T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ TABANLI YAZILIM MALİYET TAHMİNİ YÖNTEMLERİNİN KARŞILAŞTIRMALI ANALİZİ

Muaz GÜLTEKİN

DOKTORA TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Danışman

Prof. Dr. OYA KALIPSIZ

Haziran, 2019

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE ÖĞRENMESİ TABANLI YAZILIM MALİYET TAHMİNİ YÖNTEMLERİNİN KARŞILAŞTIRMALI ANALİZİ

Muaz GÜLTEKİN tarafından hazırlanan tez çalışması 28/05/2019 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda DOKTORA TEZİ olarak kabul edilmiştir.

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi

Danışman

Jüri Üyeleri

Prof. Dr. Oya KALIPSIZ, Danışman
Yıldız Teknik Üniversitesi _____

Prof. Dr. Selim AKYOKUŞ, Üye
Medipol Üniversitesi _____

Doç. Dr. Özgür Koray ŞAHİNGÖZ, Üye
İstanbul Kültür Üniversitesi _____

Doç. Dr. Mehmet Sıddık AKTAŞ, Üye
Yıldız Teknik Üniversitesi _____

Dr. Öğr. Üyesi Yunus Emre SELÇUK, Üye
Yıldız Teknik Üniversitesi _____

Danışmanım Prof. Dr. Oya KALIPSIZ sorumluluğunda tarafımca hazırlanan makine öğrenmesi tabanlı yazılım maliyet tahmini yöntemlerinin karşılaştırmalı analizi başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim

Muaz GÜLTEKİN

İmza

Aileme

ve

Biricik eşime

TEŞEKKÜR

Tez çalışmam sırasında kıymetli bilgi, birikim ve tecrübeleri ile bana yol gösterici ve destek olan değerli danışman hocam sayın Prof. Dr. OYA KALIPSIZ'a, ilgisini ve önerilerini göstermekten kaçınmayan ve tez izleme dönemlerinde değerli vakitlerini ayırarak çalışmamıza ciddi katklılar sunan sayın Prof. Dr. SELİM AKYOKUŞ'a ve sayın Dr. Öğr.Üyesi Yunus Emre SELÇUK'a sonsuz teşekkür ve saygılarımı sunarım.

Tez çalışmasını bırakmayı düşündüğüm bir dönemde bana kapılarını açan Agrocares firmasına, CEO'su sayın henri HEKMAN'a ve IT Direktörü sayın Melissa ÇOLAKOĞLUN'a sonsuz teşekkür ve saygılarımı sunarım.

Tez çalışmamı nihayete erdirme sürecinde bana her türlüğü desteği veren Barsan Global, Barsan GLobal IT Direktörü sayın Serkan DİNGA ve IT Müdürü Sayın Ceyhun KARATAŞ'a sonsuz teşekkür ve saygılarımı sunarım.

İÇİNDEKİLER

SİMGE LİSTESİ

Σ	Toplama
Π	Çarpım
#	Adet

KISALTMA LİSTESİ

Cocomo	Yapıcı Maliyet Modeli (Constructive Cost Model)
DT	Geliştirme Ekibi (Development Team)
ELS	Üstel En Küçük Kareler (Exponantial Least Square)
GBR	Gradyan Artırma Regresyon (Gradient Boosting Regression)
KLOC	Kod Satısı Sayısı (Kilo Line of Code)
LLSM	Logaritmik Regresyon Yöntemi (Logarithmic Least Square Regression)
MLP	Çok Katmanlı Perseptron Regresyon (Multi-Layer Perceptron Regression)
M-POLS	Çoklu Polinomsal En Küçük Kareler (Multiple Polynomial Least Square)
OLS	Sıradan En Küçük Kareler (Ordinary Least Square)
PBI	İş Listesindeki İş Parçası (Produtct Backlog Item)
PBL	Projeye Dayalı Eğitim (Project Based Learning)
PLSM	Kuvvet Tabanlı En Küçük Kareler Metodu (Power Least Square Method)
PO	Ürün Sahibi (Product Owner)
POLS	Polinomsal En Küçük Kareler (Polynomial Least Square)
RFR	Rastgele Orman Regresyon (Random Forest Regression)
S2ELSQ	En Küçük Kareler Tabanlı Maliyet Tahmini (Software Effort Estimation by Least Square)
SM	Scrum Uzmanı (Scrum Master)
SVR	Destek Vektör Regresyon (Support Vector Regression)
VTM	Vaka Tabanlı Muhakeme
WBS	İş Dökümü Yapısı (Work Breakdown Structure)

WLS	Ağırlıklandırılmış Lineer Regresyon (Weighted Least Square)
YSA	Yapay Sinir Ağları

ŞEKİL LİSTESİ

TABLO LİSTESİ

ÖZET

Makine Öğrenmesi Tabanlı Yazılım Maliyet Tahmini Yöntemlerinin Karşılaştırmalı Analizi

Muaz GÜLTEKİN

Bilgisayar Mühendisliği Anabilim Dalı

Doktora Tezi

Danışman: Prof. Dr. Oya KALIPSIZ

Yazılım maliyet tahmini bir yazılım mühendisliği projesini geliştirmek için gerekli olan her türlü kaynağın önceden tahmin edilmesi işlemidir. Yazılım efor tahmini kavram olarak basit olsa da gerçekte zor ve karmaşıktır. Bu yüzden birçok yazılım projesi öngörülen zamanda bitirilememiş ya da proje masrafları düşünülen miktardan çok fazla olmuştur. Yazılım projelerinin farklı aşamalarında yapılan bütçe çalışmalarında, maliyet tahmin zorlukları dolayısıyla, satış tutarı ve maliyet analizleri gerçekçi hesaplanamamaktadır. Bu zorluklar, projenin kendine özgü özelliklerinden kaynaklandığı gibi, kontrol dışı bilgi eksikliğinden, bilgilerin değerlendirilmesindeki öznel yorumlardan, maliyet analiz çalışmalarında ortaya çıkan direkt ve endirekt maliyet ayrım hatalarından ve proje risklerinin tam olarak doğru tahmin edilememesinden kaynaklanabilmektedir. Yazılım projesine başlarken projenin ne kadar sürede biteceği, proje maliyeti, projede çalışacak kişi sayısı gibi birçok etmen önceden tahmin edilerek proje oluşturulmalıdır. Yazılım projelerinde kullanılacak olan kaynakların önceden tahmin edilmesinin doğruluğu ve güvenilirliği yazılım projesinin gidişatı için çok önemlidir. Yazılım geliştirme teknolojisinin sürekli değişen senaryolar içinde olması efor tahminini daha zorlu

hale getirmektedir. Yazılım projeleri için yapılan efor tahminin doğruluğu ve güvenilirliği yazılım şirketlerinin rekabeti açısından önemlidir. İyi tahminler yazılım projeleri yönetiminde çok önemli bir rol oynamaktadır. Yazılımın etkili ve verimli gelişimi doğru tahminler gerektirir. Bu çalışmada yazılım maliyetini hesaplamak için üç farklı model geliştirilmiştir. Her model için farklı veri seti kullanılmıştır. Yazılım maliyetini hesaplamak için geliştirilen birinci model regresyon tabanlı bir modeldir. Bu modelde Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri kullanılmıştır. Bu model ağırlıklandırılmış lineer regresyon, logaritmik regresyon, üstel regresyon, polinomsal regresyon ve çoklu polinomsal regresyon yöntemleri kullanılmıştır. Bu yöntemler uygulanırken en küçük kareler metodu kullanılmıştır. Geliştirdiğimiz modelin başarısını ortaya koymak için hata fonksiyonları kullanılmıştır. Ayrıca ortaya çıkan sonuçların regresyon tabanlı yazılım maliyeti yapan Cocomo (Constructive Cost Model) modeline göre sonuçları ortaya konulmuştur. Yazılım maliyet tahmininde sunduğumuz ikinci model ise yapay sinir ağları tabanlı modeldir. Bu modelde Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri kullanılmıştır. Bu modelde ise klasik çözümlerden farklı olarak veri setlerinin özellikleri gruplandırıldı ve veri setinin elde edilen formu üzerinden yapay sinir ağı (YSA) modeli geliştirildi. Böylece yazılım maliyetine farklı bir yapay sinir ağı yaklaşımı ile çözüm ortaya konuldu. Yazılım maliyeti tahmininde sunulan üçüncü model ise gerçekleştirilen yazılımın metodolojisini dikkate alarak geliştirilmiştir. Ayrıca bu modelde makine öğrenme algoritmaları kullanıldı. Bu model ile Scrum metodolojisi ile geliştirilen yazılımlar için regresyon tabanlı makine öğrenme algoritmalarını kullanarak yazılım maliyet tahmini gerçekleştirilmeye çalışılmıştır. Bu modelde Destek Karar Regresyon (Suppport Vector Regression-SVR), Gradyan Artırma (Gradient Boosted Machine-GBM), Rastgele Orman (Random Forest) ve Çok Katmanlı Perseptron (Multi-Layer Perceptron) gibi regresyon tabanlı algoritmalar kullanılmıştır. Bu Model'de Apache, JBoss, JIRA, Spring ve MongoDB projelerinde hazırlanan özelleştirilmiş Çevik veri setleri kullanılmıştır. Burada ham veri üzerinde normalleştirme ve iyileştirme işlemleri gerçekleştirilmiştir. Bu model de projeler 4 farklı faza ayrılmıştır. Her bir faz için maliyet tahmini gerçekleştirilmiştir. Son adımda ise toplam maliyet tahmini ortaya konmuştur. Bu modelde regresyon tabanlı olan gradyan artırma algoritması, destek vektör regresyon, çok katmanlı perseptron ve rastgele orman algoritmaları kullanılmıştır.

