software effort estimation. Our proposed method does not only bring another point of view into software effort estimation methods but it also tries to improve the software effort estimation process. This method have been tried on 10 software projects which were collected from four different software development companies located in Turkey and the obtained results are quite successful.

This thesis has been completed in 2011 and consists of 118 pages.

Keywords: Software size estimation, software effort estimation, use-case point method, software estimation techniques.

ÖNSÖZ

Bu tez çalışması süresince bana yol gösteren, destek ve yardımlarını esirgemeyen danışman hocam Sn. Prof. Dr. Fuat İNCE'ye, çalışmanın başlangıcında ve sonra ki her aşamasında destek veren ikinci danışman hocam Sn. Yrd. Doç. Dr. Erdem UÇAR'a, tez izleme komitesinde yer alan ve bu süreçte bilgilerinden faydalandığım Sn. Doç. Dr. Yılmaz KILIÇASLAN'a, her tez izlemede vermiş olduğu destekten dolayı Sn. Doç. Dr. Tahir ALTINBALIK'a, tez çalışmam sırasında bana sağladığı imkânlar ve huzurlu bir aile ortamı için Trakya Üniversitesi Bilgisayar Mühendisliği Bölümü'ne, tezi tamamlamam için bana gereken fırsatı ve motivasyonu sağlayan Maltepe Üniversitesi Mühendislik Fakültesi Dekanı Sn. Prof. Dr. Murat TAYLI'ya, çalışma hayatımda bana destek olan ve yardımlarını esirgemeyen Maltepe Üniversitesi Bilgisayar Mühendisliği Bölümü hocalarıma ve çalışma arkadaşlarıma, akademik çalışmalara başlamaya karar vermemde ve sonrasında desteklerini hiçbir zaman esirgemeyen Yaşar Üniversite Fen-Edebiyat Fakültesi Dekanı Sn. Prof. Dr. Şaban EREN'e ve aileme teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER DİZİNİ

Kısaltma	İngilizcesi	Türkçesi
AF	Adjustment Factor	Ayarlama Faktörü
AFPs	Adjusted Function Points	Düzeltilmiş İşlev Puanı
API	Application Programming Interface	Uygulama Programı Arayüzü
Cfsu	COSMIC functional size unit	COSMIC işlevsel büyüklük birimi
COCOMO 81	Constructive Cost Model 81	Geliştirici Maliyet Modeli 81
COCOMO II	Constructive Cost Model II	Geliştirici Maliyet Modeli II
COSMIC	Common Software Measurement International Consortium	Uluslararası Ortak Yazılım Ölçüm Birliği
COSMIC FFP	COSMIC Full Function Points	COSMIC Tam İşlev Puanı
CP	Class Points	Sınıf Puanı
ÇKF	Environmental Complexity Factor	Çevresel Karmaşıklık Faktörü
DAA	Unadjusted Actor Weights	Düzeltilmemiş Aktör Ağırlığı
DSI	Delivered Source Instructions	Teslim Edilmiş Kaynak Komutlar
DUCA	Unadjusted Use-Case Weights	Düzeltilmemiş Use-Case Ağırlığı
DUCP	Unadjusted Use-Case Points	Düzeltilmemiş Use-Case Puanı
EAK	Effort Adjustment Factor	Emek Ayarlama Katsayısı
FFP	Full Function Points	Tam İşlev Puanı
FP	Function Points	İşlev Puanı
FPA	Function Point Analysis	İşlev Puanı Analizi
FSM	Functional Size Measurement	İşlevsel Büyüklük Ölçümü
GUI	Graphical User Interface	Grafik Kullanıcı Arayüzü
IFPUG	International Function Point Users Group	Uluslar Arası İşlev Noktası Kullanıcıları Grubu

KDSI	Thousands of Delivered Source Instruction	Teslim Edilmiş Bin Satırlık Kaynak Komut
LOC	Lines of Code	Kod Satır Sayısı
MIS	Management Information Systems	Yönetim Bilgi Sistemleri
MK II FPA	Mark II Function Points Analysis	Mark II İşlev Puanı Analizi
MMRE	Mean of Magnitude of Relative Error	Göreceli Hata Büyüklüğü Ortalaması
MRE	Magnitude of Relative Error	Göreceli Hatanın Büyüklüğü
MSE	Mean Squared Error	Hata Kareleri Ortalaması
NEFPUG	Netherlands Function Point Users Group	Hollanda İşlev Puanı Kullanıcıları Grubu
NESMA	Netherlands Software Metrics Users Association	Hollanda Yazılım Ölçüt Kullanıcıları Birliği
OOFPs	Object-Oriented Function Points	Nesne-Tabanlı İşlev Puanı
OOmFP	Object-Oriented Method Function Points	Nesne-Tabanlı Yöntem İşlev Puanı
RMSE	Root Mean Squared Error	Kök Hata Kareler Ortalaması
SLOC	Source Lines of Code	Kaynak Kod Satır Sayısı
sÜF	Post Productivity Factor	Sonraki Üretkenlik Faktörü
TDI	Total Degree of Influence	Toplam Etki Derecesi
TCF	Technical Complexity Factor	Teknik Karmaşıklık Faktörü
TUCP	Total Unadjusted Class Point	Toplam Düzeltilmemiş Sınıf Puanı
UCP	Use-Case Points	Use-Case Puanı
UFPs	Unadjusted Function Points	Düzeltilmemiş İşlev Puanı
UML	Unified Modeling Language	Birleşik Modelleme Dili
ÜF	Productivity Factor	Üretkenlik Faktörü

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Yazılım soyut bir üründür. Birçok bilinmeyeni içermesinden dolayı yazılım geliştirme süreci hem zordur hem de zaman alır. Bilinen bir yazılım geliştirme yöntemi kullanılmış olsa bile, iyi tanımlanmış bir yazılım uygulamasının geliştirme maliyetini ve emeğini tahmin etmek kolay değildir. Yazılım projelerinde maliyet ve emek tahminleri, geliştirilecek sistemin büyüklüğünün tahmin edilmesini temel almaktadır. Geliştirilecek sistemin ilk aşamalarında yeterince bilgi olmaması nedeniyle güvenilir tahminler elde etmek zordur. Geliştirme ekibinin bilgi ve deneyimi, geliştirme süreci ile ilişkili riskler ve sistemle ilgili işlevselliğin belirlenmesi gibi faktörler güvenilir tahminlerin elde edilmesini etkilemektedir. Belki de bu faktörler içerisinde en önemlisi sistemle ilgili işlevselliğin belirlenmesidir. Genellikle yazılım geliştiriciler, bir yazılım sisteminin büyüklüğünü sezgiye ve deneyime dayalı olarak tahmin ederler.

Yazılım büyüklük ölçütleri, yazılım geliştirme emek tahmini için temel girdi olarak ortaya çıkmıştır. Yazılım geliştirme emeği, genellikle adam-saat olarak ölçülen ve yazılım büyüklüğü ile anlamlı bir korelasyonun elde edilmesidir. Geleneksel maliyet modelleri, girdi parametresi olarak yazılım büyüklüğünü alır ve ardından toplam emeği hesaplamak için bunu ayarlama faktörleri veya maliyet etmenleri kümesine uygular.

İlk önemli büyüklük ölçütü Kaynak Kod Satır Sayısı (Source Lines of Code – SLOC)'dır. Satır-tabanlı veya ifade-tabanlı olmak üzere SLOC'u saymanın birkaç farklı yolu vardır. SLOC fiziksel bir büyüklük ölçümü olarak düşünülebilir. Çünkü SLOC bir yazılım sistemi ile ilişkili kaynak kodun fiziksel hacmini ölçmektedir. SLOC ölçümü, birçok bağlamda yararlıdır ve diğer ölçümlerin de belirlenmesini sağlamaktadır.

Diğer ölçüm yöntemleri, yazılım sisteminin fiziksel büyüklüğüne karşı kullanıcıya teslim edilen işlevselliği ölçmeyi araştırmaktadır. Bu ölçüm yöntemleri işlevsel büyüklük ölçümleri olarak adlandırılmaktadır. SLOC'u tahmin etmek zordur. İşlevsel büyüklük, proje ile ilgili tahminleri erkenden elde etmek için kullanmaktadır.

İşlevsel büyüklük yöntemlerinden en önemlisi Alan Albrect tarafından 1979'da ortaya atılan İşlev Puanı (Function Points - FP) olmuştur (Fetcke, Abran, & Dumke, 2001). Bu yöntem girdilerin sayısı, çıktıların sayısı, sorguların sayısı, iç mantıksal dosyaların sayısı ve dış mantıksal dosyaların sayısı olmak üzere beş parametre kullanmaktadır. İşlev puanı, tamamlanmış ürün içerisinde kullanılan verinin büyüklüğünü ve etkileşimlerinin sayısını temel almaktadır. Sonraki yıllarda İşlev Puanı Analizi (Function Point Analysis), Mark II, Tam İşlev Puanı (Full Function Points – FFP) ve COSMIC – FFP gibi diğer işlevsel büyüklük yöntemleri önerilmiştir. Bu yöntemlerin yazılım endüstrisindeki kullanımı bir dereceye kadar başarılı olmuştur. Bu yaklaşımlar birtakım ciddi kısıtları beraberinde getirmiştir. İşlev puanlarını sayma işleminin uzman kişileri gerektirmesi bu kısıtlara örnek olarak verilebilir.

1993 yılında Gustav Karner, az çok İşlev Puanı kavramına benzer olan ama use-case'leri temel alan Use-Case Puanı (Use Case Point – UCP) kavramını ortaya atmıştır (Kusumoto, Matukawa, Inoue, & Hanabusa, 2004). Use-case puanı, İşlev Puanı Analizi ve Mark II İşlev Puanı Analizi'nin bir uzantısıdır ve bu yöntemler gibi aynı felsefeyi temel almaktadır (Ochodek, Nawrocki, & Kwarciak, 2011). Use-case puanı, nesne-tabanlı yöntemler kullanılarak geliştirilecek projelerin büyüklüğünü tahmin etmek için kullanılabilmektedir.

Nesne-tabanlı yazılım ürünlerinin modellenmesi ve tasarımı için en yaygın kullanılan yöntem UML (Unified Modeling Language)'dir (Schach, 2007). UML hedef sistemin farklı bakış açılarını sunan çeşitli diyagramlar aracılığıyla bir sisteme genel bakış ve iyi yapılandırılmış bir mimari sağlar. UML bir mimari tanımlama dili olmamasına rağmen, çeşitli UML diyagramlarının kullanımı ile yazılım sistemlerinin büyüklüğünü ve karmaşıklığını ölçmek için yararlı bilgilerin elde edilmesini sağlamaktadır. UML diyagramlarından yararlı bilgileri elde etme, üst seviyeli yazılım geliştirmede dilden bağımsız bir ölçüm avantajı sağlamaktadır.

Günümüzde UML kavramının ortaya çıkması ile yazılım büyüklüğünün belirlenmesi için use-case'lere başvurulması ve böylece emek tahminlerinin gerçeğe daha yakın hesap edildiği görülmektedir. Ancak, Karner'ın yöntemi yani use-case puanı, use-case'ler ve aktörler arasındaki ilişkiler gibi bir takım uygulama alanı

ayrıntılarını dikkate almamaktadır. Bu ayrıntılar göz önünde bulundurularak, use-case puanı yönteminin iyileştirilmesine ilişkin çalışmalar yapılmaktadır.

Tez çalışmasının, 2. Bölümü'nde yazılım proje yönetiminde çok önemli olan ölçme kavramı ve bu kavram çerçevesinde yer alan temel dayanaklara değinilmiştir. 3. Bölüm'de yazılım büyüklük kestirimi konusu ele alınmış ve yazılım büyüklük kestiriminde kullanılan yöntemler incelenmiştir. Tez çalışmasının 4. Bölümü'nde yazılım emek kestirimine yer verilmiştir. Yazılım emek kestirimi, algoritmik ve algoritmik olmayan kestirim yöntemleri olmak üzere iki şekilde sınıflandırılmış ve incelenmiştir. Özellikle algoritmik emek kestirim yöntemlerinden biri olan Use-Case Puanı yöntemi ayrıntılı olarak ele alınmıştır. Bu yöntemle ilgili olarak yapılan çalışmalar incelenmiş ve değerlendirilmiştir. 5. Bölüm'de ise Use-Case Puanı yönteminden yola çıkarak, yazılım emek kestirimi için doğrusal regresyon analizi tabanlı yeni bir emek kestirim yöntemi ortaya konulmuştur. Ortaya konan bu yeni emek kestirim yöntemi gerçek yazılım projelerinden elde edilen bir veri seti üzerinde doğrulanmıştır. Son olarak, 6. Bölüm'de ise sunulan çalışma özetlenmiş ve elde edilen sonuçlar değerlendirilmiştir.

2. YAZILIMDA ÖLÇME

Her yazılım projesinin temel hedefi ve başarılı olabilmesinin ardındaki temel gereksinim, önceden belirlenen zamanda, önceden belirlenen bütçe ile müşterinin beklentilerini karşılayacak özelliklerde bir yazılım projesi üretmektir. Ne yazık ki yazılım projelerinde bu üç koşulun bir arada sağlanması genelde mümkün olmamaktadır. Yazılım projelerinde bu başarı kriterlerinin sağlanması için yazılımda ölçüm yöntemlerinin kullanılması, yazılım sektöründe gittikçe önem kazanır olmuştur. Yazılım organizasyonları üç ana amaçla yazılımda ölçümü gündemlerine almaktadırlar:

- Yazılım projesini anlamak ve modellemek,
- Yazılım projelerinin yönetilmesine yol göstermek,
- Yazılım süreç geliştirme ve iyileştirme çalışmalarını sürdürmek,

Yazılımın ölçülebilmesi, harcanılan zaman, emek, proje büyüklüğü ve kalite gibi faktörlerin belirlenmesine olanak sağlayarak yazılım organizasyonlarının var olan performansını belirlemektedir. Bu verilerin temel hedefi, organizasyonların ilerisi için yapmış olduğu öngörüleri kaba tahmine bırakmadan yapabilmesini sağlamaktır. Yazılım organizasyonları, geliştirdikleri yazılım projeleri içerisinde çeşitli verileri toplayarak bir tarihçe birikimi sağlamalıdırlar. Çünkü bugünün verileri yarının verileri olacaktır. Organizasyonlar bu verilere dayanarak ileride alacakları projeler için büyüklük, emek ve maliyet kestirimlerini yapabilme imkânı bulabileceklerdir.

Yazılım proje yönetiminde çok önemli olan ölçme ve bu kavram çerçevesinde yapılanan kestirim yöntemleri aracılığı ile zaman ve işgücü gibi planlamalar yapılabilmektedir. Yazılım projelerinde kaliteyi arttırmak, her şeyden önce doğru ölçme yöntemlerinin kullanımına bağlıdır. Bu yöntemler, doğrudan ve dolaylı ölçülebilen büyüklükler olarak sınıflandırılmaktadır. Doğrudan ölçülebilen büyüklükler yazılım maliyeti, ortaya çıkan hata sayısı, yazılımcı emeği, yazılan satır/kod sayısı, yazılımcı adam/saat olarak sınıflandırılabilir. Dolaylı olarak ölçülebilen büyüklükler ise, yazılımın işlevselliği, güvenilirliği, bakım kolaylığı ve yazılımın kalitesi şeklinde

sınıflandırılabilir. Bu tip sınıflandırmalar yazılım projelerinde genel bir kural olarak benimsenmeyebilir. Bunun temel nedeni yazılım projeleri aynı olsa bile çalıştırılan kodların sayılarına göre değişiklik gösterebilecektir.

2.1. Ölçmenin Temel Dayanakları

Kolaylığı ve doğrudan ölçülebilirliği açısından en fazla kullanılan yazılım ölçme temeli, Satır Sayısı (Lines of Code - LOC)'dır (Ayyıldız, 2007). Bir programın büyüklüğü denince ilk akla gelen kaç satırlık kaynak kodu ile üretildiğidir. Yazılacak bir programın değerini saptamak için, programın bir biçimde ölçülmesi gerekir. Programı, değişik programcılar farklı büyüklükte kodlarla gerçekleştirebilirler. 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 (function point)'*, satır sayısına karşı bir seçenek olarak karşımıza çıkmış bir yazılım ölçüsü birimidir. İşlev puanı, satır sayısı gibi açıkça tanımlanmış bir doğrudan ölçüm birimi değilse de, yazılımı değerlendirmek için yaygın bir şekilde kullanılmaktadır (Ayyıldız, 2007).

Çizelge 2-1'de bir yazılım organizasyonu içerisinde üretilmiş yazılım projelerinin satır sayısına dayalı ölçümleri gösterilmektedir.

Çizelge 2-1. Üç Proje İçin Yapılan Satır Sayısına Dayalı Ölçmeler

Proje	Satır Sayısı	Emek	Personel	Maliyet	Hata	Sayfa Sayısı
P1	12100	24 adam-ay	3	168.000 $	134	365
P2	27200	62 adam-ay	5	440.000 $	321	1224
P3	20200	43 adam-ay	6	314.000 $	256	1050

Yazılım organizasyonları daha önce karşılaştıkları problemleri ve durumları kayıt altında tutarak bir ölçüm veri tabanı oluştururlar. Bu ölçüm veri tabanında geçmiş

projelerden elde etmiş oldukları verileri saklarlar. Ölçüm veri tabanında saklamış oldukları bu verileri göz önünde bulundurarak geliştirecekleri yeni yazılımlar için bir tahminde bulunabilirler. Daha önceden geliştirilen yazılımlardan elde edilen bilgiler, yeni geliştirilecek yazılımlar için bir emsal teşkil etmemesine rağmen bir fikir verme açısından başvurulacak bilgilerdir.

Çizelge 2-1'deki P1 projesini ele alırsak, 168.000 $'a mal olduğunu, 3 kişi tarafından 12100 satır büyüklüğünde geliştirildiğini görmekteyiz. Emeğin 24 adam-ay olması, 3 kişi ile projenin 8 ayda tamamlandığını gösterir. Hatalar, projenin geliştirmesi sırasında oluşur. Bozukluklar ise, projenin müşteriye tesliminden sonra ortaya çıkan hatalardır. Bu örnekteki diğer dolaylı ölçümlerden bazıları:

- Personel Maliyeti: (168000/24) = 7000 dolar / ay (adam başına)
- Program satırı maliyeti: (168000/12100) = 14 dolar
- Hata Oranı: (134/12100*1000) = 11 hata / 1000 satır
- Sayfa Sayısı: (365/12100*1000) = 30 sayfa /1000 satır

Bunların yanı sıra, aşağıdaki ilginç diğer ölçümlere de varılabilir. Daha geçerli olan 1000 satır anlamındaki KLOC birimi, yazılım ölçümlerinde bir standarttır. Bir başka önemli nokta da verilen bilgilerin sadece programlama değil, projenin analiz, tasarım, bakım gibi bütünü için hesaplandığıdır. Hata/Adam-Ay, LOC/Adam-Ay, Dolar/Doküman Sayfası diğer ölçümlere örnek olarak verilebilir.

İşlev puanı da, sayısal olarak verilerin gösterilmesini sağlamaktadır. Sayısal olarak yazılımın değerlendirilmesini sağlayan bu yöntemde, istenilen bileşenler için değişik ağırlıklar verilerek yazılıma katkısı araştırılabilir. Bu tip değerlendirme sistemlerinde ağırlık ve sonuç göstergeleri yazılım yöneticisi tarafından değişik şekillerde bir araya getirilerek sayısal göstergeler oluşturulabilir. Bu göstergelerde bulunan bileşenler yer değiştirilerek gösterge üzerinde anlamlı ifadeler çıkarılabilir. Bunu bir örnek üzerinden açıklamak gerekir ise, FORTRAN programlama dili ile yazılan bir program yerine C++ programlama dili kullanılarak bazı kodların fazladan yazılması engellenebilir. Nesne yönelimli dillerden birisi ile oluşturulan program yerine yapısal programlama dillerinden birisi ile yapılan programlarda daha fazla kod yazmak

maliyetleri ve hata oranını arttırabilmektedir. Bunun için işlevsel göstergelerin neler olduğunu veya nasıl hesaplanması gerektiğinin iyi belirlenmesi gerekmektedir. Belirleme işlemleri sırasında eş değer bilgisayar yazılım dilleri tablosu oluşturularak, programlama dilinin temel fonksiyonlarından birinin kaç satır kod ile yazılabildiğinin ortaya konulması gerekir. Bunun için temel olarak belirlenen matematiksel veya mantıksal birkaç genel kabul görmüş algoritma, seçilen programlama dili ile programlanır ve kaç satır kod ile oluştuğu belirlenir. Bu bize bir fonksiyon veya alt programın hangi programlama dilinde kaç satır koda (KLOC) karşılık geldiğini belirlemiş olur. Satır kod sayısı işlemine programlama dillerinin işlevselliğini de katacak olursak, yapılması gereken *Satır Sayısı (LOC) / İşlevsel Gösterge* oranını bulmaktır (Macit, 2006).

Bu oransal değer bize satır kod sayısının ne kadar işlevsel olduğu hakkında oransal bilgi verecektir. Oransal değer dönüşüm tablosu oluşturmak, işlemleri daha kolaylaştıracak ve yazılım kodları konusunda daha açıklayıcı bilgi edinmemizi sağlayacaktır.

Çizelge 2-2. Örnek Kod Satır Sayısı (LOC) / İşlevsel Gösterge Dönüşümü Tablosu

Programlama Dili	**LOC / İşlevsel Gösterge (ψ)**	**Ağırlıklı Sonuç Değer. Oranı (λ)**
C++	400	15
Turbo Pascal 7.0	350	18
Fortran 77	310	23
gcc	700	11
g++	770	10

Kaynak: (Macit, 2006)

Çizelge 2-2'de verilen örnek göz bulundurularak, ağırlıklı sonuç değerlendirme indeksi oluşturulur. Bu indeks, yazılımı geliştiren takımın başarı oranı olarak kabul edilir. Ağırlıklı sonuç değerlendirme oranı (λ), işlevsel göstergeler ile sonucu etkileyen faktördür.

Sonucu etkileyen faktörler ve bunların ağırlık değerleri indeksi Çizelge 2-3'te verilmektedir. Sonucu etkileyen her bir faktör, sonuç indeksini etkileyen ağırlık

değerlerine ihtiyaç duymaktadır. Hesaplama yöntemi ise, bu tabloda verilecek olan ağırlık değerlerine paralel değerlerdir.

Ağırlık değerleri, 0 ile 10 arasındaki rakamsal değerleri almaktadır (Macit, 2006).

$$\psi = LOC * (\lambda/100 + \varepsilon_i/100) \quad (1)$$

ψ: İşlevsel gösterge değeri

λ: Ağırlıklı sonuç değerlendirme oranı

ε_i: Ağırlık değerleri indeksi

Çizelge 2-3. Sonucu Etkileyen Faktörler ve Ağırlık Değerleri İndeksi (ε_i)

i	**Değer Kriteri**	**ε_i Değeri**
1	Hızlı kod geliştirme desteği gerekli mi?	0..10
2	Veri haberleşmesinin önemi nedir?	0..10
3	Gerçek zamanlı veriler kullanılacak mı?	0..10
4	Dağıtık işlem ve süreçler var mı?	0..10
5	Çoklu kullanıcı desteği var mı?	0..10
6	İşletim sistemi servis desteği var mı?	0..10
7	Kullanıcı arabirimleri tekli mi? / çoklu mu?	0..10
8	Veri depolama aygıtları dağıtık mı?	0..10
9	Program kodları yeniden kullanılır mı?	0..10

Kaynak: (Macit, 2006)

Çizelge 2-3'ten alınan değerler, Denklem 1'de verilen formülde yerine konarak işlevsel gösterge (ψ) değeri hesaplanır. Sonuç olarak bulunan değerler bütün yazılım projeleri için kesinlik belirtmeyebilir. Kesinlik durumunu belirlemek oldukça zordur. İşlevsel gösterge (ψ) değeri bize daha çok somut olarak ele alamadığımız, kesinlik ifade etmeyen bileşenlerin hesaplanmasında yardımcı olur.

2.2. Yazılım Ölçütleri

Yazılım ölçümü, yazılım süreçlerinin anlaşılması, kontrol edilmesi ve yönetilmesi için gerekli bir süreçtir. Yazılım ölçümü yazılım hataların analiz edilmesini, karmaşıklığının azaltılmasını, süreçlerin izlenmesini ve değerlendirilmesini sağlar. Büyüklük (size), emek (effort), süre (duration), maliyet (cost) ve kalite (quality) olmak üzere beş temel yazılım ölçütü vardır (Schach, 2007). Yazılım geliştirme sürecinde gelişimi sağlayabilmek için bu beş temel ölçüte ihtiyaç duyulmaktadır.

Yazılım ölçütleri yazılım geliştirme sürecinde, gereksinim duyulan büyüklük, emek, maliyet ve geliştirme süresine ilişkin kestirimlerin yapılmasına yardımcı olur. Yazılım maliyet ve emek kestirimi, bir yazılım projesinin başarıyla tamamlanmasında önemli bir rol oynar. Bir yazılım projesine ilişkin kaynaklar, projeyi tamamlamak için gerekli olan emeğe göre tahsis edilmektedir. Emek kestiriminin gerçeğe yakın elde edilmesi, yazılım projesinin zamanında tamamlamasını sağlar. Yazılım projeleri için gerekli olan emeği tahmin etmek üzere pek çok model ve yaklaşım geliştirilmiştir (Gencel & Demirors, 2008). Bu model ve yaklaşımların çoğu emek kestirimi için temel girdi olarak yazılımın büyüklüğünü kullanırlar. Büyüklük ölçütü sadece emek kestirimi için değil, aynı zamanda maliyet kestirimi içinde temel girdidir. Özetle büyüklük, emek ve maliyet ölçütleri arasında sürekli bir ilişki söz konusudur.

Bu tezin de konusu olan yazılım emek kestirimidir. Ancak emek kestirimini yapabilmemiz için büyüklük ölçütüne ihtiyacımız vardır. Maliyet kestirimi ise tez konusunun dışında bırakılmıştır. Bununla birlikte literatür taramamızda yer almaktadır.

3. YAZILIM BÜYÜKLÜK KESTİRİMİ

Bir yazılım projesinin maliyetini ve süresini proje tamamlanmadan belirleyebilmek için harcanacak iş gücünün yani emeğin tahmin edilmesi gerekmektedir. Harcanan emeği tahmin edebilmek için ise yazılım projesinin büyüklüğü tahmin edilmelidir. Bu tahminin elde edilmesi aşamasında, kodun uzunluğu veya kullanıcıya sağlanan işlevsellik gibi birçok yöntem kullanılabilir. Yazılım büyüklük ölçümünde kullanılacak yöntemleri; teknik ve işlevsel büyüklük ölçüm yöntemleri olarak sınıflandırabiliriz. Bu bölüm kapsamında teknik ve işlevsel büyüklük kestirim yöntemleri incelenmiştir.

3.1. Teknik Büyüklük Kestirim Yöntemleri

Teknik büyüklük ölçütleri, yazılımın büyüklüğünü geliştirici bakış açısından tanımlar. Günümüzde teknik büyüklük ölçüm yöntemi olarak, Kod Satır Sayısı (Lines of Code - LOC) ile kestirim yöntemi kullanılmaktadır.