Bu tez çalışmasında yazılım maliyet tahmini için bütüncül bir yazılım maliyet tahmin modeli ortaya konulmuştur. Böylece yazılım geliştiriciler ve karar vericiler, projeye başlamadan önce daha doğru bir yazılım efor tahmini ile daha başarılı sonuçlar ortaya koyabileceklerdir.

ABSTRACT

A Comparive Analysis of Machine Learning Based Software Cost Estimation Methods

Muaz GÜLTEKİN

Department of Computer Engineering

Doctor of Philosophy Thesis

Advisor: Prof. Dr. Oya KALIPSIZ

Software effort estimation is the process of estimating the resources required to develop various kinds of software project in advance. Software effort estimation, although simple in concept, is practically difficult and complex process. So many software projects could not be completed in the estimated time or the cost of the project has been much more than the amount that is considered. Planning the budget for each stage of software development life cycle is difficult due to software development phase characteristics. So, that brings more uncertainty in computation of the total budget estimation such as sales amount and analysis effort. These difficulties may arise from the lack of out-of-control information, subjective interpretations in the evaluation of information, direct and indirect cost discrimination errors in cost analysis studies and inaccurate estimation of project risks, as a result of the specific features of the project. Before the start the project there are many factors must be defined and estimated correctly such as time, cost, scope and the number of the people. The accuracy of the estimate effort of software projects and software reliability is very important for the progress of the project. The estimation effort of the software development is constantly changing. This is sourcing from technological improvment which force the decision makers to adopt new scenarios. The accuracy of the estimate effort of software projects is crucial for

the competitiveness of software companies and reliability. In this thesis three model were developed for software effort estimation. Each model was run on different dataset. The first Effort Estimation model is regression based. Cocomo81, Cocomonasa and Cocomonasa2 dataset were used in this model. Weighted linear regression, logarithmic regression, exponential regression, power regression, polynomial regression and multiple polynomial regression algorithms were used. In this model the least square fitting solution used to represent the success of each algorithms. Finally, the result of each algorithms is compared to Cocomo model and the success of each algorithm represented according to Cocomo model. The second proposed software effort estimation model is artificial neural network based. Cocomonasa and Cocomonasa2 datasets were used in this model. This model is different from traditional solution. In this Model, as first step features are grouped according their types. After that Artificial Neural Network is developed and the success of the model is represented. The third model of software cost estimation is developed considering the methodology of the software. In addition, this model has used machine learning algorithms. In this model, software cost estimation has been tried to be realized by using regression-based machine learning algorithms for software developed with Scrum methodology. In this model Apache, JBoss, JIRA, Spring and MongoDB and Customized Agile data sets were used. In this model, normalization and improvement on raw data has been carried out. The projects in this model are divided into 4 different phases. Cost estimation was performed for each phase. In the last step, the total cost estimate has been established with suggested aggregaiton function. Gradient Boosting Algorithm, Support Vector Regression, Multi-Layer Perceptron and Random Forest Regression algorithms are used by this model. In this thesis, an integrated software cost estimate for software cost estimation models have been revealed. Thus, software developers and decision makers will be able to achieve more successful results with a more accurate software effort estimate before starting the project.

1
Giriş

1.1 Literatür Özeti

Bir yazılım projesini geliştirmeye başlamadan önce harcanacak eforu önceden tahmin etmek, projenin başarısı için önemli bir faktördür. Eğer bir projenin başarılı bir şekilde yönetilmesi ve sonuçlanması isteniyorsa bu proje için gereken eforun önceden tahmin edilmesi gerekir. Tom Demarco'nun ifade ettiği gibi ölçemediğimiz bir değeri kontrol etmemiz mümkün değildir. Bu kısımda yazılım maliyet tahmininde kullanılan Analoji Yöntemi, Uzman Kişi Yöntemi, Fonksiyon Nokta (Function Point-Use Case Point) tabanlı yöntemler, Hesaplamalı yöntemler, Regresyon Tabanlı yöntemler, Yapay Sinir Ağları tabanlı yöntemler ve Makine Öğrenmesi tabanlı yöntemler ayrıntılı bir şekilde açıklanmıştır.

Yapay Sinir Ağları modeli kullanılarak birçok farklı model bugüne kadar geliştirilmiştir. 2007 yılında Prof. Dr. Oya Kalıpsız ve Murat Ayyıldız tarafından yazılım maliyetini tahmin etmek için yapay sinir ağları modeli geliştirilmiştir [1]. Ayrıca konu ile ilgili olarak Reddy ve Raju [2] "A Concise Neural Network Model for Estimating SoftwareEffort" adlı çalışmayı yayımlamışlardır. Bunun gibi birçok çalışma yapılmıştır. Birçok çalışmada, proje için oluşturulan özelleştirilmiş veri setleri kullanılarak yazılım maliyet tahmini yapılmaya çalışılmıştır. Bu veri setlerinin çoğu kamuya açık (public) değildir.

Geliştirilen YSA modelleri gerçek değer ile hesapladığı değer arasındaki farkı en az indirmeye çalışır. İstenen değer elde edildikten sonra YSA'yı oluşturan ağırlıkların katsayıları belirlenir. Böylece YSA modeli gerçekleştirilmiş olur. Önerdiğimiz YSA modelinin başarısını belirlemek için çözüm geliştirildi. YSA modelinin başarısını belirlemek için geliştirilen bu çözüm, YSA'nın ürettiği çıkış değeri ile gerçek değer arasındaki hata miktarının hesaplanıp ve elde edilen değerin gerçek değere oranının bulunması ile gösterildi. Ayrıca Boehm tarafından geliştirilen Cocomo modeline göre başarı sonuçları ortaya konuldu [3].

Yazılım maliyet tahmininde kullanılan bir başka model çeşidi ise regresyon tabanlı modellerdir. Yazılım maliyet tahmininde kullanılan en popüler regresyon tabanlı model Boehm tarafından geliştirilen Cocomo modelidir. Cocomo modeli proje türlerine göre (Embedded, Semi-Detached ve Organic) yazılım maliyet tahmininde kullanılan üstel tabanlı regresyon formülleri sunmaktadır. Bu nedenle yazılım maliyet tahmininde kullanılan regresyon tabanlı birçok model geliştirilmiştir [4]. Regresyon tabanlı geliştirilen modeller de daha çok bir ya da iki farklı regresyon çözümü üzerinden gidilerek modeller geliştirilmiştir.

Yazılım maliyet tahmininde kullanılan başlıca regresyon türleri lineer regresyon, üstel regresyon ve polinomsal regresyon yöntemleridir. Ağırlıklandırılmış lineer regresyon ve lineer olmayan regresyon çözümleri kullanılarak yazılım maliyet tahmini yapılmaya çalışılmıştır. Konu ile ilgili olarak, Dr. Fedtova ve Dr. Alvelos regresyon yöntemi kullanarak "Software Effort Estimation with Multiple Linear Regression: Review and Practical Application" adlı çalışmayı gerçekleştirmişlerdir [4].

Yazılım maliyetinde en çok tercih edilmesi gereken yöntem regresyon tabanlı yöntemler olmalıdır [4]. Bu nedenle bu alanda ortaya konacak modellerin başarısı önemlidir. Çünkü regresyon tabanlı modellerde giriş değerlerine bağlı olarak çalışan ve sonuç üreten bir sistem olmalıdır. Cocomo modeli regresyon tabanlı bir modeldir. Bu model giriş değeri olarak kod satır sayısını (KLOC-Kilo Line Of Code) almaktadır. Cocomo sunduğu üstel fonksiyonlar ile kod satırı sayısını giriş değeri olarak alarak efor tahmini yapabilmektedir. Bu nedenle geliştirilen birçok regresyon tabanlı model, Cocomo'dan daha iyi sonuç veren formüller üzerinde yoğunlaşmıştır.

Bir yazılımın başarıya ulaşmasında en önemli faktörlerden bir tanesi yazılımın belli bir metodoloji dahilinde geliştirilmesidir. Yazılım geliştirilirken bir yazılım geliştirme modeline bağlı olarak geliştirilmesi yazılımın başarıya ulaşmasında önemli bir faktördür. Bu nedenle birçok yazılım geliştirme metodolojisi geliştirilirmiştir. Bu modellerin başlıca kullanılanları Şelale, V-Modeli, Yinelemeli, Artırımsal, Spiral ve Çevik modelidir. Her yazılımın bir yaşam döngüsü olmalıdır. Bu döngü analiz, tasarım, geliştirme, test ve bakım süreçlerinden oluşmaktadır. Bu

süreçler sayesinde istenilen yazılım, istenilen kalitede ve istenilen performans kriterlerinde kullanıcıya teslim edilir.

Günümüzde en çok kullanılan yazılım geliştirme modellerinden bir tanesi Scrum modelidir [5]. Ken Schwaber ve Jeff Sutherland, 1995 yılında OOPSLA konferansında ilk kez Scrum'ı sunmuştur. 2001 yılında, Sutherland, Schwaber on beş uzman ile beraber Utah'da bir araya gelerek, Çevik Manifestosu 'nu hazırlamıştır [6]. Scrum Çevik metodolojisinin bir türüdür. Scrum modeli, çalışan ürünü hızlı bir şekilde kullanıcıya teslim etme prensibine göre çalışır. Scrum modeli dinamik bir yapıya sahiptir. Scrum Modeli değişikliklere açık bir modeldir. Scrum, öngörülebilirliği optimize etmek ve riski kontrol etmek için yinelemeli, artımlı bir yaklaşım kullanır. Scrum metodolojisinde projede gerçekleştirilecek olan işler ürün iş listesi (product-backlog) denilen listede toplanır. Bu listedeki her bir işin süresinin doğru bir şekilde tahmin edilmesi projenin başarısı için çok önemlidir.

Scrum modeli ile gerçekleştirilen yazılım projelerinin maliyetinin tahmin edilmesi diğer metodolojiler ile gerçekleştirilen yazılım projelerinin maliyet tahmininden farklıdır. Scrum dinamik bir yapı sunarken Şelale modeli daha statik bir model sunmaktadır. Bu nedenle Şelale modelinde yazılım maliyet tahmini daha öngörülebilirinken bu durum Scrum modelinde daha karmaşık ve zordur.

Scrum yazılım geliştirme metodolojisinde yazılım maliyet tahmini için farklı yöntemler sunulmuştur. Bu çalışmaların başında B. Michael tarafından yapılan "Delivering large-scale IT projects on time, on budget, and on value," [7] çalışması gelmektedir. Bu çalışmada regresyon tabanlı, sınıflandırma tabanlı, YSA tabanlı yöntemler kullanılmıştır.

Scrum yazılım maliyet tahmininde kullanılan yöntemlerden bir tanesi ise regresyon tabanlı makine öğrenmesi algoritmalardır. Scrum ile geliştirilen yazılım proje maliyet tahminine ait herkese açık veri seti bulunmamaktadır. Bu nedenle bu bağlamda yapılan çalışmalar proje için oluşturulan özel veri setlerinden oluşmaktadır.

Scrum ile geliştirilen yazılım projelerinde maliyet tahmini için K-Means, Bayes ve karar ağaçları yöntemleri kullanılmıştır. Bu bağlamda karar ağaçları ile başarılı sonuçlar elde edilmiştir [8].

1.2 Tezin Amacı

Bu tez çalışmasında amaç bir yazılımı geliştirmeye başlamadan önce yazılımı geliştirmek için gerekli maliyeti tahmin etmektir. Standish grubunun 2012'de yaptıkları bir çalışmaya göre yazılım projelerinin %74'ü kendileri için belirlenen süreleri aşmıştır ve yine Standish grubuna göre yazılım projelerinin %59'u kendileri için belirlenen bütçeleri aşmıştır [9]. Bu nedenle bir yazılıma başlamadan önce doğru bir tahmin yaparak ise başlamak projenin başarıya ulaşmasında çok önemli bir faktördür.

Bu nedenle bu tezde yazılım maliyetini doğru bir şekilde hesaplamak için 3 farklı yazılım maliyet tahmin modeli ortaya konulmuştur. Üç farklı model ortaya konulmasının nedeni geliştirilen projenin metodolojisini de göz önünde bulundurularak daha iyi sonuçlar elde edilmek istenmesidir. Çünkü yazılım geliştirirken kullanılan metodoloji de yazılımın başarısında önemli rol oynar ve yazılım maliyetinin belirlenmesinde önemli bir faktördür. Bu bağlamda öncelikli olarak regresyon tabanlı bir model geliştirilerek yazılım maliyet tahmini yapılmıştır. Fakat burada geleneksel yöntemlerden farklı olarak 6 farklı regresyon yöntemi kullanılarak çok daha başarılı yazılım maliyet tahminlerinin yapılması amaçlanmıştır. Geliştirilen ikinci modelde, yapay sinir ağları kullanılarak yazılım maliyet kestirimi yapılmaya çalışılmıştır. Bu model ile yazılım maliyetinde en az hata ile yazılım maliyet tahmini yapılamaya çalışılmıştır. Bu model de veri setlerinin özelliklerinin gruplandırılmasıyla yazılım maliyeti tahmininde daha iyi sonuçlar elde edilmesi amaçlanmıştır. Yazılım maliyet tespitinde geliştirilen üçüncü model ile Scrum metodolojisi ile geliştirilecek olan yazılımlar için yazılım maliyet tahminin yapılması amaçlanmıştır. Bu model regresyon tabanlı makine öğrenmesi algoritmalarını kullanarak yazılım maliyet tahmini yapmayı amaçlamıştır. Scrum Yazılım geliştirme metodolojisinde yapılacak işler iş listesine atılır. Burada işler öncelik sırasına göre koşumlara (Sprint) atılır. Her bir sprintte yapılacak işin eforu tahmin edilir. Bu model, yapılacak her bir iş parçasının ilgili sprinte atılmadan önce

eforunu makine öğrenme algoritmaları vasıtası ile doğru bir şekilde tahmin etmeyi amaçlamıştır. Bu tezde üç farklı yazılım maliyeti tahmini modeli sunularak, yazılım metodolojisine bağlı olarak daha doğru ve anlamlı yazılım maliyeti tahminin yapılması amaçlanmıştır. Yazılım maliyet tespitinde bütüncül bir çözüm ortaya konulması amaçlanmıştır. Bu açıklamalar ışığında aşağıda ifade edilen araştırma sorularına cevap bulunmayan çalışılmıştır.

Soru 1: Regresyon tabanlı yazılım maliyet tahmininde farklı regresyon algoritmaları kullanıldı. Neden regresyon tabanlı algoritmalar tercih edildi?

Soru 2: Regresyon tabanlı yazılım maliyet tahmini modelinde kullanılan algoritmalardan en başarılı sonucu ortaya koyan algoritma hangisidir? Bu algoritmanın başarılı olmasında etkili olan faktörler hangileridir?

Soru 3: Geliştirilen Regresyon tabanlı yazılım maliyet tahmini modeli bugüne kadar geliştirilen regresyon tabanlı yazılım maliyet tahmini modellerine göre üstünlüğü ve getirdiği yenilik nedir?

Soru 4: Geliştirilen Regresyon tabanlı yazılım maliyet tahmini modeli farklı veri setleri üzerinde test edildiğinde aynı başarıyı ortaya koyabilecek midir?

Soru 5: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı modelin bugüne kadar geliştirilen yapay sinir ağları tabanlı yazılım tahmini modellerine göre farkı nedir?

Soru 6: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı modelde kullanılan veri setinin özellikleri gruplandırılmıştır. Neden bu gruplandırma işlemi yapıldı?

Soru 7: Bu gruplandırma işleminin sonuncuda elde edilen sonuçların gruplandırma yapılmadan geliştirilen yapay sinir ağları modeline göre daha ortaya koyduğu düşünüldüğünde burada rol alan gruplandırma şemantiği genelleştirilebilir mi?

Soru 8: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde neden regresyon tabanlı makine öğrenmesi algoritmaları kullanıldı? Sınıflandırma tabanlı algoritmalar kullanılmaz mıydı?

Soru 9: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde fazlara ayrılarak çalışan bir mimari geliştirilmiştir. Fazlara ayrılma işlemi geliştirilen modelin başarısını nasıl etkilemiştir?

Soru 10: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde Fazlara ayrılma işlemi hangi kriterlere göre gerçekleştirilmiştir?

Soru 11: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde aritmetik tabanlı Story Point notasyonu kullanılmıştır. Aritmetik gösterim yerine farklı bir gösterim kullanılması durumunda maliyet hesaplanmasında nasıl bir başarı ortaya çıkabilir?

1.3 Hipotez

Yazılım Geliştirme süreci kendi içinde yaşayan bir süreçtir. Bu süreç temelde analiz, tasarım, geliştirme, test ve bakim adımlarından oluşmaktadır. Yazılım geliştirme metodolojileri istenen yazılımın, istenen kalitede ve istenen performansta olmasını sağlamaya çalışırlar. Bir yazılımın başarılı olmasında yazılım maliyeti önemli rol oynamaktadır. Bir yazılımın başarılı olması için maliyetlendirilmesinin doğru bir şekilde yapılması gerekmektedir. Yazılım maliyeti tahmini regresyon tabanlı olmalıdır. Çünkü hesaplanacak olan maliyet miktarı yazılım geliştirmede kullanılacak olan kaynakların (insan kaynağı, yazılımın özelliği, yazılımın büyüklüğü, yazılımın işlevselliği, yazılımın karmaşıklığı vb.) miktarına göre değişir. Bu nedenle yazılım maliyeti için geliştirilecek olan modeller regresyon tabanlı olmalıdır. Ayrıca yazılım maliyet tahmini için oluşturulan veri setleri regresyon analizlerine uygun olmalıdır. Bu tezde, kullanılacak doğrusal olmayan regresyon algoritmalarının daha başarılı sonuçlar vereceği düşünülmektedir. Bunun yanı sıra regresyon tabanlı yapay sinir ağları da yazılım maliyetinde daha doğru sonuçlar ortaya koyacağı düşünülmektedir. Son yıllarda yazılım maliyeti kestirimi için yapılan çalışmalarda, yazılım geliştirme modeline bağlı olarak yazılım maliyet kestirimi yapılmaya başlandığı gözlemlenmektedir [10]. Scrum metodolojisinde yapılacak olan yazılım maliyet tahmini, bu modele özgü olmalıdır. Çünkü bu model kendi içinde bir yaşam döngüsü olan ve klasik yazılım geliştirme süreçlerinden farklılık arz etmektedir. Bu nedenle Scrum modeli için yazılım maliyeti tahmini kendine özgü olmalıdır. Yazılım maliyet tahmini için regresyon tabanlı makine

öğrenme algoritmalarının daha başarılı sonuçlar üreteceği düşünülmektedir. Çünkü burada yazılım maliyeti tahminde kullanılan veri seti regresyon tabanlı makine öğrenme algoritmalarının uygulanmasına daha uygundur. Önerdiğimiz hipotezin doğruluğu Bölüm 5'te modellerin ortaya koyduğu sonuçlar vasıtası ile gösterilmiştir.