3.1.1. Kod Satır Sayısı Yöntemi ile Kestirim

Kod Satır Sayısı, en eski ve en geniş çapta kullanılan yazılım büyüklük ölçütüdür. Teslim edilen yazılıma ait kaynak kodun satır sayısı olarak tanımlanmaktadır. Yazılım projelerinde emek, maliyet ve süre tahminleri ile diğer ölçütlerin normalizasyonu için bu ölçüt kullanılmaktadır. En eskilerinden biri olmasına rağmen, hala en popüler yazılım büyüklük ölçütlerinden biri kod satır sayıdır. Çünkü kod satır sayısı nesneldir, ölçülmesi ve anlaşılması kolaydır. Ancak, kod satır sayısı dile bağımlı olduğu için, farklı dillerde yazılmış programlar doğrudan karşılaştırılamaz. Kod satır sayısının doğru

olarak ölçümü, yalnızca proje kodunun yazılmasından sonra mümkündür. Projenin başlangıcında yani daha henüz kod yazılmamışken sadece bir ölçüm uzmanı tarafından ölçüm yapılabilir.

Kod satır sayısı değişik şekillerde kullanılmaktadır (Fenton, 1994). Bu standart bir sorundur. Bu standart sorunları maddeleyecek olursak;

- Kod satır sayısı sayılırken, boş satırlar ve açıklama satırları sayılmaz. Bu, açıklanmamış kod satır sayısı (non-commented lines of code) olarak adlandırılır. Diğer durumlarda, sadece açıklanmamış kod satır sayısı sayılmaz, ama kod açıklama satır sayısı (comment lines of code) sayılır. Toplam büyüklük, bunlara ilave olarak hesaplanır.

- Bazı durumlarda, farklı olarak çalıştırılabilir ifadelerin (executable statements) sayısı sayılırken, açıklama satırları, veri bildirimleri ve başlıkları göz ardı edilir.

- Aynı zamanda, Teslim Edilmiş Kaynak Komutlar (Delivered Source Instructions – DSI), yazılmış koddan ziyade teslim edilmiş kodu ölçmek için kullanılabilmektedir. "Program metni içindeki karakterleri sayısı", kod satır sayısından ziyade bir programın uzunluğunu ölçmek için kullanılmaktadır.

Bu uzunluk ölçütleri, birbirlerine kolaylıkla dönüşebilmektedirler. Kod satır sayısı ölçütünü kullanan bu varyasyonlar ve sayım için mevcut bir standardın oluşturulmaması nedeniyle; girdi olarak kod satır sayısını kullanan kestirimler içinde hata ve ölçümleri karşılaştırmak zordur.

Kod satır sayısı kestiriminde, proje tahmin edilen alt birimlerine ayrıştırılır. Parçala - yönet stratejisi sonucunda ortaya çıkan, üzerinde tahmin yapılması daha kolay olan her bir alt birim için satır sayıları önerilir. Bu kestirimler yapılırken de en küçük, en olası ve en büyük ihtimaller belirlenip, bunlarla bir ortalama işlemi yapılabilir. Bir birim için tahmin edilecek en küçük satır sayısına k, en olası satır sayısı tahminine o ve en büyük tahmin değerine de b denecek olursa, o birim için; satır sayısı kestirimi: $(k+4o+b)/6$ şeklinde hesaplanabilir (Ayyıldız, 2007).

Birimler için ayrı ayrı tahminler yapılır ve daha önceki deneyimlerden benzeri birimlerin geliştirilmesindeki şirketin verimliliği gibi değerler kullanılarak satır sayısı tahminlerinden emek, zaman ve maliyet kestirimlerine varılır. Projenin bütünü için, birimlerin emek, zaman ve maliyet kestirimleri toplanarak değerler elde edilir. Birimlerin satır sayıları toplanarak proje bütünü hakkında emek ve zaman gibi kestirim hesaplarını bir kerede yapmaktan kaçınılmalıdır. Satır sayısı büyüklüğü ile diğer sonuç değerlerinin doğrusal olmayan bir ilişki içinde olduklarını unutmamak gerekir (Ayyıldız, 2007).

3.2. İşlevsel Büyüklük Kestirim Yöntemleri

En fazla kullanılan diğer bir yazılım büyüklük ölçüm grubu, uzunluk niteliğinin aksine yazılımın işlevsellik niteliği ile ilgilidir.

İşlevsel Büyüklük Ölçümü (Functional Size Measurement - FSM), kullanıcıya teslim edilecek yazılımın işlevselliğini temel alır. FSM, işleve göre karmaşıklığın ve büyüklüğün belirlenmesi ile ölçülmektedir. Uzunluk ölçütleri, yazılım büyüklüğünü geliştirici bakış açısından göre değerlendirirken; FSM yazılımın büyüklüğünü kullanıcı bakış açısından ele alır.

İlk olarak İşlev Puanı (Function Points) ve İşlev Puan Analizi (Function Points Analysis – FPA) 1979 yılında IBM'in satır sayısına (Lines of Code – LOC) alternatif olarak yazılım büyüklük ölçümü için Allan Albrecht tarafından ortaya çıkartılmıştır. 1983'de ise, Allan Albrecht ve John Gaffney tarafından Yönetim Bilgi Sistemlerinin (Management Information Systems – MIS) büyüklüğünü ölçmek için FSM yöntemi geliştirilmiştir. Daha sonra farklı kitleler tarafından orijinal FPA yöntemi üzerinde yapılan düzenlemelerle, aralarında sayım yöntemi farklı birçok FSM yöntemi geliştirilmiştir (Fetcke, Abran, & Dumke, 2001).

FSM yöntemleri arasındaki bu farklılık yüzünden, 1996 yılında ISO çeşitli FSM konsepti geliştirme ve bunları bir standart altında toplayarak genel ölçüm prensibi

oluşturma çabasına girmiştir. 1998 yılında "İşlevsel Büyüklük", "Temel İşlevsel Bileşenler", "Temel İşlevsel Bileşen Tipleri" ve FSM yönteminin karakteristiği gibi temel FSM bileşenleri konseptini içeren 'ISO/IEC 14143 – Bilgi Teknolojileri – Yazılım Ölçümü – İşlevsel Büyüklük Ölçümü' nün ilk bölümü yayınlanmıştır (ISO/IEC TR 14143-3, 2003).

2000 yılında ise, IEEE buna uygun bir kabul standardı hazırlamıştır. Takip eden yıllarda ISO/IEC 14143'un kalan bölümleri de ISO tarafından yayınlanmıştır.

Bugüne kadar ISO sertifikalı 4 adet İşlevsel Büyüklük Ölçüm yöntemi yayınlanmıştır (Gencel Ç. , 2005). Bunlar; IFPUG FPA, MK II FPA, COSMIC FFP ve NESMA FSM'dir. Bu başlık altında yukarıda sayılan yöntemler incelenmiştir.

3.2.1. İşlev Puanı Yöntemi

İşlevsellik doğrudan ölçülemeyeceğine göre, bir yazılım projesinde işlevselliğe etkisi olan birçok etken bir arada incelenerek ürüne olan yansımaları ağırlıklandırılır. Sonuçta bir rakam ortaya çıkar ve bu rakam değişik projeleri göreceli olarak değerlendirmede yararlı olur.

Satır sayısının yerine işlev puanı koyularak diğer dolaylı ölçümlere ulaşılabilir. İki ayrı yaklaşımı birleştiren "dönüştürme" tabloları bulunmaktadır. Satır sayılı ya da işlev puanlı ölçmelerde birinden diğerine geçmek, dönüştürme tablosunda verilen değerlerle mümkündür.

Bu çevrimi değişik programlama dilleri açısından Çizelge 3-1'de verilmektedir.

Çizelge 3-1. Satır Sayısı / İşlev Puanı Dönüşümü

Programlama Dili	Ortalama LOC/FP
Derleme	320
C	128
COBOL	105
FORTRAN	105
Pascal	90
Ada	70
Nesne Tabanlı Diller	30
4. Kuşak Diller (4GL)	20
Kod Üreticiler	15

Kaynak: (Pressman, 2009)

İşlev puanı hesaplamak için karmaşıklığa etki eden faktörlerin ağırlıklandırılmaları ve sonuçta karmaşıklığa olan etkileri denenerek bu ağırlıkların ayarlanması şeklinde yöntemler uygulanmıştır. İşlev puanının hesaplanması için iki işlem yapılır.

Önce, nesnel ölçümlere dayalı bir büyüklük olan Düzeltilmemiş İşlev Puanı (Unadjusted Function Points - UFPs) hesaplanır. İşlev Puanında sistemin işlevselliği beş ayrı ölçme parametresi ile incelenmektedir. UFPs'yi bulmak için, ölçme parametreleri basit, orta ve karmaşık olmak üzere üç karmaşıklık düzeyine göre sınıflandırılır ve Çizelge 3-2'de verilen ağırlık faktörleri ile çarpılır. Bu ölçülen değerler toplanarak UFPs elde edilir (Schach, 2007).

Çizelge 3-2. Nesnel Ölçümlere Dayalı Büyüklük Hesaplama Tablosu

Ölçme Parametresi	Adet		Basit	Orta	Karmaşık		
Kullanıcı Girdi Sayısı		x	3	4	6	=	
Kullanıcı Çıktı Sayısı		x	4	5	7	=	
Kullanıcı Sorgulama Sayısı		x	3	4	6	=	
Kütük (Dosya) Sayısı		x	7	10	15	=	
Dış Arayüz Sayısı		x	5	7	10	=	
Düzeltilmemiş İşlev Puanı (Unadjusted Function Points - UFPs)						=	

Dış arayüz sayısı, kütükler ve bunun gibi her türlü makine tarafından anlaşılan bilgi aktarım noktaları sayısıdır. Kullanıcı girdileri, kullanım sırasındaki veri girişleridir. Sorgulama ile karıştırılmamalıdır. UFPs bu şekilde bulunduktan sonra Denklem 2'deki İşlev Puanı (FP) formülünde yerine konur (Sommerville, 2006):

$$FP = UFPs * (0.65 + 0.01 * K_i) \quad (2)$$

Burada K_i, (i=1'den 14'e kadar) 14 adet "Karmaşıklık Ayarlama Faktörü" dür ve Çizelge 3-3'teki soruların cevabından meydana gelir.

Bu sorulara verilecek cevapların değeri 0 ile 5 arasında değişmektedir. Böylece ortaya çıkacak 14 değer toplanır ve 0.01 ile çarpılarak formüle konur.

Çizelge 3-3. Karmaşıklık Ayarlama Faktörleri (K_i)

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 veri girişi gerekecek mi?
7.	Çevrimiçi giriş, fazla ekranlı veya fazla işlemli mi?
8.	Ana kütükler çevrimiçi 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?

Kaynak: (Sommerville, 2006)

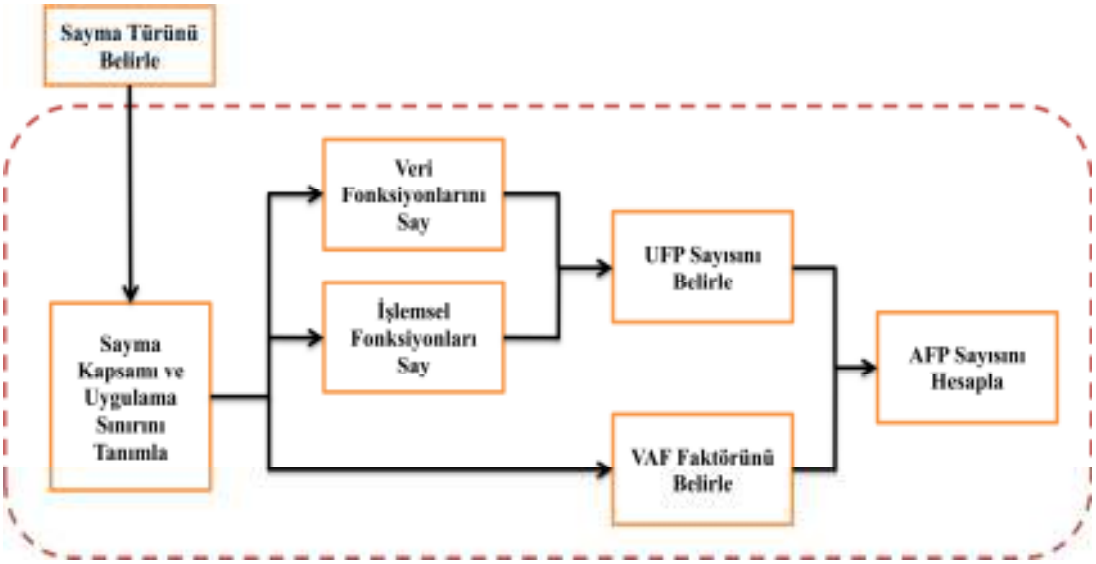
3.2.2. IFPUG İşlev Puanı Analizi

Orijinal FPA yöntemini sürdürmek için 1984'te Uluslararası İşlev Noktası Kullanıcıları Grubu (International Function Point Users Group - IFPUG) oluşturuldu (ISO/IEC 20926, 2003).

IFPUG yazılımın işlevselliğini ölçmek için, FPA yönteminin kullanımını teşvik eden, kar amacı olmayan ve üyeleri tarafından idare edilen bir örgüttür. IFPUG uygulama yazılımı geliştirme ve bakım faaliyetlerinin yönetiminde FPA'ın kullanımını teşvik etmektedir ve FPA'yı yazılım büyüklüğü için standart metodoloji olarak kabul etmektedir (IFPUG, IFPUG: International Function Point User Group, 2009).

IFPUG, başlangıçta Albrecht tarafından sunulmuş olan tekniği geliştirerek, bu yöntemi standartlaştırmıştır. Resmi IFPUG Ölçüm Uygulama Kılavuzları sırasıyla 1986, 1988, 1990, 1994, 1999 ve 2009'da yayınlanmıştır (IFPUG, IFPUG: International Function Point User Group, 2009).

IFPUG FPA en yaygın olarak kullanılan FSM yöntemidir (IFPUG, 1999). IFPUG işlev puanı ölçüm prosedürü Şekil 3-1'de verilmektedir (Cândido & Sanches, 2004).



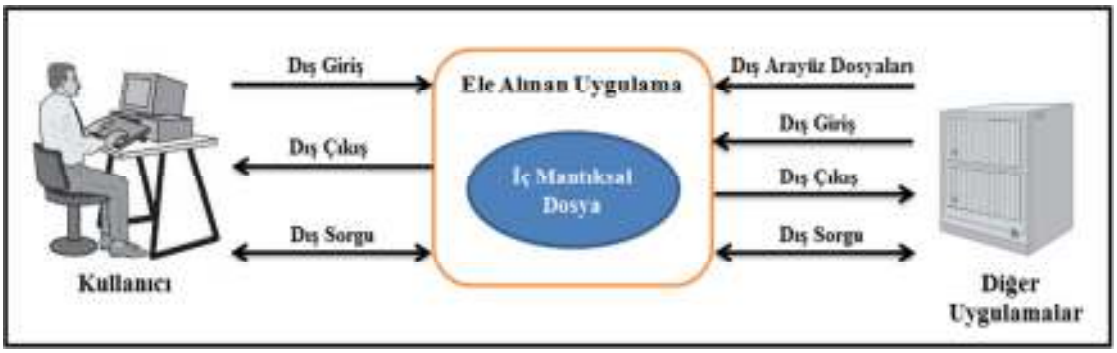
Şekil 3-1. IFPUG İşlev Puanı Ölçüm Prosedürü

Kaynak: (Cândido & Sanches, 2004)

IFPUG İşlev Puanı Analizinde, sistemin işlevselliği 5 ayrı bileşenle incelenmektedir.

- Giriş Verileri: Veri giriş ekranları, mantıksal dâhili dosyalar.
- Çıkış Verileri: Ekran çıktıları, raporlar.
- Arayüz: Başka bir sistemle paylaşım.
- Veri Dosyaları: Dâhili kullanıcı verileri, saklanan veriler.
- Sorgular: Kullanıcı isteği doğrultusunda alınan hızlı veri çıkışları.

Şekil 3-2'de IFPUG FPA bileşenleri arasındaki işleyiş görülmektedir .



Şekil 3-2. IFPUG FPA Bileşenleri

Kaynak: (Özkanat, 2007)

Her bir bileşenin zorluk derecesi basit, orta ve karmaşık gibi Çizelge 3-4'te verilen rakamsal değerlere bağlı olarak ölçülebilmektedir (Özkanat, 2007).

Ölçülen bu değerler toplanarak Düzeltilmemiş İşlev Puanı'nı (Unadjusted Function Points - UFPs) oluşturmaktadır.

Çizelge 3-4. İşlev Puanı Karmaşıklık Dereceleri

Bileşen	Basit	Orta	Karmaşık
Girdiler (I)	3	4	6
Çıktılar (O)	4	5	7
Veri Dosyaları (F)	7	10	15
Ara Yüzler (I)	5	7	10
Sorgular (Q)	3	4	6

Çizelge 3-5'de verilen 14 Genel Sistem Özelliğine göre sistemin beklenilen uygulama zorluğu için ilave bir ayarlama faktörü tahmin edilmektedir. Her bir genel sistem özelliği, hiçbir gerekli etkisi olmadığı anlamına gelen 0'dan 5'e bir değer almaktadır (Özkanat, 2007).

Çizelge 3-5. İşlev Puanı Genel Sistem Özellikleri

	Genel Sistem Özellikleri	**Kısa Açıklama**
1	Veri İletişimleri	Sistemin uygulaması ile bilgi değişimi veya transferinde yardımcı olmak için kaç tane iletişim aracı vardır?
2	Dağıtılan Veri/İşleme	Dağıtılan bilgi ve işleme fonksiyonları nasıl idare edilmektedir?
3	Performans	Hedefler, yanıtlama zamanı ve iş çıkarma performansı önemli midir?
4	Çok Kullanılan Konfigürasyon	Uygulamanın idare edileceği mevcut donanım platformu ne kadar yoğun kullanılmaktadır?
5	İşlem Oranı	İşlem oranı yüksek midir?
6	Çevrimiçi Veri Girişi	Hangi oranda bilgi çevrimiçi girilmektedir?
7	Son Kullanıcı Verimliliği	Uygulama son kullanıcı verimliliği için mi tasarlanmıştır?
8	Çevrimiçi Güncelleme	Kaç veri dosyası çevrimiçi güncellenmektedir?
9	Karmaşık İşlem Yapma	Dâhili işlem yapma karmaşık mıdır?
10	Yeniden Kullanılabilirlik	Uygulama yeniden kullanılabilir olması için mi tasarlanmıştır?
11	Dönüştürme/Kurulum Kolaylığı	Sistemde otomatik dönüşüm ve kurulum da dâhil edilmiş midir?
12	İşlevsel Kolaylık	Yedekleme, başlatma ve kurtarma gibi operasyonlar ne kadar otomatiktir?
13	Çoklu Saha Kullanımı	Uygulama çoklu örgüte sahip çoklu sahalar için özellikle mi tasarlanmış, geliştirilmiş ve desteklenmiştir?
14	Değişimi Kolaylaştırma	Uygulama kullanıcı tarafından kullanım kolaylığı ve değişimi kolaylaştırmak için özel olarak mı tasarlanmış, geliştirilmiş ve desteklenmiştir?

İşlev puanı genel sistem özelliklerine ait görev değerleri Çizelge 3-6'da verilmiştir.

Çizelge 3-6. İşlev Puanı Genel Sistem Özellikleri Görev Değerleri

0	1	2	3	4	5
Etkisiz	Önemsiz	Hafif	Orta	Önemli	Gerekli

Çizelge 3-6'da verilen genel sistem içerisindeki 14 özellik için verilen değerler toplanarak, Denklem 3'de verilen formüle bağlı bir şekilde tekrar hesaplanır (Sommerville, 2006).

$$Adjusted\ Function\ Points\ (AFPs) = UFPs * (0.65 + 0.01 * TCF) \quad (3)$$

UFPs: Düzeltilmemiş İşlev Puanları, TCF: Teknik Karmaşıklık Faktörü

TCF (Technical Complexity Factor) değerleri 0 ila 70 arasında değişiklik gösterebileceği için, bu da AFP'yi ± %35 şeklinde etkilemektedir.

3.2.3. Mark II İşlev Puanı

Charles Symons'a göre (Jones, 1991);

- Bir uygulamanın bileşenlerinin belirlenmesi Albrecht'in FPA'ında zordur,
- Albrecht yaklaşımı iç karmaşıklıkla ilgili hiçbir düşünceye sahip değildir,
- Küçük sistemler daha büyük sistemlerle birleştirildiklerinde, Albrecht yaklaşımında toplam işlev puanı sayısı küçük sistemleri kapsayan tüm işlev puanlarının toplamından daha azdır.
- On dört ayarlama faktörü yeterli değildir.

Symons, Albrecht'in FPA yönteminde gördüğü bu eksiklikleri çözmek için 1980'lerde İngiltere'de MKII İşlev Puanını geliştirdi. Ayrıca MK II, kullanıcıya

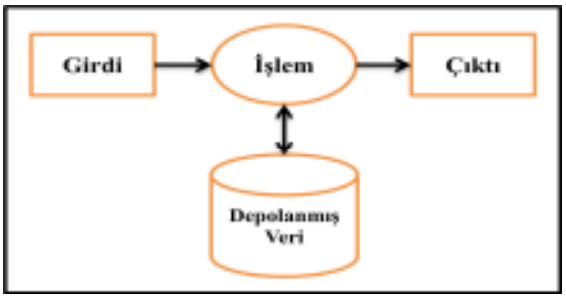
sağlanan işlevselliğin değerinden çok işlevselliği üretmek için gerekli emeğe odaklamak için tasarlanmıştır (Symons C. , 1991).

MK II, uygulamayı mantıksal işlem gruplarına ayrıştırmaktadır. Symons mantıksal bir işlemi; *"bilgi almak için bir gereksinim ya da kullanıcıyı ilgilendiren bir olay ile tetiklenen benzersiz bir girdi/işlem/çıktı birleşimi"* olarak tanımlar (Symons C. , 1991).

Her işlem aşağıdaki 3 bileşenden oluşmaktadır:

- Girdiler: Dış ortamdaki bir kullanıcıdan yazılıma gelen veriler,
- Çıktılar: Yazılımdan dış ortamdaki bir kullanıcıya giden veriler,
- İşlemler: Kalıcı hafızaya depolama, kalıcı hafızadan geri alma ve silme.

MK II işlem bileşenleri arasındaki ilişki Şekil 3-3'de görülmektedir.



Şekil 3-3. MK II İşlem Bileşenleri

Kaynak: (Symons C. , 1991)

Düzeltilmemiş İşlev Puanı (Unadjusted Function Points - UFP), girdi, çıktı ve işlemlerin toplam sayılarının bir fonksiyonudur. Bileşen ağırlıkları Çizelge 3-7'de verilmektedir.

Çizelge 3-7. MK II Bileşen Ağırlıkları

Bileşen	Ağırlık
Girdiler	0.58
İşlemler	1.66
Çıktılar	0.26

Kaynak: (Symons C. , 1991)

MK II yöntemi, gereksinim özelliklerinden bilgi sistemlerinin büyüklüğünü ölçmek için tasarlanmıştır. Bunun yanında, MKII yöntemi bilgi süreçlerini ölçmeyi hedeflemektedir (Symons C. , 2001). MK II yöntemi, bilimsel ve gerçek-zamanlı yazılım uygulamalarına uygulanabilir, ama yöntem bazı değişiklikler gerektirebilir (UKSMA, 1998).

3.2.4. NESMA İşlev Puanı Analizi

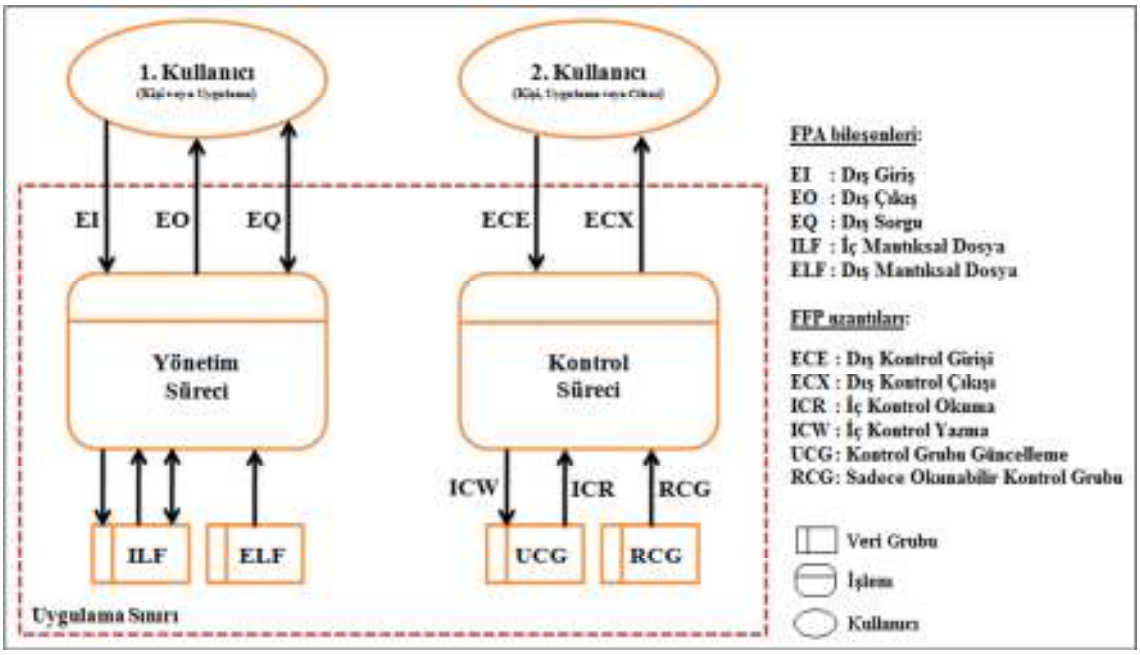
1989'da, Hollanda Yazılım Ölçüt Kullanıcıları Birliği (Netherlands Software Metrics Users Association - NESMA), Hollanda İşlev Puanı Kullanıcıları Grubu (Netherlands Function Point Users Group - NEFPUG) olarak kuruldu. NESMA, Avrupa'daki en büyük FPA kullanıcı grubudur. İşlev Puanı Analizinin uygulanması ile ilgili tanımlar ve ölçüm kılavuzunun ilk versiyonu 1990'da yayınlanmıştır. Bu yöntem, IFPUG FPA yönteminin ilkelerini temel almaktadır. IFPUG FPA'a benzer olarak işlevselliğin büyüklüğü için, Dış Giriş, Dış Çıkış, Dış Sorgu, İç Mantıksal Dosya ve Dış Arayüz Dosyası gibi işlev türlerini kullanmaktadır. NESMA FPA'ın farkı, ölçüm uygulamaları kılavuzunun daha somut kurallar, ipuçları ve örnekler vermesidir (NESMA, 1997) (Gencel Ç. , 2005).

3.2.5. Tam İşlev Puanı

Tam İşlev Puanı (Full Fuction Points - FFP) yöntemi, Quebec Üniversitesi ve SELAM (Software Engineering Laboratory in Applied Metrics) işbirliği ile 1997 bir araştırma projesinin sonucu olarak geliştirilmiştir (St-Pierre, Maya, Abran, & Desharnais, 1997). Tam İşlev Puanı yöntemi, Yönetim Bilgi Sistemlerinin yanında gerçek zamanlı sistemlerde de işlevsel büyüklük ölçümünün uygulanabilmesi için tasarlanmıştır.

Tam İşlev Puanı, IFPUG FPA yönteminin bir uzantısıdır. Şekil 3-4' te görüldüğü üzere Tam İşlev Puanı, 11 bileşeni 3 temel fonksiyon içinde kullanmaktadır.

- Yönetim fonksiyonları: IFPUG FPA'ın 5 temel bileşeni.
- Veri fonksiyonları: Güncellenmiş kontrol grubu ve sadece okunabilir kontrol grubu.
- İşlemsel fonksiyonlar: Dış kontrol girişi, dış kontrol çıkısı, iç kontrol okuma, iç kontrol yazma



Şekil 3-4. Tam İşlev Puanı Bileşenleri

Kaynak: (St-Pierre, Maya, Abran, & Desharnais, 1997)

3.2.6. COSMIC Tam İşlev Puanı

Uluslararası Ortak Yazılım Ölçüm Birliği (COSMIC - Common Software Measurement International Consortium), işlevsel yazılım büyüklük ölçümüne yönelik yeni bir metot geliştirmek amacıyla 1998'de kurulmuştur (Abran, Desharnais, Oligny, St-Pierre, & Symons, 2003). IFPUG, Mark II ve NESMA gibi mevcut yazılım büyüklük

kestirim yöntemlerinin en iyi karakteristik özellikleri alınarak, yeni bir işlev büyüklük ölçüm yöntemi olarak COSMIC FFP'yi Kasım 1999'da yayınlamıştır.

COSMIC FFP yöntemi, Yönetim Bilişim Sistemi (Management Information Systems) projelerinin yanı sıra, gerçek-zamanlı yazılım projelerinde kullanılmak üzere tasarlanmış işlev puanı yönteminin bir evrimidir.

COSMIC FFP yöntemi, geliştirici ve son kullanıcı bakış açıları olmak üzere birçok ölçüm bakış açısına sahiptir. Oysa son kullanıcı bakış açısından IFPUG ile aynı, geliştirici bakış açısından ise tasarlanmış sistem mimarisi olarak düşünülür.

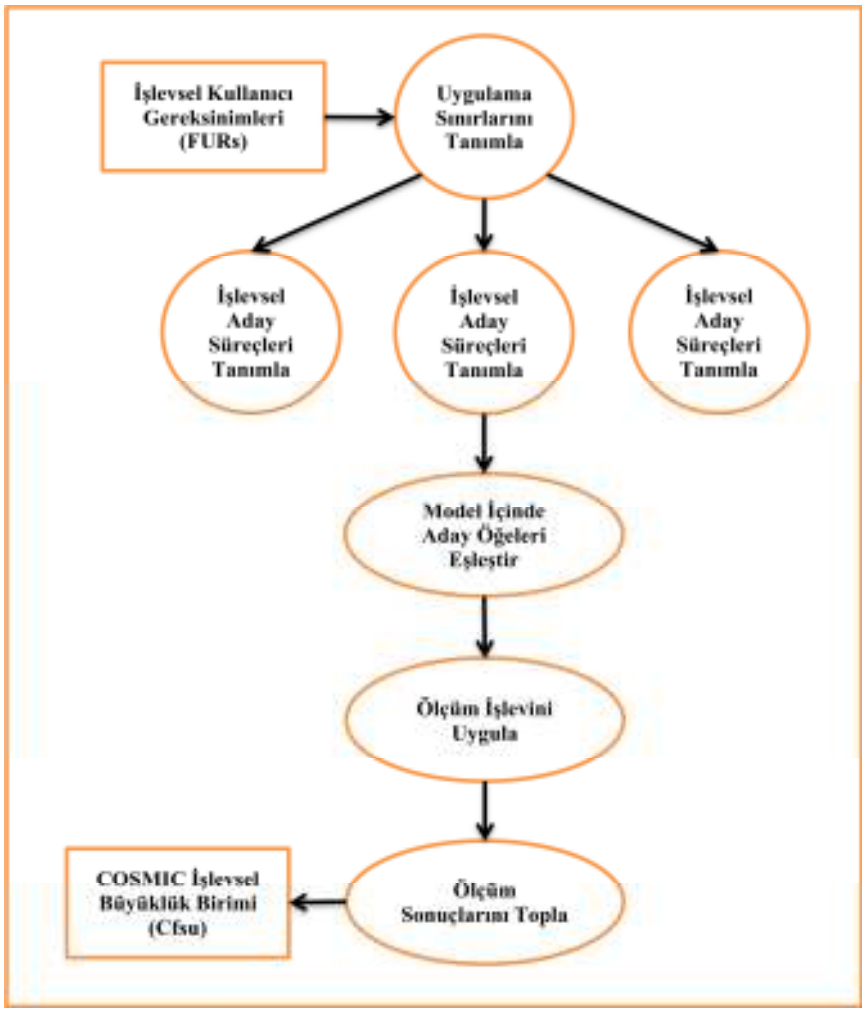
COSMIC FFP yöntemi, İşlevsel Kullanıcı Gereksinimlerini (Functional User Requirements - FURs) temel alan bir yazılımın işlevsel büyüklüğünü ölçmek için tasarlanmıştır (Abran, Fagg, Meli, & Symons, 2002). Teknik ve Kalite Gereksinimleri, İşlevsel Kullanıcı Gereksinimlerine (FUR) dâhil değildir. Bir yazılım ya İşlevsel Kullanıcı Gereksinimlerinden ya da daha önceden geliştirilmiş bir yazılım parçasından ortaya çıkarılır. Bir yazılım parçasının işlevsel büyüklüğü zaten ölçülebilir. Yazılımın işlevsel büyüklüğü dört İşlevsel Tabanlı Bileşeni (Base Functional Component - BCF) temel alarak ölçülmektedir. İşlevsel Tabanlı Bileşenler; Giriş (Entry), Çıkış (Exit), Okuma (Read) ve Yazma (Write)'dır.

COSMIC FFP yönteminde ölçüm süreci beş adımda icra edilmektedir.

- Yazılımın sınırlarını ölçebilmek için, yazılım ve donanım arasındaki etkileşimin özellikleri ve gereksinimleri tanımlanır.
- Olası bütün işlevsel süreçler, tetikleyen olaylar ve veri grupları, gereksinimlerden tanımlanır. Bu aşamada bunlar aday faktörler olarak değerlendirilebilir.
- Aday öğeler (işlevsel süreçler, tetikleyen olaylar ve veri grupları), COSMIC-FFP kurallarını temel alan COSMIC-FFP yazılım durum modeli içinde eşleştirilir. Bu eşleştirmede, her bir işlevsel süreç, süreç tarafından işlenen veri grupları ve bir tetikleme olayı ile ilişkili olmalıdır. Bu eşleştirme aynı zamanda katmanların tanımlanmasına izin verir.

- COSMIC-FFP alt süreçleri (Giriş, Çıkış, Okuma ve Yazma), her bir işlevsel süreç içinde tanımlanır. Her bir veri faaliyeti, 1 COSMIC işlevsel büyüklük birimi (Cfsu) verir.
- Ölçümü gerçekleştiren kişi, ölçülmüş olan yazılımın toplam işlevsel büyüklüğünü elde etmek için ölçüm sonuçları toplamını hesaplar.

Müşteriye iletilen büyüklük, bazen beklenen büyüklükten farklı olabilir. COSMIC FFP yöntemindeki ölçüm sürecine ait adımların akışı Şekil 3-5'de verilmiştir.



Şekil 3-5. COSMIC-FFP Yöntemi ile Yazılım Büyüklük Ölçüm Prosedürü

Kaynak: (Abran, Desharnais, Oligny, St-Pierre, & Symons, 2003)

3.2.7. Nesne Puanı

Yazılım büyüklük ölçümünde başvurulan başka bir yöntem de, Nesne Puanı (Object Points)'dır. Bu yöntem, Kauffman ve Kumar'ın daha önceki çalışmalarına dayanarak 1993'te New York Üniversitesi'nde geliştirilmiştir (Banker, Kauffman, Wright, & Zweig, 1994). Yöntemin temelini oluşturan kavramlar, FPA yöntemi içerisinde yer alan kavramlara çok benzerdir, fakat işlevlerin yerine nesneler sayılmaktadır. Nesneler, üçüncü nesil programlama dilleri modüllerini, raporlarını ve ekranlarını içerir. Nesne Puanı, nesneye yönelik programlama içerisinde yer alan nesne sınıfları ile aynı değildir. Ham nesnelerin sayısı ve her bir nesnenin karmaşıklığı tahmin edilir. Ardından toplam ağırlık hesaplanır. Aynı zamanda, yeniden kullanım yüzdesi ve beklenen verimlilik de bu yöntem ile tahmin edilebilmektedir.

3.2.8. Nesne-Tabanlı İşlev Puanı

Nesne-Tabanlı İşlev Puanı (Object-Oriented Function Points - OOFPs), nesne-tabanlı yazılım geliştirme projelerinde kullanılmak üzere IFPUG FPA' in bir uyarlaması olarak 1998'de Caldiera tarafından geliştirilmiştir (Antoniol, Fiutem, & Lokan, 2003). Nesne-Tabanlı İşlev Puanı metodunda temel fikir, sınıflar ve bu sınıflara ait metotlar için IFPUG FPA' deki gibi işlemler ve mantıksal dosyaların değiştirilmesidir. Nesne-Tabanlı İşlev Puanı, sınıf metotları için FPA'in sınıflarını ve işlemlerini, FPA'in mantıksal dosyalarıyla eşleştirir. Çizelge 3-8'de IFPUG FPA ile Nesne-Tabanlı İşlev Puanı arasındaki bileşen türü dönüşümleri verilmektedir.

Çizelge 3-8. IFPUG FPA ile OOFPs Arasındaki Bileşen Türü Dönüşümleri

	IFPUG FPA	**OOFPs**
Mantıksal Dosyalar	Dış arayüz dosyası	Dış sınıf
	İç mantıksal dosya	İç sınıf
İşlemler	Girdi	Servis isteği
	Çıktı	
	Sorgu	

FPA' da mantıksal dosyalar, uygulama sınırına göre dış ara yüz dosyaları ile iç mantıksal dosyalar içine bölünmüştür. OOFPs'de dış sınıflar, dış ara yüz dosyalarına ve iç sınıflara karşılık gelen dış servisleri veya kütüphane sınıflarını yeniden kullanmaktadır. OOFPs'de, işlem türlerinin tamamı, girdiler, çıktılar ve sorgular sınıf metotları olan Servis İsteklerine karşılık gelmektedir. OOFPs'in hesaplanması Denklem 7'deki gibidir (Antoniol, Fiutem, & Lokan, 2003);

$$OOFP_{ILF} = \sum_{O \in A} w_{ILF}(DET_O,\ RET_O) \tag{4}$$

$$OOFP_{EIF} = \sum_{O \notin A} w_{ELF}(DET_O,\ RET_O) \tag{5}$$

$$OOFP_{SR} = \sum_{O \in A} w_{SR}(DET_O,\ FTR_O) \tag{6}$$

$$OOFP = OOFP_{ILF} + OOFP_{EIF} + OOFP_{SR} \tag{7}$$

ILF: Internal Logic File – İç Mantıksal Dosya

ELF: External Logic File – Dış Mantıksal Dosya

SR: Service Request – Servis İsteği

DET: Data Element Types – Veri Elemanı Türleri

RET: Record Element Types – Kayıt Elemanı Türleri

"A", uygulamaya ait nesneler kümesini ifade eder. "O", nesne modelindeki genel bir nesnedir. "W" ise, ağırlık fonksiyonudur.

Buna göre her bir sınıfın karmaşıklığı, Veri ve Kayıt Elemanı Türlerinin sayısı ile belirlenebilir. Burada, Veri Elemanı Türleri (Data Element Types), "DET" ile Kayıt Elemanı Türleri (Record Element Types) ise, "RET" ile ifade edilir. DET, sınıfın basit özelliklere sahip nitelikleridir. RET ise, sınıfın karmaşık özelliklere sahip nitelikleridir.

Daha sonra OOFPs'i saymak için, IFPUG kuralları ile ortaya çıkarılan mantıksal dosyalara göre karmaşıklık başvuru tablosu kullanılır. Her bir Servis İsteği'nin (Service Request - SR) karmaşıklığı, her metodun argümanlarının sayısına göre hesaplanır. Basit

argümanlar DETs olarak sayılırken, nesneler gibi karmaşık argümanlar Başvurulan Dosya Türleri (File Types Referenced – FTRs) olarak sayılmaktadır. Bir metodun DETs ve FTRs'leri sayıldığında düşük, orta ve yüksek karmaşıklık olarak metotları sınıflandırmak için dış girişlere göre IFPUG Kılavuz Uygulamaları Ölçüm tablosu kullanılmaktadır.

OOFP, geliştirilmiş sınıflar ve servis isteği ile yeniden kullanılan sınıflar ve servis istekleri arasında net bir ayrım yoluyla yeniden kullanılabilirlik düzeyini ölçme yeteneğine sahiptir (Antoniol, Fiutem, & Lokan, 2003).

3.2.9. Nesne-Tabanlı Yöntem İşlev Puanı

Nesne-Tabanlı Yöntem İşlev Puanı (OO-Method Function Points - OOmFP), nesne-tabanlı sistemler için 2001 yılında Pastor ve arkadaşları tarafından tasarlanmış yeni bir işlevsel büyüklük ölçüm yöntemidir (Pastor, Abrahão, Molina, & Torres, 2001).

OOmFP, IFPUG FPA ölçüm kurallarına uygun şekilde tasarlanmıştır. Ancak, IFPUG ölçüm kuralları, nesne-tabanlı yöntemlerde kullanılan kavramlar açısından yeniden tanımlanmıştır (Gencel Ç. , 2005).

OOmFP, IFPUG-FPA' de olduğu gibi, veri ve işlevsel fonksiyonları dikkate alır. Sınıflar, İç Mantıksal Dosyalar (ILF) ve Dış Arayüz Dosyalarının (EIFs) eski görünümü olarak kabul edilmektedir. Servisler, bir sınıf ya da eski görünüm içinde tanımlanan Dış Girişler (EIs) olarak sınıflandırılmaktadır. Sunum modeli içinde tanımlanan, Örnek Etkileşim Birimi (Instance Interaction Unit - IIU), Etkileşim Birimi Yoğunluk (Population Interaction Unit - PUB) ve Ana-Detay Etkileşim Birimi (Master-Detail Interaction Unit - MDIU) modelleri bir sınıfın nesne topluluğunu görselleştirmek için, Dış Girişler (EOs) ve Dış Sorgular (EQs) olarak kabul edilmektedir (Gencel Ç. , 2005).

İşlevsel büyüklük ölçümü kavramsal şema düzeyinde yapılmaktadır. Nesne-tabanlı yöntem içinde bulunan kavramsal model görünümleri, tüm bilgileri ölçüm için

kullanmaktadır. Aynı zamanda nesne-tabanlı kavramlar, kalıtım (inheritance) ve kümeleme (aggregation) gibi kavramları ele almaktadır (Laranjeira, 1990).

OOmFP karmaşıklık ağırlıkları Çizelge 3-9'da verilmektedir.

Çizelge 3-9. OOmFP Karmaşıklık Ağırlıkları

Fonksiyon Tipi	Düşük	Orta	Yüksek
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

Kaynak: (Gencel Ç. , 2005)

Sonuç olarak, OOmFP değeri Denklem 9'da verilen formül ile hesaplanmaktadır (Laranjeira, 1990):

$$OOmFP_{veri} = \sum_{i=1}^{n} OOmFP_{sınıf_i} + \sum_{j=1}^{m} OOmFP_{eski\_görünüm_j} \tag{7}$$

$$OOmFP_{işlem} = \sum_{i=1}^{n} OOmFP_{servis_i} + \sum_{j=1}^{m} OOmFP_{IIU_j|PIU_j|MDIU_j} \tag{8}$$

$$OOmFP = OOmFP_{veri} + OOmFP_{işlem} \tag{9}$$

OOmFP: OO-Method Function Points – Nesne-Tabanlı Yöntem İşlev Puanı

4. YAZILIM EMEK KESTİRİMİ

Emek kestirimi, bir yazılım projesini geliştirmek için kaç kişi ve ne kadar zaman gerektiğini tahmin ederken kullanılmaktadır. Bir yazılım projesinde emek yatırımı, son yıllarda proje yönetim süreci içindeki en önemli analiz değişkenlerinden biridir. Yazılım projeleri başlatılırken bu değişken değerinin bilinmesi, yazılım yöneticilerinin gelecek faaliyetleri gerektiği gibi planlamasını sağlar. Emek kestiriminde iyi tahminler elde etmek için önceki projelerden elde edilen verileri dikkate almak önemlidir.

Yazılım emek kestiriminde kullanılan yöntemler, algoritmik ve algoritmik olmayan kestirim yöntemleri olarak sınıflandırılmaktadır. Bu bölüm kapsamında algoritmik ve algoritmik olmayan emek kestirim yöntemleri ele alınmıştır.

4.1. Algoritmik Olmayan Emek Kestirim Yöntemleri

Algoritmik olmayan emek kestirim yöntemleri; uzman kararı, benzerlik ile kestirim ve büyüklük verisi kullanarak kıyaslamadır. Bu kestirim yöntemlerini özellikle yazılım sektöründe yeni iş yapmaya başlayan kurumlar tercih etmektedir. Daha çok bu yöntemler geçmiş proje deneyimine sahip uzman kişiler üzerine kuruludur.

4.1.1. Uzman Kararı ile Kestirim

Uzman kararı yöntemi, yazılım endüstrisinde emek kestirimi için en çok kullanılan yöntemdir. Yıllardan beri proje yöneticileri kendi deneyimlerine güvenmişlerdir. Uzman kararında, pek çok uzman önerilen yazılımın uygulama alanı ve geliştirme tekniklerine göre proje emeğini tahmin etmektedir (Laird & Brennan, 2006).

Yeni proje daha önce tamamlanan projelerden çok farklı değilse ve deneyimli kestirimciler mevcutsa, emek kestirimi için en uygun yöntem uzman kararıdır. İyi bir uzman bulunabilirse göreceli olarak iyi bir tahminde bulunabilir (Ayyıldız, 2007). Ancak, kestirimde uzman kararını kullanmanın bazı zorlukları da vardır. Bu kişisel bir kestirim olacaktır. Her yeni proje için çok deneyimli kestirim yapabilecek kişi bulmak zordur. Sonuç ortaya çıkana kadar uzmanın görüşünü test etmenin hiçbir yolu yoktur ve bu çok geç olabilir. Eğer uzman yoksa çok yanlış olacaktır.

4.1.2. Benzerlik ile Kestirim

Bu yöntemde, projeler pek çok nitelikle tanımlanır. Bu nitelikler tamamlanmış projeler arasından yeni projeye en çok benzeyen projeleri belirlemek için kullanılır. Yeni projenin emek kestirimi, yeni proje ile tamamlanmış projeler arasındaki farklılıklar dikkate alınarak, tamamlanmış benzer projelere göre belirlenir.

Benzerlik esaslı kestirim, bütün proje seviyesinde veya alt sistem seviyesinde gerçekleştirilebilir. Bütün proje seviyesini esas alan kestirimler, sistemin tüm maliyet bileşenlerini dikkate alır. Diğer taraftan, alt sistem seviyesini esas alan kestirimler, yeni proje ile tamamlanmış projeler arasındaki benzerlikler ve farklılıkların daha detaylı bir değerlendirmesini sağlamaktadır (Laird & Brennan, 2006).

Benzerlik esaslı kestirim için, daha önce tamamlanmış benzer projelere ait geçmiş veriler gereklidir. Bu nedenle, bu kestirim yöntemi tamamlanmış projelere ait verileri tutan veri tabanlarına gereksinim duymaktadır. Bununla birlikte bu yöntem, algoritmik benzerlikleri hesaplamak için bir araç kullanan büyük ölçekli projelerde kullanılabilmektedir (Shepperd & Schofield, 1997).

4.1.3. Büyüklük Verisini Kullanarak Kıyaslama

Bu yöntem, verimliliğin bir uygulama alanı ve büyüklük fonksiyonu olduğu düşüncesini temel almaktadır (Laird & Brennan, 2006). Özellikle yazılım projesi ile ilgili bilgilerin doğru olduğuna inanılıyorsa ve projenin büyüklüğü kod satır sayısı veya işlev puanı gibi bir yöntemle belirlenebiliyorsa, bu proje için gerekli emek basit bir bölme işlemi ile hesaplanabilmektedir. Denklem 10'da verilen formüle göre, büyüklük verisini kullanarak kıyaslama ile emeği hesaplamak çok basittir.

$$Emek = \frac{Proje\ Büyüklüğü}{Teslimat\ Oranı} \tag{10}$$

Teslimat oranı, belirli bir süre içerisinde üretilerek teslim edilen çıktıların ortalama oranını yansıtır.

4.2. Algoritmik Emek Kestirim Yöntemleri

Bu yöntemler, emek kestirimi için matematiksel modelleri kullanılırlar. Bu tür modellere geçmişe ait veriler, tahmin edilen kod satır sayısı, fonksiyon sayısı vb. istatistikler ve beraberinde programlama platformu, kullanılan yöntem gibi bilinen ortam faktörleri girdi olarak verilir. Model belirli bir doğruluk aralığında sonuç üretir. Bu tür modellerde içinde bulunulan ortama göre bazı parametre değerlerinin "kalibre" edilmesi gerekmektedir. En çok kullanılan algoritmik emek kestirim yöntemleri; COCOMO 81 (Constructive Cost Model 81), COCOMO II, Putnam'ın Yazılım Yaşam Döngü Modeli (Putnam's Software Lifecycle Model – SLIM), Use-Case Puanı ve Sınıf Puanı'dır.

4.2.1. COCOMO 81

Gereken emeğin, program büyüklüğünün bir üssel değerine bağlı olması prensibi ve endüstriden toplanan bilgiler ışığı altında bir maliyet kestirim metodu olarak 1981'de Boehm tarafından COCOMO (Constructive Costing Model) 81 modeli geliştirilmiştir (Hughes & Cotterell, 2009).

COCOMO 81 modeli göreceli olarak dosdoğru bir modeldir. Yapılacak hesapların kapsamları açısından; basit, orta ve detaylı olmak üzere üç değişik modelden oluşur. Ayrıca bu modellerde kullanılacak problemler, organik, yarık ayrık ve gömülü sınıflar altında toplanmıştır:

- Organik problemler için küçük boyuttaki programcı takımları, bildikleri ortamlarda iyi anlaşılmış uygulamaları geliştirirler. İletişim problemi azdır ve elemanlar çabucak işlerini halledebilecek durumdadırlar.

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

- Gömülü problemlerde ise geliştirilecek yazılım, sistemin donanım, kurallar, işletim süreçleri veya yazılım gibi diğer bileşenleri ile çok kuvvetli bağlantılar oluşturur. Gereksinim değişiklikleri ile problemleri halletmek olanaksızlaşmıştır. İhtiyaç belirtiminin geçerlilik irdelemesi çok pahalıdır. Elemanların her şeyi bilme olasılığı iyice azalmıştır.

COCOMO 81, bu model ve problem sınıfı saptamalarından sonra ortaya çıkan formüllerle tahmin hesaplama yolunu önerir (Hughes & Cotterell, 2009).

Çizelge 4-1'de, problem sınıflarına göre basit model için emek ve süre formülleri görülmektedir (Schach, 2007).

Çizelge 4-1. Basit COCOMO Modeli İçin Emek ve Süre Formülleri

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

Orta COCOMO modeli ile sistemin güvenilirlik, veri tabanı büyüklüğü, işletme ve kayıt sınırlandırmaları, personel özellikleri, kullanılan yazılım araçları gibi özelliklerinin hesaba katılması amaçlanmıştır.

Belirli bir dizi özelliğin, proje açısından etkileri ayrı ayrı ağırlıklandırılarak katsayılar ortaya çıkarılmıştır. Bu faktörler, ilgili özellik için düşük (<1), nominal (1) veya yüksek (>1) olarak saptanmıştır.

Katsayılar birbiri ile çarpıldığında Emek Ayarlama Katsayısı (Effort Adjustment Factor – EAK) bulunur. Bu katsayı 0.9 ile 1.4 arasında bir değer alır ve Çizelge 4-2'de gösterilen Orta COCOMO modeli formülleri kullanılarak emek hesaplaması sonuçlandırılır (Schach, 2007). Süre hesaplaması ise Basit COCOMO modelinde olduğu gibi yapılır.

Çizelge 4-2. Orta COCOMO Modeli İçin Emek Formülleri

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

Emek Ayarlama Katsayısı için sözü geçen etkenleri dört grupta toplayarak, Çizelge 4-3'te görüldüğü gibi sıralayabiliriz.

Çizelge 4-3. Emek Ayarlama Katsayısı

Maliyet Faktörleri		**Dereceler**					
		Çok Düşük	Düşük	Nominal	Yüksek	Çok Yüksek	Ekstra Yüksek
Ürün Nitelikleri							
RELY	Gereksinilen Yazılım Güvenilirliği	0,75	0,88	1	1,15	1,40	
DATA	Veritabanı Büyüklüğü		0,94	1	1,08	1,16	
CPLX	Ürün Karmaşıklığı	0,70	0,85	1	1,15	1,30	1,65
Bilgisayar Nitelikleri							
TIME	Çalışma Zamanı Kısıtı			1	1,11	1,3	1,66
STOR	Ana Depolama Kısıtı			1	1,06	1,21	1,56
VIRT	Sanal Makine Değişimi		0,87	1	1,15	1,30	
TURN	Bilgisayar Geri Dönüş Zamanı		0,87	1	1,05	1,15	
Personel Nitelikleri							
ACAP	Analist Yeteneği	1,46	1,19	1	0,86	0,71	
AEXP	Uygulama Deneyimi	1,29	1,13	1	0,91	0,82	
PCAP	Programcı Yeteneği	1,42	1,17	1	0,86	0,70	
VEXP	Sanal Makine Deneyimi	1,21	1,10	1	0,9		
LEXP	Programlama Dili Deneyimi	1,14	1,07	1	0,95		
Proje Nitelikleri							
MODP	Modern Programlama Uygulamaları	1,24	1,10	1	0,91	0,82	
TOOL	Yazılım Araçlarının Kullanımı	1,24	1,10	1	0,91	0,83	
SCED	Zaman Kısıtları	1,23	1,08	1	1,04	1,10	

Kaynak: (Schach, 2007)

Detaylı COCOMO modeli ise projenin evrelerine bağlı olarak süreç içinde değişiklikleri hesaba katarak arada bir kestirim hesaplamasını önerir. Bu model de zamana bağlılık temel değişikliktir. Projenin farklı evrelerinde emek yoğunluğu ve yapılacak işin karmaşıklığı değişecektir.