2
Yazılım Maliyet Tahmini

2.1 Yazılım Maliyet Tahmini Nedir

Proje yönetiminde maliyet tahmini, proje faaliyetlerini tamamlamak için ihtiyaç duyulan parasal kaynakların hesaplanması sürecidir [11]. Genellikle yazılımlar projeler ile geliştirilir. Bu nedenle yazılım maliyet tahmini, yazılımı tamamlamak için gerekli olan parasal kaynakların bir yaklaşımı olarak düşünülebilir. Genel bir maliyet tahmini için kullanılan girdiler, araçlar, teknikler ve çıktılar aşağıdaki Şekil 2.1'de gösterilmiştir [11]. Projede maliyet tahmininde ana çıktılar; yazılım maliyet tahmin aktiviteleri, maliyet tahminini oluşturan elemanlar ve projede yapılan güncellemelerden oluşmaktadır. Proje dokümanları da projeye ait çıktılardır. Proje çıktı dokümanları projede yapılan güncellemelerden oluşmaktadır.

Şekil 2.1 Proje Yönetimi Maliyet Tahmini

Yazılım eforu tahmini araştırma literatüründe, genellikle çaba ile maliyet kavramları arasında bir fark yoktur. Bu, esas olarak yazılım geliştirmede, hemen hemen tüm maliyetlerin, doğrudan çabaya bağlı olan personel maliyetleri olduğu gerçeğinden kaynaklanır. Bununla birlikte, global yazılım geliştirmede, maliyet oranları farklı alanlarda farklılık gösterebilir. Bu da bir bölgedeki çabanın, başka bir yerdeki çabadan daha yüksek maliyetlere neden olabileceği anlamına gelir [12]. Bununla

birlikte, bu araştırmada çaba ve maliyetler, aksi belirtilmedikçe eşanlamlı olarak kullanılacaktır.

2.1.1 Aktivite Maliyet Tahmini

Aktivite maliyetleri tahminleri, yazılımla ilgili çalışmaları tamamlamak için gerekli olan maliyetlerin niceliksel (sayısal) değerlendirmeleridir [11]. Yazılımın geliştirilmesi için ihtiyaç duyulacak tüm kaynaklar için maliyetler tahmin edilmektedir. Bu işgücü, malzeme, donanım, hizmet ve tesislerin yanı sıra bir enflasyon ödeneği veya risk/beklenmedik maliyetler gibi özel kategorileri de içerir, ancak bunlarla sınırlı değildir. Maliyet tahminleri, bazı para birimleri cinsinden ifade edilir (ör. Dolar, Euro, Yen vb.); Bazı durumlarda, gün veya ay gibi diğer ölçü birimleri, para birimi dalgalanmalarının etkilerini ortadan kaldırarak karşılaştırmaları kolaylaştırmak için kullanılır. NASA'nın yazılım maliyet tahmini ile ilgili kılavuzunda, yazılım maliyetlerinin temel denklemi yazılımı geliştirmek için gerekli işçilik maliyetleri, yazılım geliştirmek için gerekli diğer işçilik maliyetleri ve işgücü dışındaki maliyetlerinin toplamı ile ifade edilmektedir. NASA tarafından sunulan yazılım geliştirme maliyetine ait maliyet elemanları Tablo 2.1'de gösterilmiştir [13].

Tablo 2.1 Yazılım Geliştirme Maliyet Elemanları

Efor Tipi	Elemanlar	Açıklamalar
Yazılım geliştirme Eforu	Yazılım sistem mühendisliği	Fonksiyonel tasarım, yazılım gereksinimleri ve ara yüz özellikleri gibi aktivitelerin yazılım mimarları, yazılım sistem mühendisleri ve alt sistem mühendisleri tarafından gerçekleştirilmesidir.
	Yazılım mühendisliği	Ünite tasarımı, kod, birim testi ve yazılım bileşenlerinin bilişim mühendisleri ve yazılım geliştiricileri tarafından entegre edilmesidir.
	Yazılım test mühendisliği	Test planlarının oluşturulması ve gerçekleştirilmesi işlemlerini kapsar. Birim test haricindeki bütün test işlemlerinin gerçekleştirilmesidir.
Diğer işgücü	Yazılım proje yönetimi ve destek	Yazılım proje yöneticisi, yazılım proje alt yöneticisi, teknik lider ve sistem yönetimi tarafından gerçekleştirilen yazılım projesi yönetimi ve yazılım projesinin konfigürasyon yönetimidir.
	Yazılım sistemi düzeyinde test desteği	Yazılımın geliştirilmesinden ve simülasyonundan oluşur.
	Montaj, test ve lansman operasyonları	
	Yönetim destek maliyetleri ve Kalite Süreci	
	Bağımsız doğrulama ve onaylama testleri	
İş gücü dışındaki maliyetler	Destek ve hizmetler (servisler)	İş istasyonları, test ortamları, simülatörler, yer destek ekipmanları, ağ ve telefon ücretleri ve benzerlerinden oluşur.
	Yazılım alımı	Geliştirme ortamları, derleyiciler, lisanslar, test araçları ve geliştirme araçlarından oluşur.
	İş gezileri	Proje ile ilgili iş görüşmeleri ve konferanslardan oluşur

2.1.2 Tahmin Esasları

Yazılım Tahmin Esasları, maliyet tahmininin nasıl elde edildiğine dair net ve eksiksiz bir anlayış sağlamalıdır [12]. Maliyet tahminini destekleyen ek ayrıntıların miktarı

ve türü, kullanıldığı alana göre değişir. Etkinlik maliyeti tahminlerinin değerini belirlemek için aşağıdaki maddeler kullanılır.

• Yazılım Maliyet Tahmini dokümantasyonu (Nasıl geliştirildi?)

• Yazılım Maliyet Tahminin kapsamı ile ilgili dokümantasyon (Tahmin edilen nedir?)

• Yapılan tüm varsayımların dokümantasyonu

• Bilinen sistem üzerindeki kısıtların belgelenmesi

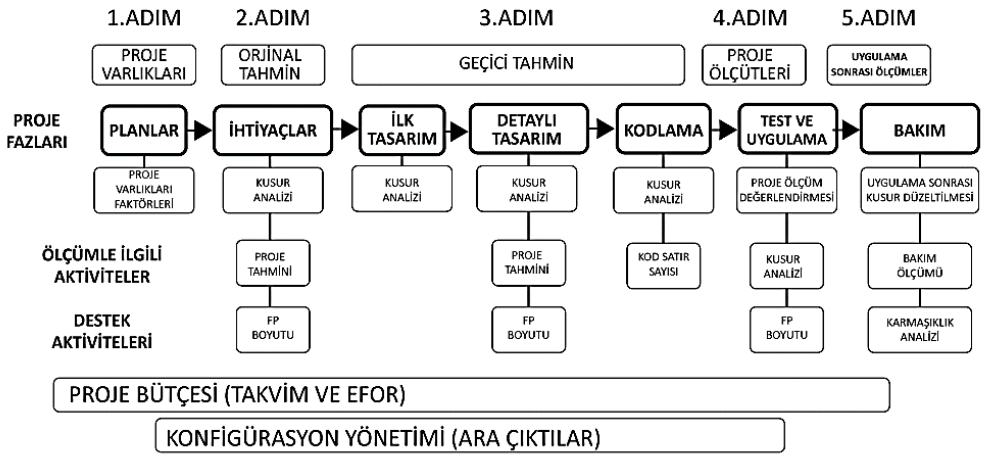
• Muhtemel tahminlerin aralığının belirtilmesi (Bir değer aralığı arasında ürünün maliyetinin bekleneceği anlamına gelir (±%10))

• Tahminlerin güvenirliliğini ifade etmek için güven düzeyinin oluşturulması ve kullanılması.

Yukarıda ifade edilen adımlar doğrultusunda maliyet hesaplamasının temel adımları belirlenmiş olur. Ama bu adımların proje ilerledikçe çok daha detaylandırılması gerekmektedir. Burada sadece kabaca ve basitçe ifade edilmiştir.