1981'de Boehm tarafından ortaya konan COCOMO 81 modeli daha sonra geliştirilmiş ve COCOMO II adını almıştır (Hughes & Cotterell, 2009).

4.2.2. COCOMO II

Barry Boehm ve arkadaşları, ilerleyen yıllar içinde COCOMO 81 üzerinde iyileştirmeler gerçekleştirerek, bir maliyet kestirim modeli olarak COCOMO II'yi yayınladılar (Hughes & Cotterell, 2009).

COCOMO II modeli, spiral yaşam döngü modelini destekler. Bunun yanında, veri tabanı programlama, prototipleme gibi yazılım geliştirmedeki farklı yaklaşımları da tanır. COCOMO II yaklaşımı, uzmanlar tarafından başlangıç değeri olarak düzenlenmiş çeşitli üs değerlerini ve çarpanlarını kullanır. Ancak, yürütülen projelerin performans bilgilerini içeren bir veri tabanı oluşturulur ve bu veri tabanı uzman kararlarının gerçek projelerden elde değerler ile devamlı olarak yer değiştirebilmesi için periyodik olarak analiz edilir. Önceden birlikte kullanılan süreç modellerinin, günümüzde geniş bir çeşitliliğe sahip olması yeni modellerin dikkate alınmasını gerektirir.

Daha önce de belirtildiği gibi, kestirimler sistem yaşam döngüsünün farklı aşamalarında gereklidir. COCOMO II, ayrıntılı kestirimi gerçekleştirmek üzere üç farklı model göz önünde bulundurularak tasarlanmıştır (Hughes & Cotterell, 2009):

- *Uygulama birleştirme modeli (application composition model):* Bu model sistemin dış özelliklerini tecrübe edecek kullanıcılar için tasarlanmıştır. Bunu sağlamak için genellikle prototiplerden yararlanılır. Bu noktada verimliliği yüksek uygulama geliştirme araçları kullanılarak oluşturulan küçük uygulamalar ile geliştirme durdurulabilir.

- *Erken tasarım modeli (early design model):* Bu model temel yazılım yapıları için tasarlanmıştır. Kapsamlı ve karmaşık sistemler de işlem yoğunluğu büyük olacağından, bu sistemler için performans önemlidir. Bunun yanında bu sistemlerin mimarisine daha fazla dikkat edilmesi gerekir.

- *Sonraki mimari model (post architecture model):* Belirlenen gereksinimlere göre geliştirilecek bir sistemi yaratmak için yazılım yapıları üzerinde ayarlama, düzenleme ve son değişikliklerin yapılmasıdır.

Uygulama birleştirme modeline göre emek kestirimi için, COCOMO II geliştiricileri nesne puanlarının sayılmasını önermiştir. Bu öneriyi yazılımın belirgin dış özelliklerinin sayılmasına yönelik, işlev puanı yaklaşımı izlemektedir. Giriş türleri gibi mantıksal olanlardan ziyade, ekranlar ve raporlar gibi uygulamanın fiziksel özelliklerinin bir noktada toplaması ile farklılık gösterir. Bunun daha çok prototipler üzerinden elde edilen gereksinimler de faydalı olduğu görülmektedir.

Erken tasarım modelinde, bir sistemin büyüklüğünü ölçmek için işlev puanı yaklaşımı önerilmektedir. İşlev puanı sayısı, kullanılacak programlama diline ilişkin katsayının işlev puanı ile çarpılması sonucu satır sayısına dönüştürülebilir. Bu durumda aşağıdaki model, adam-ay tahminini hesaplamak için kullanılabilir (Hughes & Cotterell, 2009).

$$pm = 2.94\ (size)^{sf} * \ (em_1) * \ (em_2) * \ldots * \ (em_n) \tag{11}$$

- *pm;* adam-ay biriminde emektir.
- *size,* işlev puanı sayısından türetilen KDSI ile ölçülür.
- *sf,* skala katsayısına ilişkin üs değeridir.

Skala katsayısı Denklem 12'de verilen şekilde hesaplanmaktadır (Hughes & Cotterell, 2009):

$$sf = 0.91 + 0.01 * \sum(üs\ faktör\ oranları) \tag{12}$$

Skala katsayısının değeri önemlidir. Bu katsayı değeri büyük yazılım projelerinde emek tahminini arttırmaktadır. Skala katsayısını hesaplamak için kullanılan üs faktörlerine ilişkin nitelikler aşağıda listelenmiştir:

- *Emsallerini Kullanma (Precedentedness – PREC):* Bu nitelik, mevcut projenin geçmiş benzer durumlara veya emsallerine göre derecesidir. Yeni sistem beraberinde büyük yenilikler getirmesine karşın, daha fazla belirsizlik içerir ve

belirlenen üs faktör değerleri daha yüksek olur.

- *Geliştirme Esnekliği (Development Flexibility – FLEX):* Bu nitelik, gereksinimlerin karşılanmasına yönelik farklı yolların sayısını yansıtır. Geliştirme esnekliği daha az iken, üs faktörünün değeri daha yüksektir.

- *Mimari/Risk Çözümleme (Architecture/Risk Resolution – RESL):* Bu nitelik, gereksinimler ile ilgili belirsizliklerin derecesini yansıtır. Gereksinimler değişme eğiliminde ise üs faktörü yüksek bir değer alacaktır.

- *Takım Uyumu (Team Cohesion – TEAM):* Bu nitelik, birbirine sıkı sıkıya bağlı küçük bir takıma karşı dağınık büyük bir takımın derecesini yansıtır.

- *Süreç Olgunluğu (Process Maturity – PMAT):* Bir yazılım daha yapısal ve düzenli bir şekilde üretildiğinde, bu yazılıma ilişkin belirsizlikler azalacak ve üs faktörü değeri de daha düşük olacaktır.

Bir proje için ölçek faktörlerinin her biri bir dizi değerlendirmelere göre; çok düşük, düşük, normal, yüksek, çok yüksek ve ekstra yüksek şeklinde sınıflandırılmıştır. Bu sınıflandırma Çizelge 4-4'te görülmektedir (Hughes & Cotterell, 2009).

Çizelge 4-4. COCOMO II Skala Faktörü Değerleri

Faktör	Çok Düşük	Düşük	Normal	Yüksek	Çok Yüksek	Ekstra Yüksek
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Kaynak: (Hughes & Cotterell, 2009)

4.2.3. Sınıf Puanı

Nesneye dayalı geliştirmede sınıf diyagramları, büyük ölçüde tasarım dokümanını temel alan niceliksel bilgilere sahiptir. Sınıf diyagramları, geliştirilen sistemin mantıksal blokları olan sınıf hiyerarşisini ve hedef sistemin yapısal işlevselliğini içerir. Sınıf Puanı (Class Points - CP) yaklaşımı, 1998'de ortaya atılmıştır (Kim, Lively, & Simmons, 2006) (Costagliola & Tortora, 2005).

Bu yaklaşım, tasarım dokümanını temel alarak nesne-tabanlı yazılım ürünlerinin büyüklüğünü tahmin etmek için tasarlanmıştır.

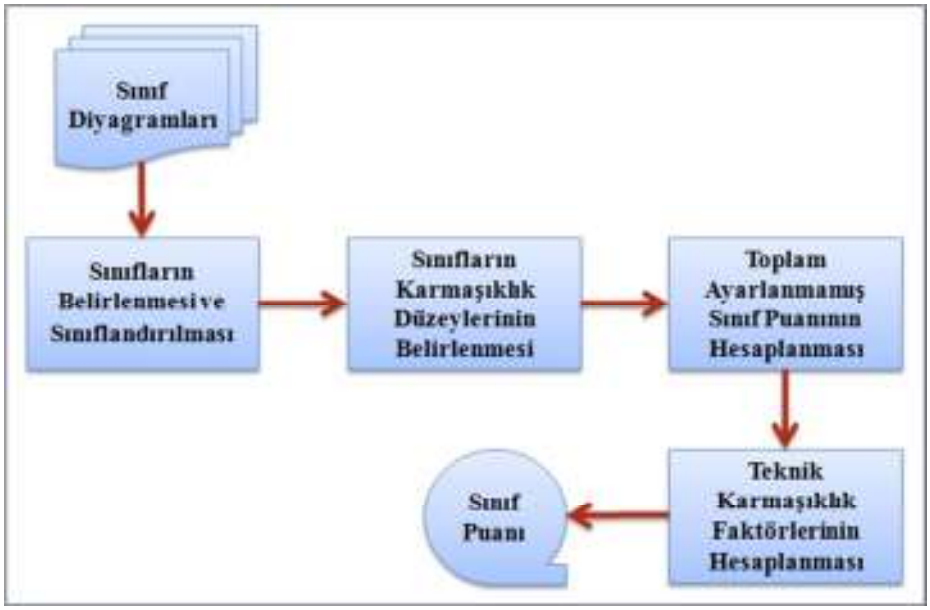
Sınıf puanı yönteminin altında yatan temel fikir, FP ölçümü tarafından gerçekleştirilen işlev saymaya benzerlikte bir program içindeki sınıfların ölçümüdür. Bu fikir prosedürel programlama içerisindeki temel programla elemanları olan fonksiyon ve prosedürlerin gözlemlenmesinden türemiştir. Hâlbuki nesne-tabanlı yaklaşımın mantıksal yapı taşı, birbirleri ile ilişkili olan ve gerçek dünyadaki nesnelere karşılık gelen sınıflardır.

Özellikle, bu yaklaşımla beraber CP_1 ve CP_2 olarak adlandırılan iki ölçüm ortaya atılmaktadır.

- CP_1 yazılım geliştirme sürecinin başında bir ön büyüklük tahmini yürütmek için kullanılmaktadır.

- Geliştirilecek sistemle ilgili daha fazla bilgi elde edildiğinde ise CP_2 uygulanarak, bu ön tahmin iyileştirilir (Costagliola & Tortora, 2005).

Şekil 4-1'de görüldüğü üzere, Sınıf Puanı kullanarak bir hedef sistemi ölçmek için dört önemli adım vardır (Costagliola & Tortora, 2005).



Şekil 4-1. Sınıf Puanı Hesaplama Adımları

Kaynak: (Costagliola & Tortora, 2005)

Sınıf Puanı'nı hesaplamak için, ilk adım sınıfları belirlemek ve bunları sınıflandırmaktır. Bunun için tasarım dokümanı analiz edilir. Tasarım dokümanı analiz edilirken, Problem Alan Türü (Problem Domain Type – PDT), İnsan Etkileşim Türü (Human Interaction Type – HIT), Veri Yönetim Türü (Data Management Type), Görev Yönetim Türü (Task Management Type – TMT) olmak üzere dört tür sistem bileşeni kullanılır. Bu sınıflandırma, sınıflar arasında daha kolay karşılaştırma sağlayacak ve karmaşık sistemlerin birbirlerinden ayırt edilmesinde yararlı olacaktır. Özetle, analiz hedef sistemin özellikleri kullanılarak yapılmaktadır.

Sınıfların belirlenmesi ve sınıflandırılmasından sonra, niteliklerin sayısı, servis isteklerinin sayısı ve dış metotların sayısıyla belirlenmiş olan her bir sınıf için karmaşıklık düzeyi belirlenmektedir.

Aslında, CP_1 içinde her bir sınıfın karmaşıklık düzeyi, Dış Metotların Sayısı (Number of External Methods – NEM) ve Servis İsteklerinin Sayısı (Number of Services Requested – NSR) üzerinden belirlenmektedir. NEM ölçüsü, genel olan yerel metotların sayısı ile verilen ve nesne-tabanlı bir sistemde tek bir sınıfın arayüz büyüklüğünü tahmin etmemize olanak sağlamaktadır. NSR sistem bileşenlerinin ara bağlantıları için bir ölçüm sağlar. Bu ölçütte tek bir sınıf için geçerlidir ve diğer sınıflar

için farklı servis isteklerinin sayısı ile belirlenmektedir. CP_2 içinde, yukarıdaki ölçümlerin yanı sıra, her bir sınıfın karmaşıklık düzeyini değerlendirmek üzere Niteliklerin Sayısı (Number Of Attributes – NOA) dikkate alınmaktadır.

Çizelge 4-5 ve Çizelge 4-6'da bir sınıfı karmaşıklık düzeyini hesaplamak için kullanılacak tablolar görülmektedir (Costagliola & Tortora, 2005).

Çizelge 4-5. CP_1 için Bir Sınıfın Karmaşıklık Düzeyini Değerlendirme Tablosu

	0 – 4 NEM	**5 – 8 NEM**	**≥ 9 NEM**
0 – 1 NSR	Düşük	Düşük	Orta
2 – 3 NSR	Düşük	Orta	Yüksek
≥ 4 NSR	Orta	Yüksek	Yüksek

Çizelge 4-6. CP_2 için Bir Sınıfın Karmaşıklık Düzeyini Değerlendirme Tablosu

0 – 2 NSR	**0 – 5 NOA**	**6 – 9 NOA**	**≥ 10 NOA**
0 – 4 NEM	Düşük	Düşük	Orta
5 – 8 NEM	Düşük	Orta	Yüksek
≥ 9 NEM	Orta	Yüksek	Yüksek

(a)

3 – 4 NSR	**0 – 4 NOA**	**5 – 8 NOA**	**≥ 9 NOA**
0 – 3 NEM	Düşük	Düşük	Orta
4 – 7 NEM	Düşük	Orta	Yüksek
≥ 8 NEM	Orta	Yüksek	Yüksek

(b)

≥ 5 NSR	**0 – 3 NOA**	**4 – 7 NOA**	**≥ 8 NOA**
0 – 2 NEM	Düşük	Düşük	Orta
3 – 6 NEM	Düşük	Orta	Yüksek
≥ 7 NEM	Orta	Yüksek	Yüksek

(c)

Her sınıfa bir karmaşıklık düzeyi atamadan önce, sınıfa bir ağırlık atamak için bu tür bilgiler kullanılmaktadır. Sonra, Toplam Düzeltilmemiş Sınıf Puanı (Total Unadjusted Class Point – TUCP) değeri, Denklem 13'te görüldüğü üzere ağırlıklı toplam olarak hesaplanmaktadır.

$$TUCP = \sum_{i=1}^{4} \sum_{j=1}^{3} x_{ij} * w_{ij} \tag{13}$$

$x_{ij} = j\ karmaşıklık\ düzeyinde\ i\ bileşen\ türü\ sayısı$

$w_{ij} = j\ karmaşıklık\ düzeyinde\ i\ bileşen\ türünün\ ağırlık\ değeri$

Çizelge 4-7'de Toplam Düzeltilmemiş Sınıf Puanı (Total Unadjusted Class Point – TUCP) değerini hesaplamak için kullanılacak tablo görülmektedir (Kusumoto, Matukawa, Inoue, & Hanabusa, 2004).

Çizelge 4-7. Toplam Düzeltilmemiş Sınıf Puanı Hesaplama Tablosu

Bileşen Türü	**Açıklama**	**Karmaşıklık**			
		Düşük	***Orta***	***Yüksek***	***Toplam***
PDT	Problem Alan Türü	… x 3 = …	… x 6 = …	… x 10 = …	…
HIT	İnsan Etkileşim Türü	… x 4 = …	… x 7 = …	… x 12 = …	…
DMT	Veri Yönetim Türü	… x 5 = …	… x 8 = …	… x 13 = …	…
TMT	Görev Yönetim Türü	… x 4 = …	… x 6 = …	… x 9 = …	…
Toplam Düzeltilmemiş Sınıf Puanı (TUCP):					

Çizelge 4-8'de teknik karmaşıklığı hesaplamak için kullanılacak tablo görülmektedir (Costagliola & Tortora, 2005).

Çizelge 4-8. Teknik Karmaşıklık Hesaplama Tablosu

	Sistem Karakteristiği	**E.D.**		**Sistem Karakteristiği**	**E.D.**
C1	Veri İletişimi	…	*C10*	Yeniden Kullanılabilirlik	…
C2	Dağıtık Fonksiyonlar	…	*C11*	Kurulum Kolaylığı	…
C3	Performans	…	*C12*	İşlem Kolaylığı	…
C4	Konfigürasyon Kullanım Yoğunluğu	…	*C13*	Çoklu Bölgeler	…
C5	İşlem Oranı	…	*C14*	Değişim Kolaylığı	…
C6	Online Veri Girişi	…	*C15*	Kullanıcı Uyarlanabilirliği	…
C7	Son Kullanıcı Verimliliği	…	*C16*	Hızlı Prototipleme	…
C8	Online Güncelleme	…	*C17*	Çoklu Kullanıcı Etkileşimi	…
C9	Karmaşık İşlem	…	*C18*	Çoklu Arayüzler	…
				Toplam Etki Derecesi (Total Degree of Influence – TDI):	

Teknik karmaşıklık faktörü, her biri 0 ila 5 arasında ölçeklendirilmiş 18 farklı hedef yazılım sistemi özelliğinin etki derecesi ile belirlenmiştir.

- Etkisi yok = 0
- Önemsiz Etki = 1
- Orta Karar Etki = 2
- Ortalama Etki = 3
- Önemli Etki = 4
- Güçlü Etki = 5

Hedef sistemin teknik karmaşıklığını hesaplamak için Denklem 14'ten yararlanılır.

$$TCF = 0.55 + (0.01 * TDI) \quad (14)$$

Sınıf Puanı (CP) değeri, FPA'da ki gibi genel sistem özellikleri göz önünde bulundurularak elde edilen değer (TCF) ile TUCP'nin çarpılması sonucu hesaplanmaktadır (Kim, Lively, & Simmons, 2006) (Costagliola & Tortora, 2005).

$$CP = TUCP * TCF \quad (15)$$

Bu ölçümün ayrıntılı prosedür ve denklemleri, Gennaro Costagliola ve Genoveffa Tortora yazmış oldukları "*Class Point: An Approach for the Size Estimation of Object-Oriented Systems*" konu başlıklı makalede tanımlanmıştır.

Ancak bu yaklaşım, genel yazılım projelerine uygulandığında zayıf noktalarının bulunduğu tespit edilmiştir. Sınıf Puanı sadece niteliklerin sayısını, gerekli hizmetlerin sayısını ve dış yöntemlerin sayısını sayar, problem ile ilgili bilgiler eksiktir.

Buna ek olarak, sınıfların sayısı, parametrelerin sayısı, ilişkilerin gerçekleştirilme sayısı, kullanıcıların sayısı, kalıtımın sayısı gibi hedef sistemlerin emek kestirimini etkileyen yararlı bilgiler vardır. Ayrıca, Sınıf Puanı TDI (Total Degree of Influence), TCF (Technical Complexity Factor), karmaşıklık düzeyi ve bileşen türü ile ilgili uzman kararı kullanır. Her bir faktörün belirlenen değeri, en son sonuçların varyansı oluşturulurken büyük oranda uzmanın kararına bağlı olacaktır.

Bu problemleri çözmek için, Sınıf Puanı ile ilgili yeni bir kavram önerilmiştir. Bu yaklaşım, bir sonraki bölümde anlatılan Use-Case Puanı ile benzer yararlara sahiptir. Uzman karar gibi öznel faktörlerin dışında sınıf diyagramları üzerine odaklanılmaktadır.

Sınıf Puanı'nın bu yeni tanımı, bir sistemin mimari karmaşıklığının anlaşılmasını sağlamıştır. Bu yeni tanım aşağıdaki gibidir (Periyasamy & Ghode, 2009):

1. Sınıfların Sayısı (Number of Classes – NOC): Sınıfların sayısı, sistemin mimari karmaşıklığını tanımlar ve emek kestirimi için büyük ölçüde ilişkili olan hedef sistemi tasarlamakta kullanılmaktadır.

$$NOC = \sum_{i=1}^{n} noc_i \tag{16}$$

2. Kalıtım İlişkilerinin Sayısı (Number of Inheritance Relationships – NOIR): Bu tanım, hedef sistemi tasarlamak için kaç tane kalıtım ilişkisinin kullanılmış olduğunu belirterek, sınıflar arasındaki ilişki niteliklerinden birini gösterir.

$$NOIR = \sum_{i=1}^{n} noir_i \tag{17}$$

3. Kullanım İlişkilerinin Sayısı (Number of Use Relationships – NOUR): Bu tanım, hedef sistemi tasarlamak kaç tane kullanım ilişkisinin kullanılmış olduğunu belirterek, sınıflar arasındaki ilişki niteliklerinden birini gösterir.

$$NOUR = \sum_{i=1}^{n} nour_i \tag{18}$$

4. Gerçekleştirme İlişkilerinin Sayısı (Number of Realize Relationships – NORR): Bu tanım, hedef sistemi tasarlarken kaç tane gerçekleştirme ilişkisi kullanılmış olduğunu belirterek, sınıflar arasındaki ilişki niteliklerinden birini gösterir.

$$NORR = \sum_{i=1}^{n} norr_i \tag{19}$$

5. Metotların Sayısı (Number of Methods – NOM): Hedef sistemi tasarlamak için kaç metot kullanılmıştır. Bu metotlar, büyük ölçüde emek kestirimi ile ilişkili olacaklardır ve aynı zamanda sistemin mimari karmaşıklığını tanımlarlar.

$$NOM = \sum_{i=1}^{n} nom_i \tag{20}$$

6. Parametrelerin Sayısı (Number of Parameters – NOP): Bu tanım, sınıf içerisinde verilen metotların kaç parametre kullandığını gösterir. Bu parametreler, büyük ölçüde emek kestirimi ile ilişkili olacaklardır ve aynı zamanda sistemin mimari karmaşıklığını tanımlarlar.

$$NOP = \sum_{i=1}^{n} nop_i \tag{21}$$

7. Sınıf Niteliklerinin Sayısı (Number of Class Attributes – NOCA): Hedef sistemi tasarlamak için kaç tane nitelik kullanılmıştır.

$$NOCA = \sum_{i=1}^{n} noca_i \tag{22}$$

8. Birlikteliklerin Sayısı (Number of Associations – NOASSOC): Hedef sistemi tasarlamak için kaç tane birliktelik kullanılmıştır.

$$NOASS = \sum_{i=1}^{n} noass_i \tag{23}$$

9. Sınıf Başına Ortalama Metot Sayısı (Average Number of Methods per Class – ANM_CLS): Hedef sistem içindeki sınıf başına ortalama metotların sayısıdır.

$$ANM\_CLS = \frac{NOM}{NOC} \tag{24}$$

10. Sınıf Başına Ortalama Parametre Sayısı (Average Number of Parameters per Class – ANP_CLS): Hedef sistem içindeki sınıf başına ortalama parametre sayısıdır.

$$ANP\_CLS = \frac{NOP}{NOC} \tag{25}$$

11. Sınıf Başına Ortalama Sınıf Niteliği Sayısı (Average Number of Class Attributes per Class – ANCA_CLS): Tasarım dokümanı içinde sınıf başına ortalama sınıf niteliği sayısıdır.

$$ANCA\_CLS = \frac{NOCA}{NOC} \tag{26}$$

12. Sınıf Başına Ortalama Birliktelik Sayısı (Average Number of Associations per Class – ANASSOC_CLS): Hedef sistem içindeki sınıf başına ortalama birliktelik sayıdır.

$$ANASS\_CLS = \frac{NOASS}{NOC} \tag{27}$$

13. Sınıf Başına Ortalama İlişkilerin Sayısı (Average Number of Relationships per Class – ANREL_CLS): Hedef sistem içindeki sınıf başına ortalama ilişkilerin sayıdır.

$$ANREL\_CLS = \frac{(NOIR + NOUR + NORR)}{NOC} \tag{28}$$

14. Class Points (CP): Hedef sistem içindeki Sınıf Puanları'dır.

$$\text{CP} = \sum (NOC + NOIR + NOUR + NORR + NOM + NOCA + NOASS) \tag{29}$$

Denklem (1) ile denklem (6), sınıf diyagramının yapısal karmaşıklığını tanımlayarak sınıf diyagramından temel bilgileri toplamak için kullanılmaktadır. Denklem (7) ile denklem (13), sınıf diyagramının yapısal karmaşıklığı hakkında ilgili bilgileri sağlamak için önceki denklemler ile kolaylıkla hesaplanabilir.

Son olarak Sınıf Puanı, 14. denklemde görüldüğü üzere bütün değerlerin toplanmasıyla hesaplanabilmektedir.

4.2.4. Use-Case Puanı

Use-Case Puanı yöntemi, nesne-tabanlı yöntemler kullanılarak geliştirilen bir projenin teknik ve çevresel karmaşıklığına ilişkin iki ayarlama faktörü ile use-case modellerini temel alan bir yazılım proje emek kestirim tekniğidir. 1993 yılında Gustav Karner tarafından ortaya atılmıştır (Periyasamy & Ghode, 2009) (Kim, Lively, & Simmons, 2006).

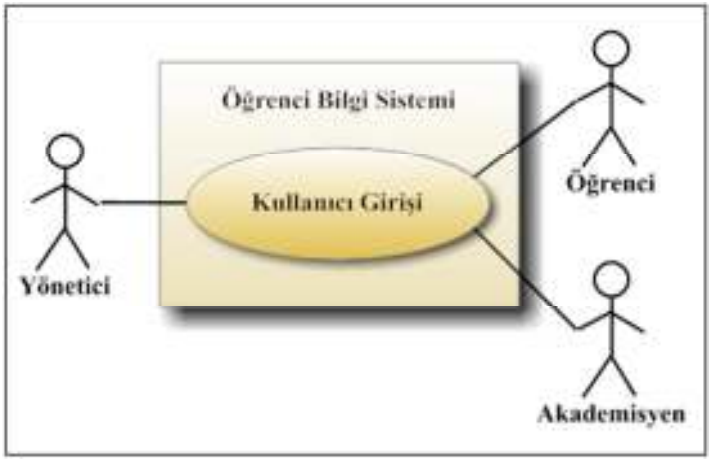
Bu yöntem İşlev Puanı Analizi (Function Point Analysis) ve Mark II İşlev Puanı Analizi (MK II Function Point Analysis)'nin bir uzantısıdır ve bu yöntemler gibi aynı felsefeyi temel almaktadır (Braz & Vergilio, 2006) (Anda, Benestad, & Hove, 2005).

Nesne-tabanlı yazılım üretiminde, use-case'ler sisteme ilişkin işlevsel gereksinimleri tanımlarlar. UCP yöntemini ele almadan önce bu yöntemin temelini oluşturan temel kavramlardan bahsetmek gerekir.

Use-case, bir sistemin sahip olması gereken ve etkileşime girildiğinde dış dünyaya sağlanan işlevsellik olarak tanımlanabilir. Genel anlamda use-case, bir sistem ile bu sistemin kullanıcısı yani aktörü arasındaki etkileşimi modelleyen yapıdır (Schach, 2007). Bir use-case modeli, use-case diyagramları ile bu use-case'lere ilişkin açıklamaların yer aldığı senaryolardan oluşmaktadır.

Use-case diyagramları, sistem içerisinde yer alan sınıflar arasındaki mesajlaşmayı ve sistemin zaman içerisindeki değişimini gösteren diyagramlardır. Bu diyagramlar aktörler ve use-case'ler arasındaki ilişkilerden oluşmaktadır. Use-case modelini oluşturan diğer önemli bir yapı ise senaryolardır. Senaryolar, aktörler tarafından başlatılan çeşitli olaylar dizisidir.

Şekil 4-2'de, Öğrenci Bilgi Sistemi içerisinde yer alan "Kullanıcı Girişi" use-case' sine ilişkin use-case diyagramı örneği görülmektedir. Verilen örnekte görüldüğü üzere yönetici, öğrenci ve akademisyen "Kullanıcı Girişi" use-case' sinin aktörleridir.



Şekil 4-2. Kullanıcı Girişi Use-case Diyagramı

"Kullanıcı Girişi" use-case'si için yazılan olası bir senaryo örneği Çizelge 4-9'da verilmektedir.

Çizelge 4-9. Kullanıcı Girişi Use-case'ine İlişkin Senaryo Örneği

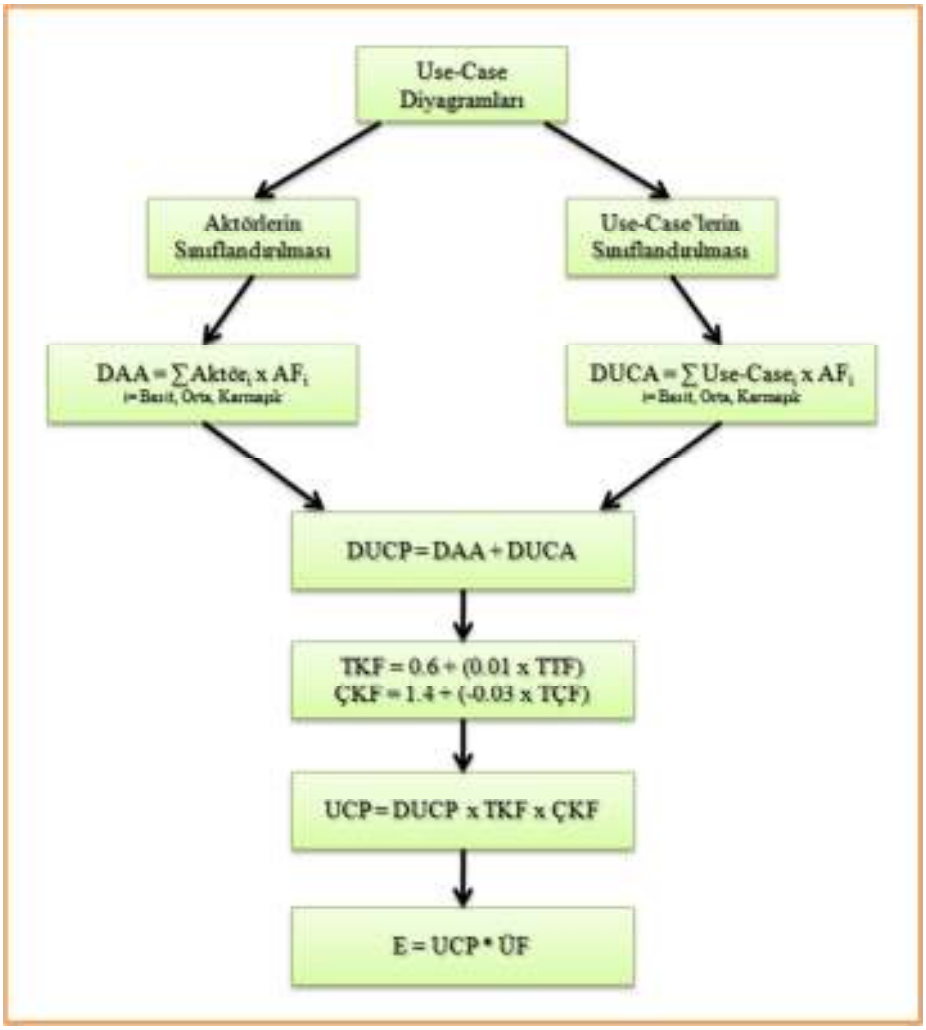
Kullanıcı Girişi use-case' i, kullanıcının sisteme giriş yapabilmesini sağlar. Kullanıcı;
1. "Kullanıcı Adı" ve "Şifre" sini sisteme girer. 2. Sistem tarafından kullanıcı bilgileri kontrol edilir. 3. "Kullanıcı Adı" ve "Şifre" doğru ise kullanıcı, sahip olduğu yetkiye göre ilgili sayfaya yönlendirilir.

Ayrıca bu use-case senaryonun gerçekleştirilmesine yönelik çizilen işbirliği diyagramı (collaboration diagram) Şekil 4-3'de görülmektedir.

Şekil 4-3. Kullanıcı Girişi Use-case Senaryosuna İlişkin İşbirliği Diyagramı

Use-case diyagramları, gereksinimlerin analiz edilmesi safhasında belirlenmiş hedef sistemin işlevsel davranışlarını incelerler. Sistemin büyüklüğü, mimarisi ve uygulama sahası biraz anlaşılmışken, use-case diyagramlarını temel alarak emek kestirimi gerçekleştirilebilir.

Bu nedenle use-case modeli ilk gelişim aşamasında geliştirilecek sistemin büyüklüğünü tahmin etmek için kullanılabilir. Şekil 4-4'te UCP hesaplama adımlarını temel alan ana akış diyagramı görülmektedir.



Şekil 4-4. UCP Emek Kestirim Adımları

Kaynak: (Kim, Lively, & Simmons, 2006)

4.2.4.1. Aktör ve Use-Case'lerin Sınıflandırılması

Use-case puanı, sistemin use-case analizinden sayılabilir. İlk adım, aktörleri basit, orta ve karmaşık tipte sınıflandırmaktır. Basit tipte bir aktör, tanımlı bir Uygulama Programı Arayüzüne (Application Programming Interface - API) sahip başka bir sistemi temsil eder. Orta tipte bir aktör, TCP/IP gibi bir protokol aracılığıyla haberleşen başka bir sistemi temsil eder. Karmaşık tipte bir aktör ise, bir Web sayfası veya bir Grafik Kullanıcı Arayüzü (Graphical User Interface - GUI) aracılığıyla karşılıklı etkileşen bir kullanıcıyı temsil eder. Her bir aktör tipine bir ağırlık faktörü atanmıştır.

Çizelge 4-10. Aktör Ağırlıkları Tablosu

Aktör Tipi	Ağırlık Faktörü
Basit	1
Orta	2
Karmaşık	3

Öncelikle toplam Düzeltilmemiş Aktör Ağırlığı (Unadjusted Actor Weights - UAW) hesaplanmaktadır. Toplam Düzeltilmemiş Aktör Ağırlığını (DAA) hesaplamak için Çizelge 4-10'da verilen sınıflara ait aktörler sayılır. Denklem 30'da görüldüğü üzere bu aktör sayıları kendi sınıfına ait Ağırlık Faktörü (AF) ile çarpılır ve çarpımlar toplanır.

$$\mathrm{DAA} = \sum_{i = \{Basit, Orta, Karmaşık\}} \mathrm{Aktör_i} * \mathrm{AF_i} \tag{30}$$

Her use-case, ikinci senaryo da dâhil, use-case açıklamasındaki işlemlerin sayısına bağlı olarak basit, orta veya karmaşık olarak tanımlanır. Bir işlem ya tamamen gerçekleştirilmiş ya da hiç gerçekleştirilmemiş faaliyetler kümesidir. İşlem sayısını sayma, use-case adımlarını sayarak yapılabilir. Karner, bağımlı ve genişletilen use-case'lerin sayılmamasını önermiştir, ama bunu niçin söylediği açık değildir.

Use-case karmaşıklığı Çizelge 4-11'de verilen biçimde ağırlıklandırılmış ve tanımlanmıştır.

Çizelge 4-11. Use-Case Ağırlık Tablosu

Use-Case Tipi	İşlem Sayısı (x)	Ağırlık Faktörü
Basit	$x \leq 3$	5
Orta	$4 \leq x \leq 7$	10
Karmaşık	$x \geq 8$	15

Use-case karmaşıklığının ölçmek için diğer mekanizma, belirli bir use-case'i gerçekleştiren sınıfların belirlenerek işlemlerin yerine kullanılabilecek bu sınıfların sayılmasıdır. Use-case'leri basit, orta ve karmaşık olmak üzere üç tipte sınıflandırabiliriz:

- Basit tipte bir use-case, basit bir kullanıcı arayüzüne sahiptir. Tek bir veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 3 veya daha az basamaktan oluşur ve tasarımı 5 veya daha az sınıf içerir.

- Orta tipte bir use-case, ortalama bir kullanıcı arayüzüne sahiptir. İki veya daha fazla veri tabanı nesnesi ile iletişim kurar. Normal (başarılı) senaryosu 4 ile 7 arasında basamaktan oluşur ve tasarım 5 ile 10 arasında sınıf içerir.

- Karmaşık tipte bir use-case, karmaşık bir kullanıcı arayüzüne sahiptir. Üç veya daha fazla veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 8 veya daha fazla basamaktan oluşur ve tasarımı 11 veya daha fazla sınıf içerir.

Düzeltilmemiş Use-Case Ağırlığını (Unadjusted Use-Case Weights - UUCW) elde etmek için, use-case'ler sayılır. Elde edilen use-case sayıları kendi sınıfına ait Ağırlık Faktörleri (AF) ile çarpılır.

Denklem 31'de görüldüğü üzere bu çarpımlar toplanarak Düzeltilmemiş Use-Case Ağırlığı (DUCA) hesaplanmış olur.

$$DUCA = \sum_{i = \{Basit, Orta, Karmaşık\}} Use - case_i * AF_i \quad (31)$$

Denklem 32'de de görüldüğü üzere, Düzeltilmemiş Aktör Ağırlığı (DAA) ile Düzeltilmemiş Use-Case Ağırlığı (DUCA)'nın toplanması sonucu Düzeltilmemiş Use-Case Puanı (Unadjusted Use-Case Points - UUCP) elde edilir. Düzeltilmemiş Use-Case Puanı, DUCP olarak kısaltılmıştır.

$$DUCP = DAA + DUCA \quad (32)$$

4.2.4.2. Teknik Karmaşıklık Faktörünün Hesaplanması

Use-case puanı yöntemi aynı zamanda FPA yönteminin Teknik Karmaşıklık Ayarlama (Technical Complexity Adjustment) faktörüne karşılık gelen teknik faktörler çarpanını kullanır. Yazılım geliştiren takım teknik faktörleri değerlendirir ve etki derecesine her bir faktöre 0 ile 5 arasında bir karmaşıklık değeri atanır. 0 değeri o faktörün projede etkisinin olmadığı, 3 değeri orta derecede etkili olduğu, 5 değeri ise yüksek derecede etkili olduğu anlamına gelir.

Teknik Karmaşıklık Faktörü (TKF), Denklem 33'de verilen formül kullanılarak bulunur.

$$TKF = 0.6 + (0.01 * \sum_{i=1}^{13} TF\_deger_i * TF\_agirlik_i) \quad (33)$$

Her bir teknik faktörün (TF) etki değeri, o faktörün Çizelge 4-12'de verilen ağırlığı ile çarpılır ve çarpımlar toplanır (Kim, Lively, & Simmons, 2006).

Çizelge 4-12. Teknik Faktör Ağırlık Tablosu

Teknik Faktör	Açıklama	Ağırlık Faktörü
T1	Dağıtık Sistem	2
T2	Yanıt veya Çıktı Performans Hedefleri	1
T3	Son Kullanıcı Verimliliği	1
T4	Karmaşık Dâhili İşlem	1
T5	Kodun Yeniden Kullanılabilirliği	1
T6	Kurulum Kolaylığı	0.5
T7	Kullanım Kolaylığı	0.5
T8	Taşınabilirlik	2
T9	Değişim Kolaylığı	1
T10	Eş Zamanlılık	1
T11	Özel Güvenlik Özellikleri İçerme	1
T12	Üçüncü Parti Yazılımlar için Doğrudan Erişim Sağlama	1
T13	Kullanıcı Eğitim Gerekliliği	1

4.2.4.3. Çevresel Karmaşıklık Faktörünün Hesaplanması

Use-case puanı hesaplanırken, teknik faktörlerin yanında çevresel faktörlerinde göz önünde bulundurulması gerekir. Use-case puanı hesaplanırken, programcı motivasyonu, kullanım kolaylığı gibi işlevsel olmayan gereksinimleri ölçmek amacıyla çevresel faktörler çarpanı kullanılır.

Yazılım geliştiren takım çevresel faktörleri değerlendirir ve her bir faktöre etki derecesine göre 0 ile 5 arasında bir etki değeri atar. 0 değeri o faktörün projede etkisinin olmadığı, 1 değeri yüksek derecede negatif etkili olduğu, 3 değeri orta derecede etkili olduğu, 5 değeri ise yüksek derecede pozitif etkili olduğu anlamına gelir.

Çevresel Karmaşıklık Faktörü (ÇKF), Denklem 34'teki formül kullanılarak bulunur.

$$ÇKF = 1.4 + (-0.03 * \sum_{i=1}^{8} EF\_deger_i * EF\_agirlik_i) \quad (34)$$

Her bir çevresel faktörün (EF) etki değeri, o faktörün Çizelge 4-13'te verilen ağırlığı ile çarpılır ve çarpımlar toplanır (Kim, Lively, & Simmons, 2006).

Çizelge 4-13. Çevresel Faktör Ağırlık Tablosu

Çevresel Faktör	Açıklama	Ağırlık Faktörü
E1	UML ile Tanışıklık	1.5
E2	Uygulama Deneyimi	0.5
E3	Nesneye-Tabanı Deneyim	1
E4	Lider Analist Yeteneği	0.5
E5	Motivasyon	1
E6	Sabit Gereksinimler	2
E7	Yarı-Zamanlı Çalışanlar	-1
E8	Zor Programlama Dili	-1

4.2.4.4. Use-Case Puanının Hesaplanması

Use-Case Puanı (UCP), Denklem 35'de görüldüğü üzere Düzeltilmemiş Use-Case Puanı (DUCP), Teknik Karmaşıklık Faktörü (TKF) ve Çevresel Karmaşıklık Faktörünün (ÇKF) çarpımı ile bulunur.

$$UCP = DUCP * TKF * ÇKF \quad (35)$$

4.2.4.5. Emeğin Hesaplanması

Bir yazılım şirketinde geliştirilen bir proje için gerekli Emek (E), Denklen 36'da görüldüğü üzere, Use-Case Puanı (UCP) ve o şirketin Üretkenlik Faktörünün (ÜF) çarpımı ile tahmin edilir. Üretkenlik Faktörü, geçmiş projeler göz önünde bulundurularak her use-case başına harcanan adam-saat oranıdır.

$$E = UCP * \text{Ü}F \tag{36}$$

Gustav Karner, bir projenin emek tahmini için, her bir UCP başına 20 adam-saatlik bir varsayılan değer önermiştir (Kim, Lively, & Simmons, 2006).

Steve Sparks ise, yapılan saha çalışmalarındaki deneyimleri göz önünde bulundurarak, Üretkenlik Faktörü' nün 15 ile 30 adam-saat arasında değişmesi gerektiğini belirtmiştir (Banerjee, 2001).

Schneider ve Winters, projenin değişkenliği ve personelin tecrübe seviyesini temel alan Karner'ın önerisi için bir iyileştirme yöntemi önermiştir. Önerilen bu yönteme göre; E1 ile E6 arasında 3 ve üzerinde puan alan çevresel faktörlerin sayısı sayılır, 3'ün altında puan alan E7 ile E8 arasındaki çevresel faktörlerin sayısına eklenir. Eğer toplam 2 veya daha az ise her use-case başına 20 adam-saat; eğer toplam 3 veya 4 ise her use-case başına 28 adam-saat; eğer toplam 4'den büyük ise her use-case başına 36 adam-saat önerilmiştir (Banerjee, 2001). Ancak, her use-case başına 36 adam-saatlik bir üretkenlik faktörü kullanıldığı durumda, projenin son derece riskli olduğu kabul edilebilir. Böyle bir durumda üretkenlik faktörünün ayarlanabilmesi için projede değişikliklerin yapılması gerektiği tavsiye edilir.

Aslında en doğru Üretkenlik Faktörü, zaman içerisinde use-case'lerin tasarlanması ve gerçekleştirilmesi için harcanmış olunan iş gücünün kaydedilmesi ile bulunabilir.

4.2.4.6. Use-Case Puanı Yöntemi ile İlgili Olarak Yapılan Çalışmalar

UCP ile ilgili olarak yapılan birkaç çalışma vardır. Bunlardan Bente Anda ve arkadaşları tarafından yapılan "*Estimating Software Development Effort based on Use-Cases – Experiences from Industry*" başlıklı çalışmada, UCP'yi temel alan emek kestirimine yönelik bir yöntemin üç endüstri çalışmasına uygulanması ve bu uygulama sonunda elde edilen sonuçlardan bahsedilmiştir (Anda, Dreiem, Sjøberg, & Jørgensen, 2001). Bu çalışma ile use-case'leri uygulayarak süreç tahminlerini iyileştirmek isteyen organizasyonlara yol göstermeye çalışılmıştır.

Use-case puanı yaklaşımının, genel yazılım projelerine uygulandığında zayıf noktalara sahip olduğu yönünde bir görüş "*An Effort Estimation by UML Points in the Early Stage of Development*" başlıklı makalede sunulmuştur. Çalışmada, sadece aktörler ve use-case'lerin sayısı sayıldığından, UCP bilgisinin eksik olduğu, bunun yanında teknik ve çevresel faktörler ile Düzeltilmemiş Aktör/Use-case Ağırlıklarına ilişkin uzman tahmini gerektiği belirtilmiştir (Kim, Lively, & Simmons, 2006).

Bu faktörlerin her biri için belirlenen değer, büyük oranda uzman görüşüne bağlı olacağından, en son sonuçlarda varyansın artacağı ifade edilmiştir. Bu sorunların üstesinden gelmek için, diyagrama daha fazla odaklanacak, uzman kararlarını hariç tutacak ve hesaplaması daha kolay olacak yeni bir yaklaşım önerilmiştir.

Çalışma içerisinde use-case diyagramlarının geliştirilecek sisteme ilişkin geliştiricilere dinamik bir bakış açısı sunduğu ve bu sayede geliştiricilerin gelişim safhasının başlangıcında daha fazla bilgiye sahip olabileceği ifade edilmiştir. Geliştiricilerin, hedef sistem hakkında yeterli bilgiye sahip olması, müşteri ile arasındaki kavramsal farkı azaltmasını ve daha kolay iletişim kurmasını sağlayacaktır.

UCP yöntemi ile ilgili birçok kavram ilgili makale içerisinde yeniden tanımlanmıştır (Kim, Lively, & Simmons, 2006):

1. Aktörlerin Sayısı (Number Of Actors – NOA): Aktörlerin sayısı, hedef sistemi geliştirmek için kullanılmaktadır.

$$NOA = \sum_{i=1}^{n} noa_i \tag{37}$$

2. Use-Case'lerin Sayısı (Number Of Use-Cases – NOUC): UML modelinden use-case'lerin sayısı. Bu emek kestirimini etkileyen ana bileşenlerden biridir.

$$NOUC = \sum_{i=1}^{n} nouc_i \tag{38}$$

3. Rollerin Sayısı (Number Of Roles – NOR): Bu aktör ile use-case arasındaki mantıksal işlevselliği gösterir. Bu rollerin ayrıntılı davranışı sonraki yazılım geliştirme aşamasında uygulanacaktır.

$$NOR = \sum_{i=1}^{n} nor_i \tag{39}$$

4. Use-Case Başına Ortalama Aktör Sayısı (Average Number of Actors per Use-Case – ANA_UC): Bu kavram aktörlerin sayısına göre her bir use-case'in karmaşıklık değerine ait bir oran ortaya koymaktadır.

$$ANA\_UC = \frac{NOA}{NOUC} \tag{40}$$

5. Use-Case Başına Ortalama Rollerin Sayısı (Average Number of Roles per Use Case – ANR_UC): Bu oran değeri, rollerin sayısına göre use-case'in karmaşıklığını gösterir.

$$ANR\_UC = \frac{NOR}{NOUC} \tag{41}$$

6. Use-Case Puanı (UCP): Bu tanım hedef sistemin use-case puanlarını sunar.

$$UCP = \sum (NOA + NOUC + NOR) \tag{42}$$

ANA_UC ve ANR_UC oran değerleri, UCP hakkında daha genel bir bakış açısı sağlamak için (1.), (2.) ve (3.) denklemler kullanılarak kolaylıkla hesaplanabilir.

Genel olarak, use-case diyagramı hedef sistemin yeterli bilgiye sahip olması ve geliştirici ile müşteri arasındaki kavramsal farkları azaltarak, birbirleriyle iletişim kurmalarını sağlamak için kullanılmaktadır. Bundan dolayı, emek kestirim modeline bir giriş olarak UCP değeri kullanılabilir. Örneğin; eğer NOA değeri yüksekse, bu sistem ile ortam arasında çok arayüz olduğu anlamına gelir. UCP, 6. denklemdeki bütün use-case puanlarının toplanmasıyla hesaplanır.

Use-case puanı ile ilgili olarak yapılan diğer bir çalışmada ise, use-case puanı yaklaşımı genişleterek yazılım maliyet tahmininin daha doğru yapıldığı savunulmuştur. Genişletilmiş UCP (e-UCP) yönteminde, aktörler, use-case'ler, use-case'ler ile aktörler arasındaki ilişkiler, aktörler arasındaki ilişkiler, use-case'ler arasındaki ilişkiler ve her bir use-case'in ayrıntılı senaryoları gibi bir use-case modelinin her bir bakış açısı kullanılmıştır (Periyasamy & Ghode, 2009). Bu bakış açıları içerisinden her bir use-case'e ait ayrıntılı senaryoların ele alınmasının önemli olduğu vurgulanmaktadır. Çünkü bir use-case diyagramının eksik yönleri bu sayede tamamlanmış oluyor. Karner'ın önermiş olduğu UCP yönteminden farklı olarak bu çalışmada, use-case senaryo tablosu ve use-case senaryosu içerisinde yer alan parametrelerin ağırlıklarına ilişkin bir tabloya yer verilmiştir. Bunun dışında, UCP yönteminden farklı olarak, aktör ve use-case sınıflandırmaları ve bunların ağırlık değerlerinin değiştirilmiş olduğu gözlenmiştir. Çizelge 4-14'te, e-UCP yönteminde kullanılan aktör tipleri ve bu aktör tiplerine ilişkin ağırlık değerleri verilmiştir (Periyasamy & Ghode, 2009).

Çizelge 4-14. e-UCP Yöntemi Aktör Ağırlık Sınıflandırması

Aktör Tipi	**Ağırlık Faktörü**
Çok Basit	0.5
Basit	1.0
Ortadan Daha Az	1.5
Orta	2.0
Karmaşık	2.5
Çok Karmaşık	3.0
Çok Daha Karmaşık	3.5

Çizelge 4-15'te e-UCP yöntemine göre yeniden düzenlenmiş use-case tipleri ve ağırlık faktörleri verilmektedir (Periyasamy & Ghode, 2009).

Çizelge 4-15. e-UCP Yöntemi Use-Case Ağırlık Sınıflandırması

Use-Case Tipi	İşlem Sayısı (x)	Ağırlık Faktörü
Basit	$x \leq 2$	0.5
Orta	$2 < x \leq 4$	1
Karmaşık	$4 < x \leq 6$	2
Daha Karmaşık	$x > 6$	3

e-UCP yönteminde, aktör ve use-case sınıflandırmaları ile bunlara ilişkin ağırlık faktörlerindeki değişiklikler dışında, geri kalan hesaplamalar aynen use-case puanı yöntemindeki gibi yapılmaktadır. Bu yöntem, use-case diyagramlarının daha ayrıntılı değerlendirilmesinden dolayı, aktör ve use-case sayılarının daha doğru tespit edilmesini sağlamaktadır. Aktör ve use-case sayılarının doğru tespit edilmesi ile yazılım büyüklük ve emek kestiriminin gerçeğe daha yakın tahmin edileceği konusunda bir yaklaşım sunulmuştur.

Ochodek ve arkadaşları tarafından yapılan "*Simplifying effort estimation based on Use-Case Points*" başlıklı çalışmada ise UCP yönteminin basitleştirilmesinin mümkün olup olmadığı araştırılmıştır. Bu amaçla UCP yöntemi içeresinde kullanılan değişkenlerin doğruluğunu karşılaştırmak için çapraz doğrulama prosedürü kullanılmıştır. Aralarında ilişki bulunduğu düşünülen çok sayıdaki değişken arasındaki ilişkilerin anlaşılmasını ve yorumlanmasını kolaylaştırmak ve bunun yanında değişken sayısını azaltmak amacıyla faktör analizi yapılmıştır. Bu çalışmada emeğin hesaplanması noktasında regresyon analizinden yararlanılmıştır (Ochodek, Nawrocki, & Kwarciak, 2011).

5. USE-CASE TABANLI YAZILIM EMEK KESTİRİM YÖNTEMİ

Bu bölümde tez kapsamında geliştirilen use-case tabanlı yeni bir yazılım emek kestirim yöntemi açıklanmaktadır. Bu yöntemin temelini çoklu doğrusal regresyon analizi oluşturmaktadır. Bu nedenle öncelikle regresyon analizi kısaca anlatılmış, sonrasında yeni kestirim yöntemi açıklanmıştır. Son olarak yeni yöntem gerçek proje verileri kullanılarak kalibre edilmiş ve yöntemin doğruluğu irdelenmiştir.

5.1. Regresyon Analizi

Regresyon analizi, iki ya da daha çok değişken arasındaki ilişkiyi incelemek amacıyla kullanılan bir analiz metodudur (İstatistikanaliz.com, 2011). Regresyon analizinde, değişkenler arasındaki ilişkiyi fonksiyonel olarak açıklamak ve bu ilişkiyi bir modelle tanımlayabilmek amaçlanmaktadır. Eğer tek bir değişken kullanılarak analiz yapılıyorsa buna tek değişkenli regresyon analizi, birden çok değişken kullanılarak analiz yapılıyorsa buna da çok değişkenli veya çoklu regresyon analizi denir. Bir kitlede gözlenen x ve y değişkenleri arasındaki doğrusal ilişki Denklem 43'deki ***"Basit Doğrusal Regresyon Modeli"*** ile verilebilir;

$$y = \beta_0 + \beta_1 x + \varepsilon \tag{43}$$

Burada;

- x: bağımsız değişken
- y: bağımlı değişken
- β_0: $x = 0$ olduğunda bağımlı değişkenin alacağı değer (kesim noktası)
- β_1: regresyon katsayısı
- ε : Hata terimi (Ortalaması=0 ve Varyansı=σ^2'dir)

Doğrusal modele dayalı maliyet kestiriminde en çok tercih edilen ve kullanılan kestirim yöntemi regresyon analizidir (Shepperd & Schofield, 1997).