2.1.3 Yazılım Maliyet Tahmini Ne Zaman Yapılır?

Yazılım maliyeti tahmini bir yazılım projesinde bağımsız ve bireysel bir etkinlik değildir. Bunun ile birlikte, farklı aşamalar boyunca devam eden ve yazılım geliştirmedeki diğer faaliyetlerle büyük ölçüde bağlantılı olan yinelemeli bir süreçtir. Tahminler, büyük ölçüde yazılımın gerekliliklerinden türetilmiş olup, bunlar, projeyle ilişkili araçlar, süreç ve diğer niteliklerden büyük ölçüde etkilenmektedir. Yazılım geliştirme faaliyetlerinin bir parçası olarak maliyet tahmini aşağıdaki Şekilde gösterilmiştir [14]. Şekil 2.2'de yazılım geliştirme aktivitelerinin maliyetini belirleyen adımlar arasındaki ilişki detaylı bir şekilde sunulmuştur.



Şekil 2.2 Yazılım Geliştirme Aktivitesi Olarak Maliyet Tahmini [14]

Yazılım maliyeti ile ilgili tahminler herhangi bir proje için, aşağıdakiler dahil, ancak bunlarla sınırlı olmamak üzere, geliştirme süreci boyunca çoklu maliyet tahminlerinden oluşmaktadır:

• Zorlu ön gereksinim tahminleri

• Proje gereksinimlerinden elde edilen ilk resmî tahmin

• Gereksinim değişikliklerini yansıtan yeni tahminlerin ortaya konması

• Proje geçmiş verilerini kullanarak nihai bir maliyet tahminin oluşturulması

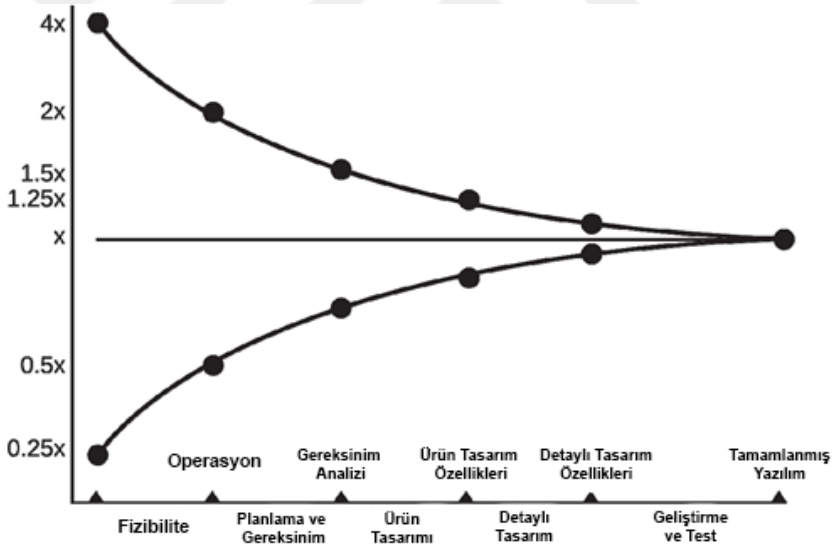
Yazılım maliyet tahmini, yazılımın yaşam döngüsünün erken aşamalarında başlamaktadır. Gereksinim analiz dokümanı hazırlanma safhasından başlar bakım safhasına kadar devam eder. Yazılım geliştirme sürecinin her safhası için maliyet hesaplama işlemi gerçekleştirilmektedir.

2.1.4 Neden Yazılım Maliyetleri Yanlış Hesaplanır?

Yazılım maliyetlerinin çok yanlış tahmin edildiği ve yazılım geliştirme projelerinde büyük sorunlara neden olduğu, sektör çapında bir sorun gibi görünmektedir. Bu problemler birçok araştırma literatüründe ve yazılım maliyet tahmini ile ilgili çalışmalarda yer almaktadır [3], [14-16]. Bir projedeki gerçek eforun değerine, tahmin edilen eforun %30'unun ilave edilmesiyle elde edilebileceği gibi mizahi "endüstri şakaları" dahi vardır. Gerçekte, bazı çalışma miktarlarının fiili eforun çok altında tahmin edildiği nadirdir [17].

Yazılım geliştirme aşamalı bir arttırma (incremental) sürecidir. Bir yazılım projesi, genel bir yazılım bakış açısı ile başlar. Yazılım kavramı, ürün gereksinimi ve proje hedeflerine göre rafine edilir. Yazılım geliştirme yaşam döngüsü boyunca iş ilerledikçe proje için yapılan tahminlerin doğruluğu artacaktır. Maliyet artışının bir nedeni de yanlış yapılan tahminlerdir. Bu durum ile ilgili olarak, Barry Boehm, yazılım yaşam döngüsünün ilk aşamalarında yapılan maliyet tahminlerinin, dört faktör tarafından hatalı sonuçlanabileceğini belirtmiştir. Bu yanlışlığın temel nedeni olarak da yazılım gereksinimlerinin açık bir şekilde anlaşılmaması şeklinde ifade etmiştir [3].

Aşağıdaki Şekil 2.3, gereksinim spesifikliği (ayrıntılandırma) arttıkça maliyet tahminlerinin doğruluğunun arttığını göstermektedir. Tahminler herhangi bir noktada gereksinimlerde ifade edilen değerden daha düşük olamaz. Bu grafik "Belirsizlik Konisi" olarak bilinir [3].



Şekil 2.3 Belirsizlik Konisi [3]

Yazılım maliyet tahminlerinin doğruluğunu geliştirmek için herhangi bir basit çözüm yoktur. Güvenilir tahminlerin oluşturulmasının anahtarı, çeşitli yöntemler ve araçlar kullanmak ve daha sonra, bir kişinin sağladığı tahminlerin neden bir diğeri tarafından sağlananlardan önemli ölçüde farklı olabileceğini anlamak için sonuçları farklı tekniklerle analiz etmektir [3]. Tahminci bu farklılıkları makul ve tatmin edici

bir oranda açıklayabilirse, o zaman projenin maliyetini yönlendiren faktörleri iyi bir şekilde anlaması muhtemeldir ve böylece yönetim tarafından gerçekleştirilen gerekli proje planlama ve kontrol işlevlerini desteklemek için daha donanımlı olacaktır.

2.1.5 Yazılım Maliyet Hesaplama Süreci

Yazılım için güvenilir bir maliyet tahmini oluşturmak için önemli miktarda veriye sahip yapılandırılmış bir yaklaşım gerekmektedir. Yazılım maliyeti tahmini dikkatle planlanması, yönetilmesi ve takip edilmesi gereken küçük boyutlu bir proje olarak görülebilir.

Birçok kuruluşun yazılım maliyeti tahmini için farklı süreçleri vardır. Bu süreçler birçok yönden farklılık gösterir ve tüm organizasyonlarda ve araştırmalarda kullanılan tek bir ortak süreç gibi görünmemektedir. Örnek olarak yazılım maliyet tahmini için iki süreç aşağıda sunulmuştur. Barry Boehm tarafından sunulan çözüm Tablo 2.2'de [3], NASA tarafından sunulan çözüm Tablo 2.3'de [13] gösterilmiştir.

Tablo 2.2 Barry Boehm Yazılım Maliyet Hesaplama Süreci

Adımlar	Aksiyon	Açıklama
Adım 1	Maliyet tahmini hedefleri belirlenir.	Sürecin sonraki aşamalarını gerçekleştirmek için gerekli detay seviyesi belirlenmeli ve hedefler ortaya konmalıdır. Bu hedefler, karar alma, farklı tahminlerin doğruluğu ve tahmin için harcanan çaba/maliyet/zaman için gereken bilgileri içermelidir.
Adım 2	Gerekli veriler ve kaynaklar için proje planı oluşturulur.	Gerekli veri ve kaynaklar için bir proje planı oluşturulmalıdır. Bu plan, tahmin faaliyetine niçin, ne zaman, kim, nerede, nasıl ve ne kadar maliyet harcayacağı sorularına cevap vermelidir. Plan, yazılımın amaçlarına ve büyüklüğüne bağlı olarak basit bir metin olabileceği gibi yazılımın büyüklüğüne bağlı olarak farklı formlarda da olabilir.
Adım 3	Yazılım gereksinimleri sabitlenir.	Yazılım gereksinimleri ve spesifikasyonları, tahmin edilen hedefler göz önünde bulundurulduğunda olabildiğince açık olacak şekilde ifade edilmelidir. Yazılım geliştirme maliyetini tahmin ederken yapılan varsayımları belgelemek değerlidir.
Adım 4	Mümkün olduğu kadar fazla detay verilmelidir.	Maliyet tahmini hedefleri göz önünde bulundurulduğunda, yazılım hakkında mümkün olduğunca fazla bilgiye ve ayrıntıya ihtiyaç vardır. Daha fazla bilgi ve detaylar mevcut olduğunda, tahminler de daha doğru olacaktır.
Adım 5	Çeşitli bağımsız maliyet tahmin teknikler ve kaynaklar kullanılmalıdır.	Gerçek tahminlerde, alternatiflerin hiçbiri diğerlerinden daha iyi olmadığı için birkaç farklı teknik kullanılması tavsiye edilir. Tekniklerin çoğu tamamlayıcı güçlü ve zayıf yönlere sahiptir.
Adım 6	Farklı tahminler karşılaştırılmalı ve yinelenmelidir.	Tahmin sonuçlarının farklı teknik ve kaynaklardan karşılaştırılması ve yinelenmesi gereklidir. Bağımsız maliyet-tahmin tekniğinin kullanılmasının en değerli yönü farklı maliyet tahminlerinin birbirine karşı olan zayıflıklarını ve güçlü yönlerini ortaya koyabilmesidir.
Adım 7	İzleme	Tahminler tamamlandıktan ve yazılım projesinin kendisi başladıktan sonra, gerçek maliyetler ve ilerlemeler hakkında veri toplamak ve bunları tahminlerle karşılaştırmak anlamlı hale gelecektir. Bu da ortaya konan tahminin başarısını ortaya koyacaktır. Bu nedenle süreç sürekli takip edilmelidir.