Delany yazılım maliyet kestirimi üzerine yapmış olduğu çalışmada (Delany, Cunningham, & Wilke, 1999), kod satır sayısı, işlev puanları ve verimlilik oranı parametrelerine bağlı olarak yazılım emeğinin kestirilebileceğini ortaya koymuştur. Diğer bir ifade ile bir yazılım projesinin geliştirilmesinde bu üç parametre ile kestirim yapılabileceğini önermiştir. Benzer olarak Shepperd yazılım maliyet kestirim yöntemleri içerisinde en çok tercih edilen yöntemin doğrusal model olduğunu savunmaktadır (Shepperd & Schofield, 1997). Doğrusal regresyon analiz modeli kullanılarak, Denklem 44'de verilen maliyet kestirim fonksiyonunun katsayıları kestirilmektedir.

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \varepsilon \tag{44}$$

Denklem 44'de, x_1 kod satır sayısını, x_2 işlev puanlarının sayısını, x_3 verimlilik oranını ve y değişkeni ise adam-saat cinsinden kestirilen emek değerini ifade etmektedir. Denklemde yer alan α_0, α_1 ve α_2 katsayıları ise kestirilecek katsayıları, ε değişkeni ise normal dağılıma sahip hata parametresini ifade etmektedir.

Maliyet kestirimi için kullanılabilen doğrusal regresyon analizi yöntemi, emek kestirimi için de kullanılabilir ve hatta daha uygundur. Maliyet kestiriminde kullanılan bu yöntem, işlev puanını temel alan kestirim yöntemleri için daha uygundur. Ancak, işlev puanını temel alan kestirim yöntemleri işlev puanlarının sayılması sürecinde uzman gerektirmesi gibi bazı ciddi sınırlamalara sahiptir. Bu kapsamda işlev puanını temel alan kestirim yöntemlerinin aksine uzman görüşü gerektirmeyen, yazılım sisteminin büyüklüğünü ve karmaşıklığını UML diyagramlarını kullanarak ölçebilen Use-Case Puanı ve Sınıf Puanı gibi algoritmik emek kestirim yöntemlerinden yararlanılabilir. Özellikle nesne-tabanlı olarak geliştirilen yazılım projelerine ilişkin emek kestiriminde Use-Case Puanı ve Sınıf Puanı yöntemleri kullanılmaktadır.

5.2. Yöntem

Bu tez çalışmasında, Use-Case Puanı yönteminden yola çıkarak, yazılım emek kestirimi için çoklu doğrusal regresyon analizi tabanlı yeni bir emek kestirim yöntemi ortaya konulmuştur. Basit bir anlayışla başlanılan bu yeni emek kestirim yöntemi için öncelikle temel girdi olarak *büyüklük* değişkeni düşünülmüştür. Bu bağlamda yeni emek kestirim yöntemi için, bir yazılım projesine ilişkin işlevsel gereksinimleri ifade eden *Büyüklük* (x_1) değişkeni tanımlanmıştır. Mevcut proje verileri kullanılarak *Büyüklük* değişkeni değerleri hesaplanmış ve basit doğrusal regresyon analizi yapılmıştır. Ayrıca regresyon doğrusunun ne derece iyi bir tahminleyici olduğunu belirlemek için r^2 (belirleme katsayısı) ve F değerlerine bakılmıştır. Elde edilen değerler[1] göz önünde bulundurularak, emek kestirim modelini tanımlamak için sadece *büyüklük* değişkeninin yetmeyeceği, buna projeye etki eden *ayarlama faktörü* değişkeninin de katılması gerektiği sonucuna varılmıştır.

Bu bağlamda yeni emek kestirim yöntemi için, *Büyüklük* (x_1) değişkeninin yanında *Ayarlama Faktörü* (x_2) değişkeni tanımlanmıştır. *Ayarlama Faktörü* değişkeni, bir yazılım projesine ilişkin işlevsel olmayan gereksinimleri ifade etmektedir.

Bu yöntemde Denklem 45'te verilen regresyon denkleminin katsayıları elde edildiğinde emek kestirim fonksiyonu tanımlanmış olur.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon \tag{45}$$

Denklem 45'de, y adam-saat cinsinden tahmini emeği, x_1 yazılım projesinin *Büyüklüğünü*, x_2 ise yazılım projesine etki eden *Ayarlama Faktörünü* ifade etmektedir. Denklemde yer alan β_0, β_1 ve β_2 kestirilmesi gereken katsayılar, ε ise hata terimidir.

Büyüklük, daha önce de yazıldığı gibi çeşitli biçimlerde tanımlanabilmektedir. Burada Use-Case Puanı'na dayalı bir tanım benimsenmiştir. Denklem 46'da *Büyüklük* değişkeni hesaplanırken, Denklem 32'de verilen formül kullanılarak ortaya çıkan

[1] Basit doğrusal regresyon analizi sonucu elde edilen değerler, EK 5'de yer almaktadır.

Düzeltilmemiş Use-Case Puanı (DUCP) ile Denklem 33'de verilen formül kullanılarak ortaya çıkan Teknik Karmaşıklık Faktörü (TKF) çarpılır. DUCP ve TKF değişkenleri, UCP yöntemi içerisinde tanımlanmış olduğundan tekrar tanımlanmamıştır.

$$Büyüklük\ (x_1) = DUCP\ x\ TKF \tag{46}$$

Yazılım projesine etki eden *Ayarlama Faktörü*, yazılım projesi içerisindeki işlevsel olmayan gereksinimleri ifade etmektedir. *Ayarlama Faktörü* değişkeninin hesaplamasında kullanılan en önemli etkenlerden biri Üretkenlik Faktörü'dür. Eğer bir yazılım organizasyonu üretkenlik ile ilgili tarihsel proje verilerine sahip değilse, Üretkenlik Faktörü olarak Karner'ın önermiş olduğu varsayılan değeri yani her use-case puanı başına 20 adam-saat'i kullanmak zorunda kalır. Yazılım organizasyonu geçmiş projelere ilişkin tarihsel verilere sahipse, kestirim yapılacak projenin Üretkenlik Faktörü'nü belirlemek için bu verilerden yararlanılabilir. Bu noktada *Sonraki Üretkenlik Faktörü (sÜF)* olarak tanımladığımız yeni bir değişken devreye girmektedir. *Sonraki Üretkenlik Faktörü (sÜF)* Denklem 48'de verilen formül yardımıyla elde edilmektedir.

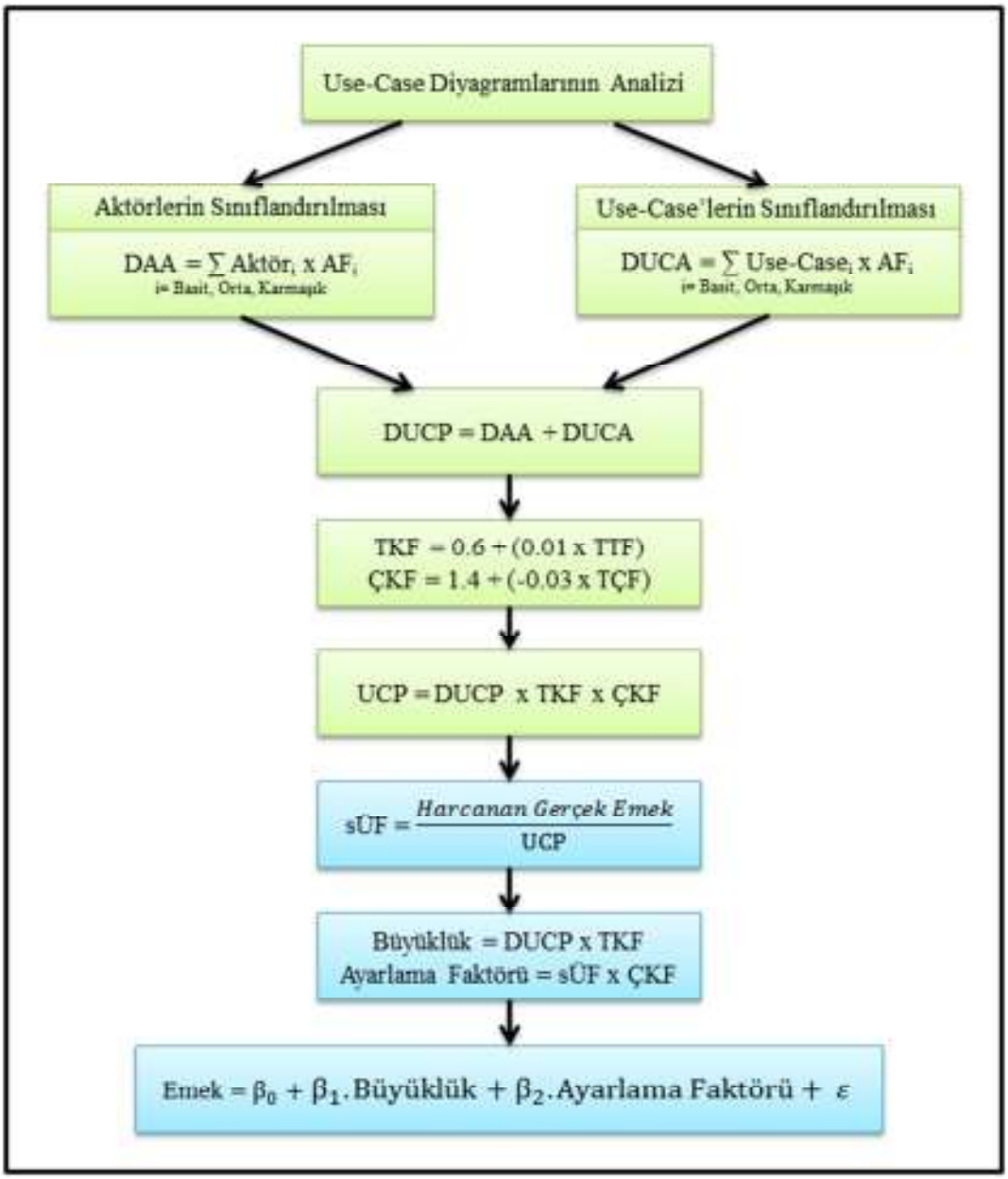
Denklem 47'de görüldüğü üzere *Ayarlama Faktörü* değişkeni hesaplanırken, Sonraki Üretkenlik Faktörü (sÜF) ile Denklem 34'te verilen formül kullanılarak ortaya çıkan Çevresel Karmaşıklık Faktörü (ÇKF) çarpılır. ÇKF değişkeni de UCP yöntemi içerisinde tanımlandığından yeniden tanımlanmamıştır. Böylece;

$$Ayarlama\ Faktörü\ (x_2) = sÜF\ x\ ÇKF \tag{47}$$

Yapılan proje sayısı arttıkça, ele alınan bir sonraki yazılım projesi için *Sonraki Üretkenlik Faktörü (sÜF);* Denklem 48'de de görüldüğü üzere, bir önceki projenin gerçekleştirilmesinde harcanan gerçek emeğin ilgili proje için belirlenen Use-Case Puanı'na (UCP) bölünmesiyle hesaplanabilir.

$$sÜF = \frac{Harcanan\ Gerçek\ Emek}{UCP} \tag{48}$$

Yöntemin veri seti üzerinde doğrulanması bölümünde bir yazılım projesi için harcanacak emek, yazılım projesine ilişkin büyüklük ve ayarlama faktörü parametrelerine bağlı olarak kestirilmektedir. Bunun yanında, Denklem 45 üzerinde veri seti kayıtları uygulanarak göreceli hatanın büyüklüğüne, ileride tanımlanacak R^2 ve F değerlerine bakılmaktadır. Öngörülen kriterlere göre sonuçlar karşılaştırılmaktadır. Ortaya konan bu yeni emek kestirim yöntemini Şekil 5-1'de görülen şekilde özetleyebiliriz.



Şekil 5-1. Use-Case Tabanlı Yeni Yazılım Emek Kestirim Yönteminin Adımları

5.2.1. Sonuçları Değerlendirme Kriterleri

Bir emek kestirim modelinin doğrulanması için genellikle kullanılan yöntemlerin başında oluşturulan emek kestirim fonksiyonunun veri kümesini ne kadar ifade edebildiği göz önünde bulundurulmaktadır. Bu da çoklu korelasyon katsayısı R^2 veya uyarlanmış R^2 değerlerine bakılarak karar verilmektedir. R^2 değeri 1'e ne kadar yaklaşırsa, regresyon denkleminin uygulandığı veri kümesini ne kadar iyi ifade ettiği anlaşılmaktadır.

Regresyon denklemi başarılı bulunsa bile, incelediğimiz bağımlı ve bağımsız değişkenler arasında gerçekten bir ilişki olup olmadığının belirlenmesi gerekir. Bunun için β_1 ve β_2 değerlerinin "0" a eşit olup olmadığı araştırmalıdır. β_1 ve β_2 yerine "0" konulursa regresyon denklemi

$$y = \beta_0 + \varepsilon$$

şekline dönüşür ki bu da bize açıkça y'nin (yani emeğin), Büyüklük (x_1) ve Ayarlama Faktörü (x_2) değişkenlerine bağlı olmadığını yani aralarında bir ilişki olmadığını gösterir.

$$H_0: \beta_1 = \beta_2 = 0$$

$$H_1: \text{En az biri diğerinden farklıdır}$$

hipotezini test etmek gerekir. Bunun içinde F dağılımından yararlanılır. H_0 hipotezi, hesaplanan F değerinin, F tablosundan *(p)* ve *(n-p-1)* serbestlik dereceleri ile bakılan kritik değer ile karşılaştırılarak test edilir. F değeri, Denklem 49'da verilen formül kullanılarak hesaplanmaktadır.

$$F = \frac{Regresyon\ Kareler\ Ortalaması}{Hata\ Kareler\ Ortalaması} \tag{49}$$

Hesaplanan $F_{değer}$, tabloda arzu edilen α seviyesinde bakılan F_{tablo} değerinden büyük çıkarsa H_0 hipotezi reddedilir. Bağımlı ve bağımsız değişkenler arasında bir ilişki olduğu, β_1 ve β_2'nin sıfıra eşit olmadığı söylenir.

Ayrıca bir çoklu regresyon modelinde bağımlı değişkenin her bağımsız değişkene bağlı olduğunu göstermek uygun olur. Yani değişken sayısını azaltarak istenilen doğrulukta kestirim yapılıp yapılmayacağının belirlenmesi gerekir. Bu tezde ortaya konan kestirim modelinde emeği yalnızca büyüklüğe bağlı olarak göstermenin yeterli olmayacağı, hatayı küçültmek için aynı zamanda ayarlama faktörüne ihtiyaç görüldüğü EK 5'de gösterilmiştir.

Bir emek kestirim modelinin başarısını ölçmede, Göreceli Hatanın Büyüklüğü (Magnitude of Relative Error – MRE) ölçütü kullanılabilmektedir (Boraso, Montangero, & Sedehi, 1996). MRE, yazılım projeleri ile ilgili emek tahmin modellerinin performansını değerlendirmek üzere yaygın olarak kullanılan bir değerlendirme ölçütüdür.

$$MRE = \frac{|Gerçek_{emek} - Tahmin_{emek}|}{Gerçek_{emek}} \tag{50}$$

Denklem 50'de verilen MRE, veri setindeki tüm yazılım projeleri için hesaplanmalıdır. Bunu yanında Denklem 51'e göre hesaplanan Göreceli Hata Büyüklüğü Ortalaması (Mean of Magnitude of Relative Error – MMRE) değeri modelin başarısı hakkında bize fikir verecektir (Finnie, Wittig, & Desharnais, 1997).

$$MMRE = \frac{1}{N}\sum_{i=1}^{n} MRE_i \tag{51}$$

Modelin doğruluğunda yaygın olarak kullanılan diğer bir ölçüm metodu Kök Hata Kareler Ortalaması (Root Mean Square Error - RMSE)'dır. RMSE, gerçek değerler ile modelleme sonucu elde edilen tahmin değerleri arasındaki farkın ölçüsü olarak sıklıkla kullanılır (Challagulla, Bastani, & I-Ling Yen, 2005). RMSE hata karelerini dikkate alması nedeniyle büyük kestirim hatalarına karşı ortalama hataya göre daha duyarlıdır. RMSE, Denklem 52'de verilen formülle hesaplanmaktadır.

$$RMSE = \sqrt{\frac{1}{N}\sum(\frac{Gerçek_{emek} - Tahmin_{emek}}{Gerçek_{emek}})^2} \tag{52}$$

5.3. Yöntemin Veri Seti Üzerinde Doğrulanması

Önerilen kestirim yönteminin gerçek veriler üzerinde denenmesi, doğrulanması ve kalibre edilmesi için piyasada belirli bir olgunluk düzeyine ve güvenilir verilere sahip yazılım firmaları ile temasa geçilmiştir. Bunlardan, yazılım geliştirme metodolojileri nesne-tabanlı olan dört ayrı yazılım firması ile anlaşılmış ve onlardan 10 yazılım projesine ilişkin veriler elde edilmiştir. Elde edilen veriler, Telekomünikasyon, Bankacılık, Finans, Kurumsal Kaynak Planlama (ERP), Muhasebe Yazılımları ile Savunma Teknolojileri alanında geliştirilmiş yazılım projelerine aittir. 10 yazılım projesine ilişkin elde edilen veriler, Ekler bölümünde ayrıntılı olarak sunulmaktadır. Ayrıca bu 10 yazılım projesi dışında, emek kestirim modeli, doğrulanma amacıyla Maltepe Üniversitesi'nin yazılım bölümü tarafından nesne-tabanlı olarak geliştirilen iki yazılım projesine ilişkin veriler üzerinde de uygulanmıştır.

Yazılım firmalarından verilerin elde edilmesi sürecinde, öncelikle A, B, C ve D firmalarının proje yöneticileri ile birebir olarak yüz yüze görüşülmüştür. Yazılım firmaları gizlilik prensipleri nedeniyle yazılım projelerine ilişkin dokümanları paylaşmak veya göstermek istememişlerdir. Proje yöneticileri ile yapılan görüşmelerde emek kestirim yönteminin doğrulanması için gerekli olan veriler birlikte belirlenmiştir. Bu veriler ya proje yöneticileri tarafından ya da proje yöneticisinin görevlendirdiği bir kişi tarafından sağlanmıştır. Bunun yanında D firması birebir olarak yazılım emek kestiriminde UCP yöntemini kullandığını belirtmiştir. Bu konuda iki yazılım projesine ilişkin olarak gerekli verileri sağlamıştır. Üniversitede geliştirilen yazılım projelerinden ise gerekli veriler; proje dokümanı, use-case modelleri, her bir use-case'in ayrıntılı tanımları ve senaryoları incelenerek elde edilmiştir. Ayrıca yazılımları geliştiren ekip üyeleri ile görüşmeler yapılmıştır.

Önce, 10 yazılım projesinin her biri için Gustav Karner'ın önermiş olduğu UCP yöntemi kullanılarak harcanılacak tahmini emek hesaplanmıştır. Her bir proje için, hesaplanan tahmini emek ile o proje için harcanan gerçek emek karşılaştırılmıştır. Ardından aynı veri seti üzerinde çoklu doğrusal regresyon analizi tabanlı yeni emek kestirim yöntemi uygulanmış ve sonuçlar karşılaştırılmıştır.

5.3.1. UCP Yönteminin Uygulanması

On yazılım projesinin her biri üzerinde UCP yöntemi uygulanmıştır. Her bir yazılım projesi için DAA, DUCA, TKF ve ÇKF değerleri Ekler kısmında yer alan proje verileri kullanılarak hesaplanmıştır. DAA değerleri Denklem 30, DUCA değerleri Denklem 31, TKF değerleri Denklem 33, ÇKF değerleri Denklem 34 ve UCP değerleri ise Denklem 35 kullanılarak elde edilmiştir.

Her bir proje için hesaplanan bu değerler Çizelge 5-1'de yer almaktadır. ÜF değeri, Gustav Karner'ın önermiş olduğu şekilde her use-case puanı başına 20 adam-saat olarak alınmıştır. Her proje için adam-saat cinsinden elde edilen tahmini emek ve gerçek emek ile tahmini emek arasındaki farklar Çizelge 5-1'de sunulmaktadır.

Çizelge 5-1. Yazılım Projelerine UCP Yönteminin Uygulanması

Projelere İlişkin Veriler	Yazılım Projeleri									
	P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**
DAA	12	10	15	13	14	8	12	7	3	6
DUCA	150	135	135	170	145	260	220	250	145	225
TKF	1,125	1,09	1,075	1,02	1,07	0,98	1,005	0,92	1,165	1,165
ÇKF	0,80	0,785	0,71	0,755	0,74	0,815	0,785	0,80	0,755	0,755
ÜF	20	20	20	20	20	20	20	20	20	20
Use-Case Puanı	146	124	114	141	126	214	183	189	130	203
Tahmini Emek	**2916**	**2481**	**2290**	**2800**	**2500**	**4280**	**3660**	**3780**	**2600**	**4060**
Gerçek Emek	**3240**	**2836**	**2584**	**3623**	**3648**	**5712**	**4692**	**5145**	**2761**	**4357**
Fark (%)	**%10**	**%13**	**%11**	**%23**	**%31**	**%25**	**%22**	**%27**	**%6**	**%7**

Çizelge 5-1'de görüldüğü üzere, UCP yöntemi kullanılarak hesaplanan emek tahminleriyle her bir yazılım projesi için harcanan gerçek emek miktarları karşılaştırıldığında, %17,5'luk bir ortalama hata görülmüş ve yöntemin iki yazılım projesi dışında gerçek emeğe yakın sonuçlar ortaya çıkarmadığı tespit edilmiştir.

5.3.2. Çoklu Doğrusal Regresyon Analizi Tabanlı Yöntemin Uygulanması

Aynı veri seti üzerine Use-Case Puanı yönteminden yola çıkarak, yazılım emek kestirimi için çoklu doğrusal regresyon analizi tabanlı yeni yöntem uygulanmış ve emek kestirim fonksiyonu elde edilmeye çalışılmıştır. Nesne-tabanlı bir yazılım projesinin gerçekleştirilmesi için harcanacak tahmini emek, *Büyüklük* ve *Ayarlama Faktörü* parametrelerine bağlı olarak kestirilmeye çalışılmıştır.

Öncelikle emek kestirim fonksiyonumuzda yer alan *Büyüklük* ve *Ayarlama Faktörü* değişken değerlerinin hesaplanması gerekir. Her bir yazılım projesine ilişkin *Büyüklük* değişkeni değerleri Denklem 46'da verilen formül kullanılarak hesaplanır. Ancak, öncelikle her bir yazılım projesi için *Düzeltilmemiş Use-Case Puanı (DUCP)*'nın ve *Teknik Karmaşıklık Faktörü (TKF)*'nün elde edilmesi gerekir. Denklem 32 kullanılarak DUCP değerleri, Denklem 33 kullanılarak da TKF değerleri hesaplanır.

$$Büyüklük\ (x_1) = DUCP\ x\ TKF \tag{46}$$

Hesaplanan DUCP ve TKF değerleri kullanılarak her bir yazılım projesi için elde edilen *Büyüklük* (x_1) değerleri Çizelge 5-2'de verilmiştir.

Çizelge 5-2. Büyüklük Değerleri Tablosu

Proje	DUCP	TKF	Büyüklük (x_1)
P1	162	1,125	182
P2	145	1,09	158
P3	150	1,075	161
P4	183	1,02	187
P5	159	1,07	170
P6	268	0,98	263
P7	232	1,005	233
P8	257	0,92	236
P9	148	1,165	172
P10	231	1,165	269

Her bir yazılım projesine ilişkin *Ayarlama Faktörü* değişkeni Denklem 48'de verilen formül kullanılarak hesaplanır. Ayarlama Faktörü değişkeninin hesaplanması için öncelikle her bir yazılım projesinin *Çevresel Karmaşıklık Faktörü (ÇKF)* ve *Sonraki Üretkenlik Faktörü (sÜF)* değerlerinin elde edilmesi gerekir. Denklem 34 kullanılarak ÇKF değerleri, Denklem 47 kullanılarak da *sÜF* değerleri hesaplanır.

$$s\text{ÜF} = \frac{Harcanan\ Gerçek\ Emek}{UCP} \tag{47}$$

Her bir yazılım projesi için Denklem 47 kullanılarak hesaplanan *Sonraki Üretkenlik Faktörü* değerleri Çizelge 5-3'te verilmektedir.

Çizelge 5-3. Yazılım Projelerinin Sonraki Üretkenlik Faktörü Değerleri

Proje	Harcanan Gerçek Emek	UCP	sÜF
P1	3240	146	22
P2	2836	124	23
P3	2584	114	23
P4	3623	141	26
P5	3648	126	29
P6	5712	214	27
P7	4692	183	26
P8	5145	189	27
P9	2761	130	21
P10	4357	203	21

Her bir yazılım projesi için *Sonraki Üretkenlik Faktörü* değerinin belirlenmesi ile beraber, bu yazılım projelerine ilişkin *Ayarlama Faktörü* (x_2) değişkenini Denklem 48'deki formülü kullanarak hesaplayabiliriz.

$$Ayarlama\ Faktörü\ (x_2) = s\text{ÜF}\ x\ \text{ÇKF} \tag{48}$$

Buna göre her bir yazılım projesine ilişkin *Ayarlama Faktörü* değişkeni değeri Çizelge 5-4'te verilmiştir.

Çizelge 5-4. Yazılım Projelerinin Ayarlama Faktörü Değerleri

Proje	sÜF	ÇKF	Ayarlama Faktörü (x_2)
P1	22	0,800	18
P2	23	0,785	18
P3	23	0,710	16
P4	26	0,755	20
P5	29	0,740	21
P6	27	0,815	22
P7	26	0,785	20
P8	27	0,800	22
P9	21	0,755	16
P10	21	0,755	16

Emek kestirim fonksiyonunu belirlemek için gerekli değişken değerlerini belirledikten sonra, veri seti üzerinde çoklu doğrusal regresyon analizine başlayabiliriz. Yazılım projelerine ilişkin çoklu doğrusal regresyon analizinde kullanılacak değişken değerleri Çizelge 5-5'te verilmektedir.

Çizelge 5-5. Çoklu Doğrusal Regresyon Analizinde Kullanılacak Değişken Değerleri

Proje	Büyüklük (x_1)	Ayarlama Faktörü (x_2)
P1	182	18
P2	158	18
P3	161	16
P4	187	20
P5	170	21
P6	263	22
P7	233	20
P8	236	22
P9	172	16
P10	269	16

Çoklu doğrusal regresyon analizi tabanlı emek kestirim modelinin doğrulanması amacıyla, önce bir yazılım projesi dışarıda bırakılıp, geri kalan dokuz yazılım projesi üzerine doğrusal regresyon analizi uygulanmıştır. Bir proje dışında, kalan diğer dokuz yazılım projesine ait büyüklük ve ayarlama faktörü değerleri kullanılarak emek kestirim fonksiyonuna ilişkin katsayılar belirlenmiştir. Katsayıların belirlenmesinin ardından dışarıda bıraktığımız yazılım projesi için emek kestirimi yapılmış ve emek kestirim fonksiyonun doğruluğuna bakılmıştır. Bu işlemler 10 yazılım projesinin her biri için sırayla yapılmıştır. Doğrusal regresyon analizi işlemleri Microsoft Excel 2010 içerisindeki bilimsel veri çözümleme araçları kullanılarak gerçekleştirilmiştir.

P1 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü değerleri kullanılarak çoklu doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3809 + 18\,x_1 + 209\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3809$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 209$ dur. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P1 yazılım projesi üzerinde uyguladığımızda Çizelge 5-6'da verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-6. P1 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	MRE
P1	3240	3229	0,003

P2 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3830 + 18\,x_1 + 210\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3830$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 210$ dur. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P2 yazılım

projesi üzerinde uyguladığımızda Çizelge 5-7'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-7. P2 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P2	2836	2794	0,014

P3 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -4030 + 19\,x_1 + 216\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -4030$, $\hat{\beta}_1 = 19$ ve $\hat{\beta}_2 = 216$ dır. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P3 yazılım projesi üzerinde uyguladığımızda Çizelge 5-8'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-8. P3 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P3	2584	2485	0,003

P4 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3866 + 18\,x_1 + 215\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3866$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 215$ dir. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P4 yazılım projesi üzerinde uyguladığımızda Çizelge 5-9'da verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-9. P4 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P4	3623	3800	0,048

P5 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3858 + 18\,x_1 + 213\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3858$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 213$ tür. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P5 yazılım projesi üzerinde uyguladığımızda Çizelge 5-10'da verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-10. P5 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P5	3648	3675	0,007

P6 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3621 + 18\,x_1 + 202\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3621$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 202$ dir. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P6 yazılım projesi üzerinde uyguladığımızda Çizelge 5-11'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-11. P6 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P6	5712	5557	0,027

P7 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3808 + 18\,x_1 + 209\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3808$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 209$ dur. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P7 yazılım projesi üzerinde uyguladığımızda Çizelge 5-12'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-12. P7 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P7	4692	4566	0,026

P8 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3787 + 18\,x_1 + 207\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3787$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 207$ dir. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P8 yazılım projesi üzerinde uyguladığımızda Çizelge 5-13'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-13. P8 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P8	5145	5015	0,025

P9 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3970 + 19\,x_1 + 215\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3970$, $\hat{\beta}_1 = 18$ ve $\hat{\beta}_2 = 215$ dir. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P9 yazılım projesi üzerinde uyguladığımızda Çizelge 5-14'te verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-14. P9 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P9	2761	2738	0,008

P10 projesi dışında geri kalan dokuz yazılım projesine ait büyüklük ve ayarlama faktörü parametreleri kullanılarak doğrusal regresyon analizi yapıldığında elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıdaki gibidir.

$$\hat{y} = -3588 + 20\,x_1 + 174\,x_2$$

Kestirilen katsayılar ise sırasıyla $\hat{\beta}_0 = -3588$, $\hat{\beta}_1 = 19$ ve $\hat{\beta}_2 = 174$ tür. Elde edilen bu katsayı değerlerine göre ortaya çıkan emek kestirim fonksiyonunu, P10 yazılım projesi üzerinde uyguladığımızda Çizelge 5-15'de verilen emek kestirimi ortaya çıkmıştır.

Çizelge 5-15. P10 Projesi İçin Elde Edilen Emek Kestirimi

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P10	4357	4307	0,011

Bir yazılım projesini dışarıda bırakıp, geri kalan dokuz yazılım projesi üzerinde regresyon analizi uygulanarak 10 yazılım projesinin her biri için emek kestirim fonksiyonu elde edilmiştir. Elde edilen emek kestirim fonksiyonları kullanılarak

dışarıda bırakılan her bir yazılım projesi için tahmini emek adam-saat cinsinden hesaplanmıştır. Tüm projeler için elde edilen tahmini emekler, MRE değerleri ve emek kestirim fonksiyonunun veri kümesini ne kadar ifade edebildiği göz önünde bulundurmamızı sağlayan R^2 ve F değerlerine ilişkin tablo Çizelge 5-16'da verilmiştir.

Çizelge 5-16. Yazılım Projelerine İlişkin Tahmini Emek, MRE, R^2 ve F Değerleri

Proje	Gerçek Emek	Tahmini Emek	MRE	R^2	F değeri
P1	3240	3229	0,003	0,993	400,97
P2	2836	2841	0,014	0,992	360,68
P3	2584	2485	0,003	0,993	425,44
P4	3623	3800	0,048	0,996	747,02
P5	3648	3675	0,007	0,993	419,78
P6	5712	5557	0,027	0,991	331,14
P7	4692	4566	0,026	0,992	392,73
P8	5145	5015	0,025	0,991	340,08
P9	2761	2738	0,008	0,993	400,92
P10	4357	4307	0,011	0,997	898,52

Çizelge 5-16'da görüldüğü üzere, her bir proje üzerinde yeni emek kestirim yönteminin uygulanması sonucu elde edilen emek tahminleri, UCP yöntemi ile elde edilen emek tahminlerinden daha iyidir. Ayrıca regresyon analizi sonucu elde edilen R^2 değerleri incelendiğinde bu değerlerin 1'e yaklaştığı görülmektedir. R^2 değerlerinin 1'e yaklaşması, regresyon denkleminin uygulandığı veri kümesini ne kadar iyi ifade ettiği sonucuna varmamızı sağlamıştır. Ayrıca regresyon analizi sonucu elde edilen F değerleri, F tablosunda $\alpha = 0.01$ için üstten 2 ve yandan 6 serbestlik derecesi ile bulunan değerlere göre karşılaştırıldığında, her proje için $F_{değer} > F_{tablo}$[2] olmasından dolayı H_0 hipotezi reddedilir. Elde edilen bu sonuçlar, regresyon denkleminde yer alan bağımlı ve bağımsız değişkenler arasında bir ilişki olduğunu ve elde edilen katsayılar ile doğru bir kestirim yapıldığını ortaya koymaktadır. Bunun yanında MMRE değeri 0.017 olarak elde edilmiştir. Elde edilen bu sonuçta kestirim yöntemimizin başarılı olduğunun bir göstergesidir. RMSE değeri ise 0.021 olarak elde edilmiştir. RMSE değerine göre de emek kestirim yönteminin başarılı olduğu söylenebilir.

[2] α=0.01 seviyesi için F dağılım tablosu, EK 6'da yer almaktadır.

Emek kestirim yönteminin doğrulanması açısından yapılan bir diğer çalışma ise, 10 yazılım projesinin tamamını regresyon analizine dâhil ederek, veri setinin tamamına uyacak bir emek kestirim fonksiyonunun elde edilmesi olmuştur. Regresyon analizi sonucu elde edilen emek kestirim fonksiyonu (regresyon denklemi) aşağıda tanımlanmıştır.

$$\hat{y} = -3835 + 18,4\, x_1 + 209,6\, x_2$$

Katsayılar sırasıyla $\hat{\beta}_0 = -3835$, $\hat{\beta}_1 = 18,4$ ve $\hat{\beta}_2 = 209,6$ dır. $\hat{\beta}_1$, *Büyüklük* değerinin katsayısı; $\hat{\beta}_2$, Ayarlama Faktörü değerinin katsayısı; $\hat{\beta}_0$ ise sabittir. Elde edilen bu kestirim fonksiyonu tüm veri seti üzerinde uyguladığında her bir yazılım projesi için gerekli tahmini emek ve ilgili proje için harcanan gerçek emek arasındaki fark Çizelge 5-17'de sunulmaktadır.

Çizelge 5-17. Yeni Kestirim Fonksiyonunun Veri Seti Üzerinde Uygulanması

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P1	3240	3283	0,013
P2	2836	2842	0,002
P3	2584	2478	0,041
P4	3623	3794	0,047
P5	3648	3691	0,012
P6	5712	5611	0,018
P7	4692	4640	0,011
P8	5145	5114	0,006
P9	2761	2680	0,029
P10	4357	4464	0,025

Çizelge 5-17'de de görüldüğü üzere, tanımlanan emek kestirim fonksiyonu yardımı ile mevcut proje verileri üzerinde yöntem doğrulanmış ve Gustav Karner'ın önermiş olduğu emek kestirim yöntemine göre daha iyi sonuçlar elde edilmiştir.

Emek kestirim yönteminin doğrulanması açısından elde edilen emek tahminleri ile ilgili projeler için harcanan gerçek emek miktarları karşılaştırılmıştır. Objektif bir doğrulama için R^2 değerine, F değerine, Ortalama Hata (Mean Error) ve Kök Hata Kareler Ortalaması (RMSE) değerlerine bakılmıştır.

Regresyon analizi sonucu R^2 değeri 0.993 çıkmıştır. R^2 değerinin 1'e yaklaşması, kestirim modelinin uygulandığı veri kümesini ne kadar iyi ifade ettiği sonucuna varmamızı sağlamıştır.

F değeri 474.06 olarak elde edilmiştir. Bu F değeri, F tablosunda $\alpha = 0.01$ için üstten 2 ve yandan 7 serbestlik derecesi ile bulunan değere göre karşılaştırıldığında, her proje için $F_{değer} > F_{tablo}$ olduğundan H_0 hipotezi reddedilir. Elde edilen bu sonuçlar, regresyon denkleminde yer alan bağımlı ve bağımsız değişkenler arasında bir ilişki olduğunu ve elde edilen katsayılar ile doğru bir kestirim yapıldığını ortaya koymaktadır.

MMRE değeri 0.020 olarak elde edilmiştir. Elde edilen bu sonuçta kestirim yöntemimizin başarılı olduğunun bir göstergesidir.

RMSE değeri 0.025 olarak elde edilmiştir. RMSE değerine göre de emek kestirim yöntemi başarılıdır.

Çoklu doğrusal regresyon analizi tabanlı emek kestirim yönteminin doğrulanması açısından mevcut veri setinin dışında, üniversitede geliştirilen iki yazılım projesi üzerinde de yöntem uygulanmıştır.

Nesne-tabanlı olarak geliştirilen bu projeler, A ve B olarak adlandırılmışlardır. A projesi, üniversitenin öğrenci işleri departmanı için geliştirilmiş Öğrenci Bilgi Sistemi, B projesi ise idari-mali işler departmanı için geliştirilmiş Muhasebe Otomasyon Sistemi'dir.

A ve B projeleri ile ilgili gerekli veri, proje dokümanları, use-case modelleri, use-case senaryoları, iterasyon planları ve proje üyeleri ile görüşülerek toplanmıştır.

Çizelge 5-18'de görüldüğü üzere, toplanan veriler üzerinde öncelikle UCP yöntemi uygulanarak, A ve B projeleri için gerekli emek tahmin edilmeye çalışılmıştır.

Çizelge 5-18. A ve B Yazılım Projeleri Üzerinde UCP Yönteminin Uygulanması

Proje ile İlgili Veriler	A Projesi	B Projesi
Düzeltilmemiş Aktör Ağırlığı	26	23
Düzeltilmemiş Use-Case Ağırlığı	415	200
Teknik Karmaşıklık Faktörü	1,005	1,04
Çevresel Karmaşıklık Faktörü	0,725	0,875
Üretkenlik Faktörü	20	20
Use-Case Puanı	321	203
Emek Tahmini (adam-saat)	6440	4040
Gerçek Emek (adam-saat)	9470	5680
Kestirim Hatası (%)	%32	%29

Çizelge 5-18'de verilen sonuçlar incelendiğinde, UCP yöntemi ile yapılan iş gücü tahmininin harcanan gerçek iş gücünden daha düşük olduğu görülmektedir. A projesi için UCP yöntemi ile yapılan iş gücü tahmini gerçek iş gücünden %32 daha düşük, B projesi için yapılan iş gücü tahmini ise gerçek iş gücünden %29 daha düşüktür.

Daha sonra, A ve B projelerine ortaya konulan yeni yöntem uygulanmıştır. Tanımlamış olduğumuz emek kestirim fonksiyonu kullanılarak her bir proje için gerekli emek tahmin edilmeye çalışılmıştır. Öncelikle Çizelge 5-19 ve 5-20'de görüldüğü üzere, emek kestirim fonksiyonunun *Büyüklük* (x_1) ve *Ayarlama Faktörü* (x_2) değerleri hesaplanmıştır.

Çizelge 5-19. A ve B Projelerinin Büyüklük Değerleri

Proje	DUCP	TKF	Büyüklük (x_1)
A	441	1,005	443
B	223	1,04	232

Çizelge 5-20. A ve B Projelerinin Ayarlama Faktörü Değerleri

Proje	sÜF	ÇKF	Ayarlama Faktörü (x_2)
A	30	0,725	22
B	28	0,875	25

Her iki yazılım projesi için gerekli *Büyüklük* (x_1) ve *Ayarlama Faktörü* (x_2) değerlerinin belirlenmesinin ardından, 10 yazılım projesi üzerinde regresyon analizi uygulanması sonucu daha önce elde ettiğimiz aşağıdaki emek kestirim fonksiyonunu (regresyon denklemini) kullanarak projeler için gerekli emek tahmin edilebilir.

$$\hat{y} = -3835 + 18,4\,x_1 + 209,6\,x_2$$

Çizelge 5-21. A ve B Projeleri İçin Yeni Yöntemle Yapılan Emek Tahmini

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
A	9470	8927	0,057
B	5680	5674	0,001

Tanımlanmış olan emek kestirim fonksiyonu (regresyon denklemi), A ve B projeleri üzerinde uygulandığında Çizelge 5-21'de verilen sonuçlar elde edilmiştir. Sonuçlar incelendiğinde, A projesi için yapılan iş gücü tahmininin gerçek iş gücünden %5 daha düşük, B projesi için yapılan iş gücü tahmininin ise hemen hemen gerçek emek ile aynı olduğu görülmektedir.

6. SONUÇ VE DEĞERLENDİRME

Başarılı bir yazılım projesinin temel hedefi, önceden belirlenen zamanda, önceden belirlenen bütçe ile müşterinin beklentilerini karşılayacak özelliklerde bir yazılım üretmektir. Birçok yazılım projesinin başarısızlığı, başlangıçta planlama aşamasında yapılan tahminlerin doğru çıkmamasından kaynaklanmaktadır. Hemen her zaman iyimser tarafta yapılan bu tahminler, süre aşımları ve bütçe aşımlarını doğurmakta, dolayısıyla başarısız diye nitelendirilen sonuçlar yaratmaktadır. Oysa teknik işler, muhtemelen olabilecek makul bir zamanda ve emekle bitirilmiş olabilir. Bir diğer deyişle baştan yapılan tahmin doğru olsaydı aynı süre ve bütçe içinde bitirilen iş, başarılı sayılacaktı.

Yazılım projelerinde bu başarı kriterlerinin sağlanması için yazılımda ölçüm yöntemlerinin kullanılması, yazılım sektöründe gittikçe önem kazanmaktadır. Yazılımların ölçülebilmesiyle birlikte yazılım organizasyonları ileride alacakları yeni projeler için, proje büyüklüğü, emek, harcanılan zaman ve bütçe gibi faktörleri belirleyebileceklerdir.

Bu noktada yazılım organizasyonlarının geliştirdikleri projelere ilişkin çeşitli verileri toplamaları, ileride alacakları projeler için kestirim yapabilme imkânı sağlayacaktır. Literatür çalışmaları ve deneyimler, yazılım büyüklük ve emek kestirimi için tarihsel verilere sahip olmanın çok önemli olduğunu göstermektedir.

Bu tez çalışmasının amacı, yazılım yöneticilerinin karşılaştığı bütçe ve süre aşımlarına bağlı problemlerin üstesinden gelebilmelerini sağlayacak yeni bir emek kestirim yöntemi ortaya koymaktır. Bu amaçla literatürde yer alan yazılım büyüklük ve emek kestirim yöntemleri incelenmiştir. Mevcut yöntemler içerisinden özellikle son yıllarda ön plana çıkan nesne-tabanlı yazılım üretim metodolojileri için uygun bir yöntem olan Use-Case Puanı (UCP) üzerinde ayrıntılı olarak çalışılmıştır. Gustav Karner tarafından ortaya atılan bu yöntem, bir projenin teknik ve çevresel karmaşıklığına ilişkin iki ayarlama faktörü ile use-case modelini temel alan bir yazılım

proje emek kestirim tekniğidir. Bu yöntem üzerinde yapılan çalışmalar sonucu aşağıdaki bulgular elde edilmiştir:

- Nesne-tabanlı üretim metodolojilerini kullanarak yazılım geliştirme işine yeni başlayacak organizasyonlara, bir yazılım projesinin büyüklüğünü ve harcanacak gerekli emeği tahmin etmek için UCP yönteminden yararlanmaları önerilir. Ancak, bu organizasyonlar tarihsel bir veri birikimine sahip olamadıkları için başlangıç aşamasında gerçekçi sonuçlar elde etmenin uzağında kalabilirler.
- Yazılım organizasyonları, yazılım projesinin işlevsel gereksinimlerini tanımlayan use-case modellerinin analizini iyi yapmalıdırlar. Bu analizleri ne kadar iyi yaparlarsa UCP yöntemi o kadar iyi sonuçlar verir.
- UCP yönteminin uygulanması aşamasında, use-case'ler ile aktörlerin karmaşıklık düzeylerinin belirlenmesine dikkat edilmelidir. Karner, kapsanan ve genişleten use-case'lerin (included and extending use-cases) dikkate alınmamasını önermiştir. Ancak bu tez çalışmasında bu öneri doğru bulunmamıştır.
- UCP yönteminde use-case'lerin ve aktörlerin karmaşıklık düzeyleri belirlenirken, aktörler arasındaki ilişkiler, use-case'ler arasındaki ilişkiler, use-case'ler ile aktörler arasındaki ilişkiler ve her bir use-case'in ayrıntılı senaryoları gibi bir use-case modelinin her bir bakış açısı ele alınmalıdır. Özellikle use-case'ler sınıflandırılırken kapsanan ve genişleten use-case'ler mutlaka göz önünde bulundurulmalıdır. Bu use-case'ler projelere ilişkin işlevsellik içermekte olduğundan fazladan emek harcaması gerektirir. Ayrıca, use-case'lere ilişkin alternatif senaryolar da göz önünde bulundurulmalıdır. Alternatif senaryolar, temel senaryolar gibi emek harcaması gerektirmektedir. Bu nedenle, alternatif senaryolar ayrı birer use-case olarak ele alınmalı ve sınıflandırmaya dâhil edilmelidir.
- Yapılan proje sayısı arttıkça, teknik ve çevresel faktörler güncellenmelidir. Bu sayede daha doğru iş gücü tahminleri yapılabilecektir.

Yapılan bu tespitlere istinaden UCP yönteminden yola çıkarak, yazılım emek kestirimi için çoklu doğrusal regresyon analizi tabanlı yeni bir emek kestirim yöntemi ortaya konulmuştur. Önerilen çözüm hem yazılım emek kestirim yöntemlerine farklı bir bakış açısı getirmekte hem de yazılım emek kestirim sürecini geliştirmeye çalışmaktadır.

Yöntemin doğrulanması amacıyla, Türkiye'deki dört farklı yazılım firması tarafından geliştirilen on yazılım projesine ait veriler toplanmıştır. Kestirim modeli, bu veriler kullanılarak geliştirilmiş, ayrıca bir üniversitenin yazılım bölümü tarafından geliştirilen iki yazılım projesi üzerinde de uygulanmıştır.

Toplanan bu veriler üzerinde önce UCP yöntemi uygulanmıştır. Elde edilen emek tahminleri ile bu projeler için harcanan gerçek emekler karşılaştırıldığında, UCP yöntemi sonuçlarının oldukça büyük hatalar verdiği tespit edilmiştir.

Ardından aynı veriler üzerinde, geliştirilen çoklu doğrusal regresyon analizi tabanlı yeni emek kestirim yöntemi uygulanmıştır. Her bir yazılım projesinin gerçekleştirilmesi için harcanacak tahmini emek, *Büyüklük* ve *Ayarlama Faktörü* parametrelerine bağlı olarak kestirilmeye çalışılmıştır. Elde edilen emek kestirim fonksiyonu kullanılarak on yazılım projesinin her biri için tahmini emekler hesaplanmıştır. Emek kestirim yönteminin doğrulanması açısından elde edilen emek tahminleri ile ilgili projeler için harcanan gerçek emekler karşılaştırılmıştır. Yapılan karşılaştırma sonucu, yöntemin kullanımı ile gerçeğe diğer yöntemden çok daha yakın emek tahmininde bulunulduğu tespit edilmiştir.

Ayrıca, objektif bir doğrulama için R^2 değerine, F değerine, Ortalama Göreceli Hata Büyüklüğüne (MMRE) ve Kök Hata Kareler Ortalaması (RMSE) değerlerine bakılmış ve aşağıdaki bulgular elde edilmiştir;

- Regresyon analizi sonucu R^2 değeri 0.993 çıkmıştır. R^2 değerinin 1'e yaklaşması, kestirim modelinin uygulandığı veri kümesini ne kadar iyi ifade ettiği sonucuna varmamızı sağlamıştır.
- F değeri 474.06 olarak elde edilmiştir. Bu F değeri, F tablosunda $\alpha = 0.01$ için üstten 2 ve yandan 7 serbestlik derecesi ile bulunan değere göre

karşılaştırıldığında, her proje için $F_{değer} > F_{tablo}$ olduğu görülmektedir. Elde edilen bu sonuçlar, regresyon denkleminde yer alan bağımlı ve bağımsız değişkenler arasında bir ilişki olduğunu ve elde edilen katsayılar ile doğru bir kestirim yapıldığını ortaya koymaktadır.

- MMRE değeri 0.020 olarak elde edilmiştir. Elde edilen bu sonuç kestirim yöntemimizin başarılı olduğunun bir göstergesidir.
- RMSE değeri 0.025 olarak elde edilmiştir. RMSE değerine göre de emek kestirim yöntemi başarılıdır.

Bu yeni yöntemin kullanımı ile beklenilenden daha iyi sonuçlar elde edilmiştir. Bunun nedenleri araştırıldığında;

- ilgili yazılım projeleri için use-case analizinin iyi yapıldığı,
- yazılım çalışmalarının deneyimli ve olgun bir ortamda yürütüldüğü,
- bu nedenle istatistiksel olarak stabil bir duruma ulaşıldığı,
- yazılım geliştirme süreçleri ve standartlarının dikkate alındığı,
- projelerin deneyimli proje yöneticileri tarafından yönetildiği

sonuçlarına varılabilinir.

Yöntem mevcut veri setinin dışında, elde edilen emek kestirim fonksiyonu kullanılarak üniversitede geliştirilen iki yazılım projesi üzerinde daha uygulanmıştır. Yöntem bu iki proje üzerinde de başarılı olmuştur.

Elde edilen bu sonuçlara göre, ortaya konan Use-Case Puanı tabanlı çoklu doğrusal regresyon analizi yöntemi ile yazılım emek kestiriminin yapılabileceği açıktır. Bu kestirim yöntemi ile gerçeğe yakın sonuçlar elde etmek için iyi bir use-case analizi yapılmalıdır. Bunun yanında proje deneyimi de gerekmektedir. 2-3 yazılım projesinden sonra stabil bir ortama erişildiğinde bu yöntem kolay bir şekilde kullanılabilir.

KAYNAKLAR

Abran, A., Desharnais, J., Oligny, S., St-Pierre, D., & Symons, C. (2003). *The COSMIC implementation guide for ISO/IEC 19761:2003, v2.2.* COSMIC FFP Measurement Manual.

Abran, A., Fagg, P., Meli, R., & Symons, C. (2002). *ISO Transposition and Clarification of the COSMIC FFP Method of Functional Sizing.* In Proceedings of the 12th International Workshop on Software Measurement (IWSM) (s. 33-42). Aachen: Shaker Publication.

Anda, B., Benestad, H., & Hove, E. (2005). *A Multiple-Case Study of Effort Estimation based on Use Case Points.* International Symposium on Empirical Software Engineering (ISESE'2005) (s. 407-416). Noosa, Australia: IEEE Computer Society.

Anda, B., Dreiem, H., Sjøberg, D., & Jørgensen, M. (2001). *Estimating Software Development Effort based on Use Cases – Experiences from Industry.* Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools (s. 487-502). London, UK: Springer-Verlag.

Antoniol, G., Fiutem, R., & Lokan, C. (2003). *Object-Oriented Function Points: An Empirical Validation.* Empirical Software Engineering, 8(3), 225-254.

Ayyıldız, M. (2007). *Yazılım Projeleri Ölçüm Sonuçları Veritabanının Oluşturulması ve Yeni Yazılım Projelerinin Maliyet Tahmininde Kullanımı, Doktora Tezi.* İstanbul: Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Banerjee, G. (2001). *Use Case Points: An Estimation Approach.* http://www.bfpug.com.br/Artigos/UCP/Banerjee-UCP_An_Estimation_Approach.pdf

Banker, R., Kauffman, R., Wright, C., & Zweig, D. (1994). *Automating Output Size and Reuse Metrics in a Repository-Based Computer Aided Software Engineering (CASE) Environment.* IEEE Transaction on Software Eng., 20(3), 169-186.

Boraso, M., Montangero, C., & Sedehi, H. (1996). *Software Cost Estimation: An Experimental Study of Model Performances, Technical Report: TR-96-22.* Italy: University of Pisa.

Braz, M., & Vergilio, S. (2006). *Software Effort Estimation Based on Use Cases.* Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06). Chicago, USA.

Cândido, E., & Sanches, R. (2004). *Estimating the Size of Web Applications By Using a Simplified Function Point Method.* WebMedia/LA-Web 2004 Conference. Ribeirão Preto, Brazil.

Challagulla, V., Bastani, F., & I-Ling Yen, P. (2005). *Empirical assessment of machine learning based software defect prediction techniques.* 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05) (s. 263-270).

Costagliola, G., & Tortora, G. (2005). *Class Point: An Approach for the Size Estimation of Object- Oriented Systems.* IEEE Transactions on Software Engineering, 31(1), 52-74.

Delany, S., Cunningham, P., & Wilke, N. (1999). *The Limits of CBR in Software Project Estimation.* German Workshop on Case-Based Reasoning.

Fenton, N. E. (1994, March). *Software Measurement: A Necessary Scientific Basis.* IEEE Transactions on Software Engineering, Vol.20(No.3), 199-206.

Fetcke, T., Abran, A., & Dumke, R. (2001). *A Generalized Representation for Selected Functional Size Measurement Methods.* Montreal, Canada: 11th International Workshop on Software Measurement.

Finnie, G., Wittig, G., & Desharnais, J. (1997). *A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks Case-Based Reasoning and Regression Models.* Journal of Systems Software, 39, 281-289.

Gencel, C. (2005). *An architectural dimensions based software functional size measurement method, PHd Thesis.* Ankara: The Graduate School of Informatics, The Middle East Technical University.

Gencel, C., & Demirors, O. (2008). *Functional Size Measurement Revisited.* ACM Transactions on Software Engineering and Methodology, 17(3), 1-36.

Hughes, B., & Cotterell, M. (2009). *Software Project Management, 5th Edition.* McGraw-Hill Education.

IFPUG. (1999). http://www.scribd.com/doc/46260527/IFPUG-Counting-Practices-Manual-4-1 adresinden alınmıştır.

IFPUG. (2009). IFPUG: International Function Point User Group: http://www.ifpug.org adresinden alınmıştır.