Tablo 2.3 NASA Yazılım Maliyet Hesaplama Süreci

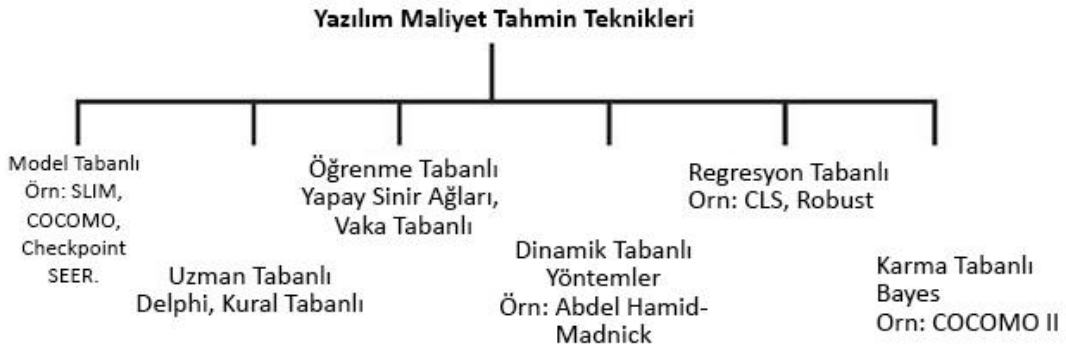
Adımlar	Aksiyon	Açıklama
Adım 1	Yazılımı fonksiyonel ve programatik olarak analiz edip bilgi toplama	Yazılım gereksinimleri, yazılım mimarisi ve programatik kısıtlamalar analiz edilmeli ve hassaslaştırılmalıdır.
Adım 2	İş elemanlarını ve tedarikçilerini tanımlama	Belirlenen bir proje için iş elemanları ve ihtiyaçları (tedarik) tanımlanmalıdır.
Adım 3	Yazılım boyutunu tahmin etme	Yazılım boyutu anlamlı kod satırı cinsiden tahmin edilmelidir.
Adım 4	Yazılım eforunu tahmin etme	Yazılım efor tahmini SLOC olarak ifade edilmelidir. Her bir iş parçası için harcanacak efor belirlenmelidir.
Adım 5	Harcanacak eforu planlama	Yazılım çalışmasını tamamlamak için gereken süre belirlenmelidir. İş parçaları içim zaman periyotları oluşturulmalıdır.
Adım 6	Maliyeti hesaplama	Yazılım projesinin toplam maliyeti tahmin edilmelidir.
Adım 7	Risklerin etkisini belirleme	Yazılım projesi riskleri tanımlanmalıdır, etkileri tahmin edilmelidir ve tahminler gözden geçirilmelidir.
Adım 8	Tahminleri modellerle doğrulama	Orijinal tahminleri doğrulamak ve doğruluğu geliştirmek için alternatif çaba, program ve maliyet tahminleri geliştirilmelidir.
Adım 9	Tahminleri, bütçeyi ve zamanlamayı ayarlama	Boyut, efor, zamanlama ve maliyet tahminleri gözden geçirilmelidir ve proje bütçesi ile karşılaştırılmalıdır. Tutarsızlıklar çözülmelidir.
Adım 10	Tahminleri inceleme ve onaylama	Yazılımın eforu, zamanlaması ve maliyet tahminleri gözden geçirilmelidir ve onaylanmalıdır.
Adım 10	İzleme, raporlama ve bakım yapma	Tahminler gerçek verilerle karşılaştırılmalıdır. Tahminin doğruluğu izlenmelidir. Her önemli ilerleme adımında büyüklük, efor, zamanlama ve maliyet tahminleri raporlanmalıdır.

2.2 Yazılım Maliyet Hesaplama Teknikleri

Bu bölümde yazılım maliyetini hesaplama teknikleri anlatılacaktır. Yazılım maliyet tahmininde kullanılan teknikler türlerine göre ayrılmış olup Şekil 2.4'te gösterilmiştir [18].

2.2.1 Genel Bakış

Yazılım maliyet tahmini teknikleri farklı şekillerde kategorize edilebilir. Yazılım Maliyet Tahmini teknikleri, aşağıdaki şekilde gösterildiği gibi altı ana kategoriye ayırılmaktır [18]. Bu bölümde bu kategoriler açıklanacak ve her kategoriden örnekler sunulacaktır.



Şekil 2.4 Yazılım Tahmin Teknikleri

Bir kategori içindeki her kategori ve/veya teknik, göreceli olarak güçlü ve zayıf yanlarına göre başka kategori ile birlikte kullanılabilir. Örneğin, uzman tabanlı yöntemler ile model tabanlı teknikler beraber kullanılabilir. Model tabanlı yöntemler geçmiş deneyimlerden elde edilen yüksek kaliteli verilerle kalibre edilebildiklerinde iyi sonuçlar verirler, bu nedenle bu iki yöntem beraber kullanılabilir.

Farklı yazılım maliyeti tahmin teknikleri ve bunların doğruluğu konusunda çok çeşitli araştırmalar vardır. Araştırmacıların çoğu, tüm yönleriyle diğerlerine kıyasla üstün bir teknik olmadığını ve farklı tekniklerin güçlü ve zayıf yanlarının genellikle tamamlayıcı olduğunu düşünmektedir [3], [13], [16]. Bu nedenle, en gerçekçi tahminleri üretmek için tekniklerin bir kombinasyonunu kullanmak ve her birinden elde edilen yazılım maliyet tahminlerini dikkatlice karşılaştırmak ve yinelemek önemlidir. Tekniklerin belirli kombinasyonu, maliyet tahmini amaçlarına göre seçilebilir.

2.2.2 Model Tabanlı Teknikler

Modele dayalı teknikler, yazılım geliştirme projelerinin efor ve maliyetini matematiksel denklemler ve fonksiyonlar yardımıyla bulmaya çalışır. Oluşturulan modellerin teorik yapısı, genellikle matematiksel modellerin fonksiyonel biçimi olarak ifade edilir. Fakat model tabanlı teknikler birçok durumda, geçmiş projelerden elde edilmiş verilerle kalibre edilmelidir.

Modele dayalı teknikler objektiftir ve kazanma arzusu veya müşteri memnuniyeti gibi insan faktörlerinden etkilenmezler. Ayrıca tekrarlanabilirler ve çoğu zaman tahminlerde hassas analizleri desteklerler. Buna ek olarak, benzer geçmiş deneyimlerden elde edilen yüksek kaliteli verilerle kalibre edildiğinde doğru sonuçlar verebilirler.

Öte yandan, modele dayalı teknikler, örneğin personel ve ekip çalışması gibi istisnai durumları dikkate alamamaktadır. Ayrıca hatalı girdi verilerini telafi edemezler. Modeller geçmiş projelerden elde edilen verilerle kalibre edilmektedir. Yeni teknolojik özellikleri, programlama dillerindeki gelişimi gibi yeni özelliklerin bu modellerin geliştirildiği döneme göre olan farklı yönlerini hesaba katmazlar. Bu da modellerin yeni teknolojiler ile beraber kalibrasyonunu gerektirir. Sonraki alt bölümde üç farklı model tabanlı teknik açıklanmıştır: Putnam Yazılım Yaşam Döngüsü Modeli (SLIM), Fonksiyon Nokta Analizi (FPA) ve Yapısal Maliyet Modeli (Cocomo). Ayrıca Fiyat-S [19], Estimacs [20], SEER-SEM [21] ve Checkpoint [22] gibi birçok popüler model tabanlı teknik de açıklanmıştır.

2.2.2.1 Putnam Modeli

Putnam modeli 1970'lerin sonunda Larry Putnam tarafından geliştirilmiştir [23]. Rayleign eğrisi kullanarak bir yazılım projesini bitirmek için gereken zamanı ve eforu tanımlar. Yazılım Yaşam Döngüsü Modeli (SLIM), Larry Putnam'in kendi şirketi olan QSM (Quantitative Software Management) şirketinin Putnam modeline dayalı olarak geliştirdiği özel araçlara verdiği isimdir. Putnam'ın modelinin ana kısmı aşağıdaki 2.1 numaralı denklemi kullanır.

$$S = C_k \times (Efor)^{1/3} t_d^{3/4} \quad (2.1)$$

Burada, td değişkeni yazılım teslim süresidir, C değişkeni yazılım geliştirme yeteneğini yansıtan çevre faktörüdür. S değişkeni, boyutu, boşluklar ve yorumlar olmaksızın yeni ve değiştirilmiş programlama satırlarını içeren etkili kaynak kodlarını (ESLOC) ifade etmektedir. Efor, projeye uygulanan toplam çabadır. İkinci önemli ilişki ise 2.2 numaralı formülde gösterilmiştir.

$$Efor = C \,/\, t_d{}^4 \tag{2.2}$$

Burada C tamamen yeni bir yazılım ya da yeniden oluşturulmuş yazılım arasında değişen insan gücünü ifade eden bir parametredir (0 – 8, 8 – 11 arasında değerler almaktadır) ve t ise E/td4 ile elde edilen sabit bir değerdir. Diğer sistemlerle etkileşime giren belirli bir büyüklükteki yeni sistemler, en büyük entropiye sahip olacak ve gelişmesi daha uzun sürecektir. Mantığın ve kodun büyük bölümlerinin zaten geliştirildiği (ve dolayısıyla entropiyi azalttığı) eski sistemlerin veya kompozitlerin yeniden inşa edilmesi en az zaman alacaktır.

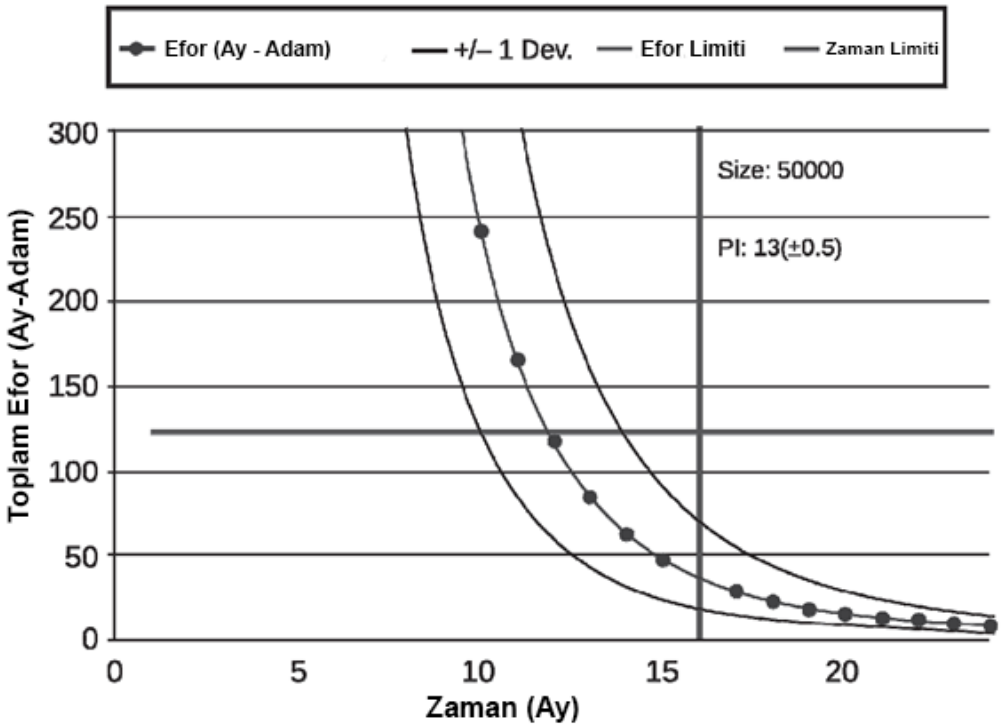
Bu iki denklemi birleştirilerek, yazılım maliyet tahmini için 2.3 ve 2.4 numaralı formüller elde edilir.

Çevre faktörü, *E*, ölçekleme faktörü *B* olarak ifade edilmektedir. Ölçekleme faktörü, proje boyutunun bir fonksiyonudur.

$$\text{Efor} = (D^{4/7} \times E^{-9/7}) \times S^{9/7} \tag{2.3}$$

$$t_d = (D^{-1/7} \times E^{-3/7}) \times S^{3/7} \tag{2.4}$$

Putnam Modeli, yazılım geliştirme eforunu aşağıdaki Şekil 2.5'te de gösterildiği gibi, eforu bir zaman fonksiyonu ile göstermektedir [23]. Eğri boyunca verilen noktalar, projeyi bir süre sonra tamamlamak için tahmini olarak toplam eforu göstermektedir.



Şekil 2.5 Putnam Modelinde Zaman Fonksiyonu Olarak Yazılım Geliştirme Eforu. ("Dev." Geliştirmeyi ifade etmektedir.)

Putnam modeli, tamamlanmış projelerden elde edilen verilerle kalibre edilebilir veya veriler mevcut değilse bir dizi kalibrasyon sorusu ile yine kalibre edilebilir. Modelin en önemli avantajlarından biri, kalibre edilmesindeki basitliktir. Çoğu yazılım kuruluşu, olgunluk düzeyine bakılmaksızın, geçmiş projeler ait boyut, efor ve süre (zaman) verisini toplayabilir. Bunları da modellerin oluşturulmasında kullanabilir.

2.2.2.2 Fonksiyon Noktası Analizi (Function Point)

Fonksiyon noktaları (Function Point) ve fonksiyon noktası analizi (FPA) Allan Albrecht [24] tarafından geliştirilmiştir. Albrecht, yazılım geliştirmede üretkenliği ölçmek için bir yöntem aramakta iken ve fonksiyonel nokta analizi, kod satırı sayısına alternatif bir ölçü olarak geliştirilmiştir. Bir işlev noktası (function point), bir bilgi sisteminin bir kullanıcıya sağladığı iş işlevselliği miktarını ifade eden bir ölçüm birimidir. Fonksiyonel nokta analizi, yazılımdaki fonksiyonların sayısına dayanan basit bir yöntemdir. Programlama dili veya dördüncü nesil araçlardan bağımsızdır. Yöntemde, yazılımın fonksiyonları tanımlanır. Her fonksiyon kendi karakteristik özelliklerine bağlı olarak oluşturulan kategorilere atılır.

- Harici (External) giriş tipi
- Harici (External) çıkış tipi
- Harici (External) sorgulama türü
- Mantıksal (Logic) iç (internal) dosya türü
- Harici (External) arayüz (interface) dosya tipi.

Fonksiyonlar tanımlandığında ve kategorilere ayrıldıktan sonra, karmaşıklık (basit, ortalama veya karmaşık) için değerlendirilir ve karmaşıklığa ve türüne göre bir dizi fonksiyon puanı atanır. Tüm işlevler için toplam fonksiyon puanı hesaplanır. Fonksiyonların boyutları için 14 teknik özellik kullanılarak değerler belirlenir. Tek bir birimin yazılım geliştirme maliyeti (para veya saatler) geçmiş projelere ve teknik özelliklere bakılarak değeri hesaplanır.

Tablo 2.4 Fonksiyon Nokta Analizine Genel Bakış [24]

#	**Genel Sistem Özellikleri**	**Kısa Açıklama**
1	Veri iletişimleri	Sistemin uygulaması ile bilgi değişimi veya transferinde yardımcı olmak için kaç tane iletişim aracı vardır?
2	Dağıtılan veri/işleme	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 işlem 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?

Tanımlanan her fonksiyon, giriş için veri girişi veya sorgu için kullanıcı sorgusu gibi özelliklerden oluşur. Bu nedenle yukarıdaki Tablo 2.4'te ifade edilen performans kriterlerine göre ayrılmalıdır ve bu ayrım önemlidir. Çünkü fonksiyon puan haritalarında fonksiyonların puanlaması bu ayrıma göre gerçekleşmektedir. Daha sonra işlev noktası analizi tekniği birkaç kez rafine edilmiş ve / veya geliştirilmiştir. Örneğin, fonksiyon noktası analizi, fonksiyon noktalarının tekniğine içsel işlevleri de dâhil edip yeni bir yöntem olarak sunulmuştur [18]. Bir algoritma, önemli bir hesaplama problemini çözmek için tamamen ifade edilmesi gereken kurallar kümesi olarak tanımlanmıştır. Her algoritmaya 1 (temel) ile 10 (karmaşık algoritmalar) arasında değişen bir ağırlık verilmiştir. Özellik noktası, algoritmaların artı fonksiyon noktalarının ağırlıklı toplamıdır. Bu ölçüm özellikle az giriş/çıkış ve yüksek algoritmik karmaşıklığı olan sistemler için kullanılmaya başlanmıştır.

2.2.2.3 Yapı Maliyeti Modeli (Cocomo)

Yapısal maliyet modeli, Barry Boehm [3] tarafından geliştirilen bir algoritmik maliyet tahmin tekniğidir. 1970'lerde TRW Aerospace firmasında 63 yazılım geliştirme projesinin bir araştırmasına dayanmaktadır. Bu yazılım geliştirme projelerinde, yazılım geliştirme şelalesi modelini, Assembly ve COBOL gibi prosedürel programlama dillerini kullanıldı. 1990'larda Cocomo, modern yazılım geliştirme projeleri ve süreçlerini tahmin etme gerekliliklerini karşılamak için Cocomo II'ye yükseltilmiştir [25]. Cocomo, parametrelerin geçmiş proje verilerinden türetildiği ve mevcut proje özelliklerine göre ayarlandığı basit bir regresyon formülü kullanır. Orijinal Cocomo'nun gittikçe daha ayrıntılı ve doğru formları vardır (Basic, Intermediate ve Detailed). Cocomo II ve formları aşağıdaki alt bölümlerde açıklanmıştır.