ISO/IEC 20926. (2003). *Software Engineering - IFPUG 4.1 Unadjusted FSM Method - Counting Practices Manual.* International Standarts Organization.

ISO/IEC TR 14143-3. (2003). *Information Technology - Software Measurement - Functional Size Measurement - Part 3: Verification of Functional Size Measurement Methods.* International Standarts Organization.

İstatistikanaliz.com. (2011). *İstatistik Analiz Yöntemleri: Regresyon Analizi*. Mayıs 2011 tarihinde Bilgi Eğitim Danışmanlık Websitesi: http://www.istatistikanaliz.com/regresyon_analizi.asp adresinden alındı.

Jones, C. (1991). *Applied Software Measurement: Assuring Productivity and Quality, 2nd ed.* McGraw-Hill.

Kim, S., Lively, W., & Simmons, D. (2006). *An Effort Estimation by UML Points in the Early Stage of Software Development.* The 2006 International Conference on Software Engineering Research and Practice (SERP'06), (s. 415-421). Las Vegas, Nevada, USA.

Kusumoto, S., Matukawa, F., Inoue, K., & Hanabusa, S. (2004). *Estimating Effort by Use Case Points: Method, Tool and Case Study.* Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), (s. 292- 299). Chicago, IL, USA.

Laird, L., & Brennan, M. (2006). *Software Measurement and Estimation, 1st ed.* John Wiley & Sons, Inc.

Laranjeira, L. (1990). *Software Size Estimation of Object Oriented Systems.* IEEE Transactions on Software Engineering, 1(5), 510-522.

Macit, İ. D. (2006). *Bölüm 1. Yazılım Mühendisliği ve Araçları (CASE), Endüstride Bilgisayar Uygulamaları Ders Notları.* Adana: Çukurova Üniversitesi Mühendislik-Mimarlık Fakültesi Endüstri Mühendisliği Bölümü.

NESMA. (1997). *Definitions and Counting Guidelines for the Application of Function Point Analysis, Version 2.0.* Netherlands Software Metrics Association (NESMA).

Ochodek, M., Nawrocki, J., & Kwarciak, K. (2011). *Simplifying effort estimation based on Use-Case Points.* Journal Information and Software Technology, 53(3), 200-213.

Özkanat, H. G. (2007). *Software Functional Size Measurement and Effort Prediction Using Use Case Points and Data Web Points for Web Applications, Master Thesis.* Istanbul: Marmara University, The Institute For Graduate Studies In Pure and Applied Sciences.

Pastor, O., Abrahão, S., Molina, J., & Torres, I. (2001). *A FPA - Like Measure for Object Oriented Systems from Conceptual Models.* In Proceedings of the 11th International Workshop on Software Measurement (IWSM'01) (s. 51-69). Montréal, Canada: Shaker Verlag.

Periyasamy, K., & Ghode, A. (2009). *Cost Estimation using extended Use Case Point (e-UCP) Model.* International Conferance on Computational Intelligence and Software Engineering (CiSE 2009). Wuhan, China.

Pressman, R. (2009). *Software Engineering: A Practitioner's Approach, 7th Ed.* McGraw-Hill.

Schach, S. R. (2007). *Object-Oriented & Classical Software Engineering, 7th Ed.* McGraw Hill.

Shepperd, M., & Schofield, C. (1997). *Estimating software project effort using analogy.* IEEE Transactions on Software Engineering, 23(11).

Sommerville, I. (2006). *Software Engineering, 8th ed.* Addison Wesley.

St-Pierre, D., Maya, M., Abran, A., & Desharnais, J. (1997). *Full Function Points: Counting Practices Manual, Technical Report.* Montréal, Canada: Université du Québec à Montréal.

Symons, C. (1991). *Software Sizing and Estimating: Mk II FPA.* John Wiley & Sons.

Symons, C. (2001). *Come Back Function Point Analysis (Modernized) – All is Forgiven!* In Proceedings of the 4th European Conference on Software Measurement and ICT Control, (s. 413-426). Germany.

UKSMA. (1998). *MK II Function Point Analysis Counting Practices Manual v1.3.1* . The United Kingdom Software Metrics Association (UKSMA).

ÖZGEÇMİŞ

Fatih YÜCALAR, 1980 yılında İstanbul / Kartal'da doğdu. Öğrenimlerini sırasıyla Ataköseoğlu İlkokulu, Semiha Şakir Ortaokulu ve Hayrullah Kefoğlu Süper Lisesi'nde tamamladı. 1998 yılında Maltepe Üniversitesi Mühendislik – Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümünü burslu olarak kazandı ve 2002 yılında bölüm birincisi olarak mezun oldu. 2003-2005 yılları arasında Maltepe Üniversitesi Bilişim Bölüm Başkanlığı'nda Yazılım Uzmanı olarak çalıştı. Aynı zamanda yaklaşık 6 ay kadar da vekâleten Bilişim Bölüm Başkan Yardımcılığı görevini yürüttü. 2005-2008 yılları arasında Maltepe Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak çalıştı. Haziran 2006'da Maltepe Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Yüksek Lisans programından mezun olarak yüksek lisans öğrenimini tamamladı. Eylül 2006'da, Trakya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Doktora programında doktora öğrenimi yapmaya hak kazandı. Eylül 2008'den beri Maltepe Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde Öğretim Görevlisi olarak yazılım mühendisliği, yazılım kalitesi, yazılım testi ve yazılım ölçümü alanlarında akademik çalışmalarına devam etmektedir.

EKLER

EK 1: A Firmasından 3 yazılım projesi ile ilgili olarak alınan veriler doğrultusunda Use-Case Puanı kullanılarak yapılan emek kestirimi

		PROJE		
PROJE İLE İLGİLİ BİLGİLER		**P1**	**P2**	**P3**
Aktör Sayıları				
Basit	Tanımlı bir Uygulama Programlama Arayüzüne (API) sahip başka bir sistemi temsil eder.	1	1	0
Orta	TCP/IP gibi bir protokol ile haberleşen başka bir sistemi temsil eder.	1	0	0
Karmaşık	Bir web sayfası veya GUI aracılığıyla karşılıklı etkileşen bir kullanıcıyı temsil eder.	3	3	5
Use-Case Sayıları				
Basit	Basit bir kullanıcı arayüzüne sahiptir. Tek bir veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 3 veya daha az basamaktan oluşur ve tasarımı 5 veya daha az sınıf içerir.	2	10	14
Orta	Ortalama bir kullanıcı arayüzüne sahiptir. İki veya daha fazla veri tabanı nesnesi ile iletişim kurar. Normal (başarılı) senaryosu 4 ile 7 arasında basamaktan oluşur ve tasarım 5 ile 10 arasında sınıf içerir.	8	7	5
Karmaşık	Karmaşık bir kullanıcı arayüzüne sahiptir. Üç veya daha fazla veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 8 veya daha fazla basamaktan oluşur ve tasarımı 11 veya daha fazla sınıf içerir.	4	1	1
Teknik Faktörler				
T1	Dağıtık Sistem	4	4	4
T2	Yanıt veya Çıktı Performans Hedefleri	4	3	4

T3	Son Kullanıcı Verimliliği	5	5	3
T4	Karmaşık Dâhili İşlem	3	3	3
T5	Kodun Yeniden Kullanılabilirliği	4	3	3
T6	Kurulum Kolaylığı	4	3	4
T7	Kullanım Kolaylığı	5	5	5
T8	Taşınabilirlik	3	3	3
T9	Değişim Kolaylığı	4	3	3
T10	Eş Zamanlılık	3	3	3
T11	Özel Güvenlik Özellikleri İçerme	4	4	5
T12	Üçüncü Parti Yazılımlar için Doğrudan Erişim Sağlama	3	4	2
T13	Kullanıcı Eğitim Gerekliliği	4	3	3
Çevresel Faktörler				
E1	UML ile Tanışıklık	3	3	3
E2	Uygulama Deneyimi	3	3	2
E3	Nesneye-Tabanı Deneyim	4	4	4
E4	Lider Analist Yeteneği	3	4	4
E5	Motivasyon	4	4	4
E6	Sabit Gereksinimler	4	4	3
E7	Yarı-Zamanlı Çalışanlar	0	0	0
E8	Zor Programlama Dili	2	2	2
Üretkenlik Faktörü		20	20	20
Use-Case Puanı		146	124	114
Emek Tahmini (adam-saat)		2916	2481	2290
Harcanan Gerçek Emek (adam-saat)		**3623**	**3648**	**5712**
Fark (1 – Tahmin / Gerçek)		0,10	0,13	0,11

EK 2: B Firmasından 3 yazılım projesi ile ilgili olarak alınan veriler doğrultusunda Use-Case Puanı kullanılarak yapılan emek kestirimi

		PROJE		
PROJE İLE İLGİLİ BİLGİLER		**P4**	**P5**	**P6**
Aktör Sayıları				
Basit	Tanımlı bir Uygulama Programlama Arayüzüne (API) sahip başka bir sistemi temsil eder.	1	1	0
Orta	TCP/IP gibi bir protokol ile haberleşen başka bir sistemi temsil eder.	0	2	1
Karmaşık	Bir web sayfası veya GUI aracılığıyla karşılıklı etkileşen bir kullanıcıyı temsil eder.	4	3	2
Use-Case Sayıları				
Basit	Basit bir kullanıcı arayüzüne sahiptir. Tek bir veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 3 veya daha az basamaktan oluşur ve tasarımı 5 veya daha az sınıf içerir.	3	3	3
Orta	Ortalama bir kullanıcı arayüzüne sahiptir. İki veya daha fazla veri tabanı nesnesi ile iletişim kurar. Normal (başarılı) senaryosu 4 ile 7 arasında basamaktan oluşur ve tasarım 5 ile 10 arasında sınıf içerir.	11	7	14
Karmaşık	Karmaşık bir kullanıcı arayüzüne sahiptir. Üç veya daha fazla veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 8 veya daha fazla basamaktan oluşur ve tasarımı 11 veya daha fazla sınıf içerir.	3	5	8
Teknik Faktörler				
T1	Dağıtık Sistem	4	3	4
T2	Yanıt veya Çıktı Performans Hedefleri	3	3	3
T3	Son Kullanıcı Verimliliği	3	4	4
T4	Karmaşık Dâhili İşlem	3	3	1
T5	Kodun Yeniden Kullanılabilirliği	1	3	1
T6	Kurulum Kolaylığı	3	3	3

T7	Kullanım Kolaylığı	5	5	5
T8	Taşınabilirlik	2	3	2
T9	Değişim Kolaylığı	3	3	2
T10	Eş Zamanlılık	4	5	4
T11	Özel Güvenlik Özellikleri İçerme	4	5	4
T12	Üçüncü Parti Yazılımlar için Doğrudan Erişim Sağlama	3	3	1
T13	Kullanıcı Eğitim Gerekliliği	2	2	2
Çevresel Faktörler				
E1	UML ile Tanışıklık	3	3	3
E2	Uygulama Deneyimi	3	3	2
E3	Nesneye-Tabanı Deneyim	4	4	4
E4	Lider Analist Yeteneği	3	4	4
E5	Motivasyon	4	4	4
E6	Sabit Gereksinimler	4	4	3
E7	Yarı-Zamanlı Çalışanlar	0	0	0
E8	Zor Programlama Dili	2	2	2
Üretkenlik Faktörü		20	20	20
Use-Case Puanı		141	126	214
Emek Tahmini (adam-saat)		2916	2481	2290
Harcanan Gerçek Emek (adam-saat)		**3623**	**3648**	**5712**
Fark (1 – Tahmin / Gerçek)		0,23	0,31	0,25

EK 3: C Firmasından 2 yazılım projesi ile ilgili olarak alınan veriler doğrultusunda Use-Case Puanı kullanılarak yapılan emek kestirimi

		PROJE	
PROJE İLE İLGİLİ BİLGİLER		**P7**	**P8**
Aktör Sayıları			
Basit	Tanımlı bir Uygulama Programlama Arayüzüne (API) sahip başka bir sistemi temsil eder.	0	0
Orta	TCP/IP gibi bir protokol ile haberleşen başka bir sistemi temsil eder.	3	2
Karmaşık	Bir web sayfası veya GUI aracılığıyla karşılıklı etkileşen bir kullanıcıyı temsil eder.	2	1
Use-Case Sayıları			
Basit	Basit bir kullanıcı arayüzüne sahiptir. Tek bir veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 3 veya daha az basamaktan oluşur ve tasarımı 5 veya daha az sınıf içerir.	13	16
Orta	Ortalama bir kullanıcı arayüzüne sahiptir. İki veya daha fazla veri tabanı nesnesi ile iletişim kurar. Normal (başarılı) senaryosu 4 ile 7 arasında basamaktan oluşur ve tasarım 5 ile 10 arasında sınıf içerir.	5	14
Karmaşık	Karmaşık bir kullanıcı arayüzüne sahiptir. Üç veya daha fazla veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 8 veya daha fazla basamaktan oluşur ve tasarımı 11 veya daha fazla sınıf içerir.	7	2
Teknik Faktörler			
T1	Dağıtık Sistem	4	3
T2	Yanıt veya Çıktı Performans Hedefleri	3	4
T3	Son Kullanıcı Verimliliği	4	3
T4	Karmaşık Dâhili İşlem	4	3
T5	Kodun Yeniden Kullanılabilirliği	2	2
T6	Kurulum Kolaylığı	3	3

T7	Kullanım Kolaylığı	4	3
T8	Taşınabilirlik	1	1
T9	Değişim Kolaylığı	3	3
T10	Eş Zamanlılık	2	0
T11	Özel Güvenlik Özellikleri İçerme	4	2
T12	Üçüncü Parti Yazılımlar için Doğrudan Erişim Sağlama	2	2
T13	Kullanıcı Eğitim Gerekliliği	3	2
Çevresel Faktörler			
E1	UML ile Tanışıklık	3	3
E2	Uygulama Deneyimi	4	3
E3	Nesneye-Tabanı Deneyim	4	4
E4	Lider Analist Yeteneği	4	4
E5	Motivasyon	4	4
E6	Sabit Gereksinimler	3	3
E7	Yarı-Zamanlı Çalışanlar	0	0
E8	Zor Programlama Dili	2	2
Üretkenlik Faktörü		20	20
Use-Case Puanı		183	189
Emek Tahmini (adam-saat)		3660	3780
Harcanan Gerçek Emek (adam-saat)		**4692**	**5145**
Fark (1 – Tahmin / Gerçek)		0,22	0,27

EK 4: D Firmasından 2 yazılım projesi ile ilgili olarak alınan veriler doğrultusunda Use-Case Puanı kullanılarak yapılan emek kestirimi

		PROJE	
PROJE İLE İLGİLİ BİLGİLER		**P9**	**P10**
Aktör Sayıları			
Basit	Tanımlı bir Uygulama Programlama Arayüzüne (API) sahip başka bir sistemi temsil eder.	3	4
Orta	TCP/IP gibi bir protokol ile haberleşen başka bir sistemi temsil eder.	0	1
Karmaşık	Bir web sayfası veya GUI aracılığıyla karşılıklı etkileşen bir kullanıcıyı temsil eder.	0	0
Use-Case Sayıları			
Basit	Basit bir kullanıcı ara yüzüne sahiptir. Tek bir veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 3 veya daha az basamaktan oluşur ve tasarımı 5 veya daha az sınıf içerir.	0	25
Orta	Ortalama bir kullanıcı ara yüzüne sahiptir. İki veya daha fazla veri tabanı nesnesi ile iletişim kurar. Normal (başarılı) senaryosu 4 ile 7 arasında basamaktan oluşur ve tasarım 5 ile 10 arasında sınıf içerir.	7	4
Karmaşık	Karmaşık bir kullanıcı ara yüzüne sahiptir. Üç veya daha fazla veri tabanı nesnesiyle iletişim kurar. Normal (başarılı) senaryosu 8 veya daha fazla basamaktan oluşur ve tasarımı 11 veya daha fazla sınıf içerir.	5	4
Teknik Faktörler			
T1	Dağıtık Sistem	4	4
T2	Yanıt veya Çıktı Performans Hedefleri	4	4
T3	Son Kullanıcı Verimliliği	5	5
T4	Karmaşık Dâhili İşlem	3	3
T5	Kodun Yeniden Kullanılabilirliği	4	4
T6	Kurulum Kolaylığı	3	3

T7	Kullanım Kolaylığı	4	4
T8	Taşınabilirlik	4	4
T9	Değişim Kolaylığı	4	4
T10	Eş Zamanlılık	4	4
T11	Özel Güvenlik Özellikleri İçerme	5	5
T12	Üçüncü Parti Yazılımlar için Doğrudan Erişim Sağlama	5	5
T13	Kullanıcı Eğitim Gerekliliği	3	3
Çevresel Faktörler			
E1	UML ile Tanışıklık	4	4
E2	Uygulama Deneyimi	3	3
E3	Nesneye-Tabanı Deneyim	4	4
E4	Lider Analist Yeteneği	4	4
E5	Motivasyon	4	4
E6	Sabit Gereksinimler	4	4
E7	Yarı-Zamanlı Çalışanlar	0	0
E8	Zor Programlama Dili	4	4
Üretkenlik Faktörü		20	20
Use-Case Puanı		183	189
Emek Tahmini (adam-saat)		3660	3780
Harcanan Gerçek Emek (adam-saat)		**4692**	**5145**
Fark (1 – Tahmin / Gerçek)		0,06	0,07

EK 5: Basit Doğrusal Regresyon Analizi Sonucu Elde Edilen Değerler

Yeni yazılım emek kestirim yönteminin ortaya konulması noktasında, öncelikle yazılım emek tahmini için sadece büyüklük değişkeninin yetip yetmeyeceği sorusuna cevap aranmıştır. Bu amaç doğrultusunda basit doğrusal regresyon analizi kullanılmıştır. Elde edilen analiz sonucu, ikinci bir değişkene yani ayarlama faktörüne ihtiyaç olduğunu ortaya koymuştur. Basit ve çoklu doğrusal regresyon analizi sonucu elde edilen değerler aşağıda sunulmuştur.

Basit doğrusal regresyon analizi sonucu elde edilen regresyon denklemi aşağıda verilmiştir.

$$\hat{y} = -691 + 22,4\, x_1$$

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P1	3240	3390	0,046
P2	2836	2848	0,004
P3	2584	2920	0,130
P4	3623	3489	0,037
P5	3648	3119	0,145
P6	5712	5191	0,091
P7	4692	4530	0,034
P8	5145	4604	0,105
P9	2761	3170	0,148
P10	4357	5336	0,225

- Regresyon analizi sonucu r^2 değeri 0.793 çıkmıştır.
- F değeri 30.672 olarak elde edilmiştir.
- MMRE değeri 0.097 olarak elde edilmiştir.
- RMSE değeri 0.114 olarak elde edilmiştir.

Çoklu doğrusal regresyon analizi sonucu elde edilen regresyon denklemi aşağıda verilmiştir.

$$\hat{y} = -3835 + 18,4\, x_1 + 209,6\, x_2$$

Proje	Gerçek Emek	Tahmini Emek	Fark (MRE)
P1	3240	3283	0,013
P2	2836	2842	0,002
P3	2584	2478	0,041
P4	3623	3794	0,047
P5	3648	3691	0,012
P6	5712	5611	0,018
P7	4692	4640	0,011
P8	5145	5114	0,006
P9	2761	2680	0,029
P10	4357	4464	0,025

- Regresyon analizi sonucu R^2 değeri 0.993 çıkmıştır.
- F değeri 474.06 olarak elde edilmiştir.
- MMRE değeri 0.020 olarak elde edilmiştir.
- RMSE değeri 0.025 olarak elde edilmiştir.

EK 6: F dağılışına ilişkin kritik değerler (α = 0.01)

	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	4.052	5.000	5.403	5.625	5.764	5.859	5.928	5.982	6.022	6.056	6.106	6.157	6.209	6.235	6.261	6.287	6.313	6.339	6.366
2	98.50	99.00	99.17	99.25	99.30	99.33	99.36	99.37	99.39	99.40	99.42	99.43	99.45	99.46	99.47	99.47	99.48	99.49	99.50
3	34.12	30.82	29.46	28.71	28.24	27.91	27.67	27.49	27.35	27.23	27.05	26.87	26.69	26.60	26.50	26.41	26.32	26.22	26.13
4	21.20	18.00	16.69	15.98	15.52	15.21	14.98	14.80	14.66	14.55	14.37	14.20	14.02	13.93	13.84	13.75	13.65	13.56	13.46
5	16.26	13.27	12.06	11.39	10.97	10.67	10.46	10.29	10.16	10.05	9.89	9.72	9.55	9.47	9.38	9.29	9.20	9.11	9.02
6	13.75	10.92	9.78	9.15	8.75	8.47	8.26	8.10	7.98	7.87	7.72	7.56	7.40	7.31	7.23	7.14	7.06	6.97	6.88
7	12.25	9.55	8.45	7.85	7.46	7.19	6.99	6.84	6.72	6.62	6.47	6.31	6.16	6.07	5.99	5.91	5.82	5.74	5.65
8	11.26	8.65	7.59	7.01	6.63	6.37	6.18	6.03	5.91	5.81	5.67	5.52	5.36	5.28	5.20	5.12	5.03	4.95	4.86
9	10.56	8.02	6.99	6.42	6.06	5.80	5.61	5.47	5.35	5.26	5.11	4.96	4.81	4.73	4.65	4.57	4.48	4.40	4.31
10	10.04	7.56	6.55	5.99	5.64	5.39	5.20	5.06	4.94	4.85	4.71	4.56	4.41	4.33	4.25	4.17	4.08	4.00	3.91
11	9.65	7.21	6.22	5.67	5.32	5.07	4.89	4.74	4.63	4.54	4.40	4.25	4.10	4.02	3.94	3.86	3.78	3.69	3.60
12	9.33	6.93	5.95	5.41	5.06	4.82	4.64	4.50	4.39	4.30	4.16	4.01	3.86	3.78	3.70	3.62	3.54	3.45	3.36
13	9.07	6.70	5.74	5.21	4.86	4.62	4.44	4.30	4.19	4.10	3.96	3.82	3.66	3.59	3.51	3.43	3.34	3.25	3.17
14	8.86	6.51	5.56	5.04	4.69	4.46	4.28	4.14	4.03	3.94	3.80	3.66	3.51	3.43	3.35	3.27	3.18	3.09	3.00
15	8.68	6.36	5.42	4.89	4.56	4.32	4.14	4.00	3.89	3.80	3.67	3.52	3.37	3.29	3.21	3.13	3.05	2.96	2.87
16	8.53	6.23	5.29	4.77	4.44	4.20	4.03	3.89	3.78	3.69	3.55	3.41	3.26	3.18	3.10	3.02	2.93	2.84	2.75
17	8.40	6.11	5.18	4.67	4.34	4.10	3.93	3.79	3.68	3.59	3.46	3.31	3.16	3.08	3.00	2.92	2.83	2.75	2.65
18	8.29	6.01	5.09	4.58	4.25	4.01	3.84	3.71	3.60	3.51	3.37	3.23	3.08	3.00	2.92	2.84	2.75	2.66	2.57
19	8.18	5.93	5.01	4.50	4.17	3.94	3.77	3.63	3.52	3.43	3.30	3.15	3.00	2.92	2.84	2.76	2.67	2.58	2.49
20	8.10	5.85	4.94	4.43	4.10	3.87	3.70	3.56	3.46	3.37	3.23	3.09	2.94	2.86	2.78	2.69	2.61	2.52	2.42
21	8.02	5.78	4.87	4.37	4.04	3.81	3.64	3.51	3.40	3.31	3.17	3.03	2.88	2.80	2.72	2.64	2.55	2.46	2.36
22	7.95	5.72	4.82	4.31	3.99	3.76	3.59	3.45	3.35	3.26	3.12	2.98	2.83	2.75	2.67	2.58	2.50	2.40	2.31
23	7.88	5.66	4.76	4.26	3.94	3.71	3.54	3.41	3.30	3.21	3.07	2.93	2.78	2.70	2.62	2.54	2.45	2.35	2.26
24	7.82	5.61	4.72	4.22	3.90	3.67	3.50	3.36	3.26	3.17	3.03	2.89	2.74	2.66	2.58	2.49	2.40	2.31	2.21
25	7.77	5.57	4.68	4.18	3.85	3.63	3.46	3.32	3.22	3.13	2.99	2.85	2.70	2.62	2.54	2.45	2.36	2.27	2.17
30	7.56	5.39	4.51	4.02	3.70	3.47	3.30	3.17	3.07	2.98	2.84	2.70	2.55	2.47	2.39	2.30	2.21	2.11	2.01
40	7.31	5.18	4.31	3.83	3.51	3.29	3.12	2.99	2.89	2.80	2.66	2.52	2.37	2.29	2.20	2.11	2.02	1.92	1.80
60	7.08	4.98	4.13	3.65	3.34	3.12	2.95	2.82	2.72	2.63	2.50	2.35	2.20	2.12	2.03	1.94	1.84	1.73	1.60
120	6.85	4.79	3.95	3.48	3.17	2.96	2.79	2.66	2.56	2.47	2.34	2.19	2.03	1.95	1.86	1.76	1.66	1.53	1.38
∞	6.63	4.61	3.78	3.32	3.02	2.80	2.64	2.51	2.41	2.32	2.18	2.04	1.88	1.79	1.70	1.59	1.47	1.32	1.00

EK 7: F dağılışına ilişkin kritik değerler (α = 0.05)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**15**	**20**	**24**	**30**	**40**	**60**	**120**	∞
1	161	200	216	225	230	234	237	239	241	242	244	246	248	249	250	251	252	253	254
2	18.5	19.0	19.2	19.2	19.3	19.3	19.4	19.4	19.4	19.4	19.4	19.4	19.4	19.5	19.5	19.5	19.5	19.5	19.5
3	10.1	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.64	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.53	4.50	4.46	4.43	4.40	4.37
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.84	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.41	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.12	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.90	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.74	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.61	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.51	2.47	2.43	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.42	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.41	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.35	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.29	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.24	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.19	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.15	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.08	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.03	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.01	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.98	1.94	1.89	1.84	1.79	1.73
25	4.24	3.39	2.99	2.76	2.60	2.49	2.40	2.34	2.28	2.24	2.16	2.09	2.01	1.96	1.92	1.87	1.82	1.77	1.71
30	4.17	3.32	2.92	2.69	2.53	2.42	2.33	2.27	2.21	2.16	2.09	2.01	1.93	1.89	1.84	1.79	1.74	1.68	1.62
40	4.08	3.23	2.84	2.61	2.45	2.34	2.25	2.18	2.12	2.08	2.00	1.92	1.84	1.79	1.74	1.69	1.64	1.58	1.51
60	4.00	3.15	2.76	2.53	2.37	2.25	2.17	2.10	2.04	1.99	1.92	1.84	1.75	1.70	1.65	1.59	1.53	1.47	1.39
120	3.92	3.07	2.68	2.45	2.29	2.18	2.09	2.02	1.96	1.91	1.83	1.75	1.66	1.61	1.55	1.50	1.43	1.35	1.25
∞	3.84	3.00	2.60	2.37	2.21	2.10	2.01	1.94	1.88	1.83	1.75	1.67	1.57	1.52	1.46	1.39	1.32	1.22	1.00