2.2.2.4 Yazılım Geliştirme Modları

Cocomo, üç farklı yazılım geliştirme modunu tanımlar. Bunlar organik, yarı müstakil (semi-detached) ve gömülüdür. Farklı yazılım geliştirme modları, formda benzer olan, ancak aynı büyüklükteki yazılım ürünleri için farklı maliyet tahminleri sağlayan ve dolayısıyla bu farklı modları ayırt etmek için önemli olan maliyet-tahmin ilişkisine sahiptir. Organik modda, nispeten küçük ekip, birbirine oldukça aşina, kurum içi bir ortamda yazılım geliştirir. Proje ile bağlantılı çoğu insan,

organizasyon içindeki ilgili sistemlerle çalışma konusunda kapsamlı deneyime sahiptir ve geliştirilmekte olan sistemin kuruluşun hedeflerine nasıl katkıda bulunacağına dair kapsamlı bir birikime sahiptir. Bir organik mod projesi, yazılımın gereksinimlerini karşılandığı, organik modlu bir projede daha yüksek verimliliğin sağlandığı ve daha küçük bütçesi olan projeler için basit bir ara yüzün sunulduğu moddur. Organik mod yazılım projelerinin diğer özellikleri şunlardır:

- yeni donanım ve fonksiyonel prosedürlerin gelişimin aynı anda olması gerekmediği durumlar
- yenilikçi veri işleme mimarileri veya algoritmaları için minimum gereksinim olduğu durumlar
- projenin erken tamamlanmasında düşük bir prim olan durumlar
- göreceli küçük boyut (<50 KLOC) olan durumlar (bu faktörler genellikle daha yüksek proje verimliliği ve daha küçük proje ekonomilerini ifade eder.)
- yarı müstakil (semi-detached) yazılım geliştirme modu
- proje karakteristiğinin orta seviyesi
- organik ve gömülü mod özelliklerinin bir karışımı.

Gömülü mod, çok sıkı kısıtlamalar içinde çalışması gereken faktörler tarafından tetiklenen projeler için ifade edilen moddur. Yazılım ürünü, güçlü bir şekilde birleştirilmiş karmaşık donanım, yazılım, yönetmelikler ve işletim prosedürlerinden oluşmaktadır. Genel olarak, bu kompleksin diğer bölümlerini değiştirme maliyetleri o kadar yüksektir ki, karakteristik özellikleri değişmez olarak kabul edilir. Yazılımın hem şartnamelere uygun olması hem de karşılaşılan ve öngörülemeyen güçlükler veya diğer ihtiyaçlar dâhilinde gerekli olan değişiklikler üzerinde ciddi çalışılması beklenir. Bu faktörler hem daha düşük üretkenliğe hem de daha büyük projelerde daha fazla bütçe giderine neden olur.

2.2.2.5 Basit Cocomo

Temel Cocomo, tanıdık bir kurum içi ortamda küçük ve orta ölçekli yazılım geliştirme projelerine uygulanabilir. Kullanımı hızlı ve kolaydır, ancak sadeliği nedeniyle doğruluğu çok sınırlıdır. Yazılım maliyetlerinin erken ve kaba tahmini için kullanılabilir.

Temel Cocomo, yazılım geliştirme eforunu (ve maliyetini) yazılım boyutunun bir fonksiyonu olarak tahmin eder. Yazılım boyutu, kod satır sayısının (kilo line of code) bir fonksiyonu olarak ifade edilir.

Temel Cocomo denklemleri aşağıdaki denklemlerin formundadır:

$$\text{Efor (ayAdam)} = a_b \text{ x } (\text{KLOC})b_b$$

$$\text{Geliştirme Zamanı (Ay)} = c_b \text{ x*}(\text{Efor})d_b \qquad (2.5)$$

$$\text{Verimlilik} = \text{KLOC} / \text{Efor}$$

$$\text{Ortalama istihdam} = \text{Efor} / \text{Geliştirme Zamanı}$$

a_b, b_b, c_b ve d_b katsayıları proje modu için kullanılan sabit değerlerdir ve aşağıdaki Tablo 2.5'te aldıkları değerler gösterilmiştir. Ayrıca, ayAdam ise yazılım maliyetlendirilmesinde kullanılan ve Cocomo modeli tarafından sunulan maliyet birimidir.

Tablo 2.5 Basit Cocomo Bileşenleri [3]

Yazılım Proje Türü	a_b	b_b	c_b	d_b
Organic	2,4	1,05	2,5	0,38
Semi-detached	3,0	1,12	2,5	0,35
Embedded	3,6	1,2	2,5	0,32

Temel Cocomo, yazılım geliştirme maliyetleri üzerinde önemli bir etkisi olabilecek donanım kısıtlamaları, personel kalitesi, deneyim ve diğer proje özellikleri gibi maliyet faktörlerini hesaba katmaz.

2.2.2.6 Orta Düzey (Intermediate) Cocomo

Orta düzey Cocomo, yazılım geliştirme eforunu yazılımın boyutunun bir fonksiyonu olarak ve bir dizi 15 maliyet faktörü ile birlikte tahmin eder. Bu maliyet faktörü özellikleri dört kategoriye ayrılır: yazılım ürün özellikleri, bilgisayar öznitelikleri, personel öznitelikleri ve proje öznitelikleri. Maliyet çarpan faktörünün değeri (EAF), her bir özniteliğin yazılım geliştirme çabası üzerindeki etkisinin (değerinin) çarpılması ile elde edilir.

Orta düzey Cocomo tahmininde, temel Cocomo'da olduğu gibi aynı formun ölçeklenmiş denklemleri kullanılarak bir efor tahmini oluşturularak başlanır. Bu efor tahmini daha sonra projenin derecelendirmelerinden diğer 15 maliyet faktörüne göre belirlenen çaba çarpanları uygulanarak ayarlanır. 15 özniteliğin her biri, "çok düşük" ile "çok yüksek" arasında değişen bir önem derecesi üzerinde bir faktör değeri alır. Aşağıdaki Tablo 2.6'da her bir çaba çarpanının aldığı değerler listelenmiştir. Tüm çaba çarpanlarının ürünü, EAF değerlerinin hesaplanması ile elde edilir. EAF için tipik değerler 0,9 ila 1,4 arasındadır. Tablo 2.6'da gösterilen özelliklere ait ayrıntılı bilgi bölüm 4'te verilecektir.

Tablo 2.6 Orta Cocomo'nun Efor Çarpanları

Özelliker	Özellikler	Çok Aşağı	Aşağı	Nominal	Yüksek	Çok Yüksek	Extra Yüksek
Ürün Özellikleri	Rely	0,75	0,88	1,00	1,15	1,40	
	Size		0,94	1,00	1,08	1,16	
	Cplx	0,70	0,85	1,00	1,15	1,30	1,65
Donanım Özellikleri	Time			1,00	1,11	1,30	1,66
	Stor			1,00	1,06	1,21	1,56
	Data		0,87	1,00	1,15	1,30	
	Turn		0,87	1,00	1,07	1,15	
Personel Özellikleri	Acap	1,46	1,19	1,00	0,86	0,71	
	Axep	1,29	1,13	1,00	0,91	0,82	
	Virt	1,42	1,17	1,00	0,86	0,70	
	Vexp	1,21	1,10	1,00	0,90		
	Lexp	1,14	1,07	1,00	0,95		
Proje Özellikleri	Modp	1,24	1,10	1,00	0,91	0,82	
	Tool	1,24	1,10	1,00	0,91	0,83	
	Sched	1,23	1,08	1,00	1,04	1,10	

Orta düzey(intermadiate) Cocomo formülü aşağıda 2.6 numaralı formülde gösterilen forma sahiptir.

$$E = a_i \times (KLOC)^{B_i} \times EAF \qquad (2.6)$$

E'nin ayAdam olarak uygulanan eforu ifade ettiği yerde, KLOC, proje için tahmin edilen binlerce kod satırı sayısının tahmini değeridir ve EAF, maliyet faktörlerine göre hesaplanan faktördür. Geliştirme süresi, D, hesaplanan E'yi Basic Cocomo'daki gibi kullanır. Orta Cocomo, geliştirme programı, faz dağıtımı ve Temel Cocomo olarak etkinlik dağıtımı için aynı tahmin ilişkisini kullanır. Tablo 2.7'de orta Cocomo bileşenlerinin proje türlerine göre aldığı değerler gösterilmiştir.

Tablo 2.7 Orta Cocomo Bileşenleri

Proje Türü	a_b	b_b	c_b	d_b
Organik	3,2	1,05	2,5	0,38
Yarı-Müstakil	3,0	1,12	2,5	0,35
Gömülü	2,8	1,2	2,5	0,32

2.2.2.7 Detaylı Cocomo

Detaylı Cocomo, Orta düzey Cocomo'nun tüm özelliklerini, maliyet faktörlerinin proje aşamaları üzerindeki etkisinin tek tek değerlendirmesiyle oluşturulur. Bu, her bir fazdaki her bir maliyet faktörleri özniteliği için farklı çaba çarpanları kullanılarak yapılır. Bu çarpanlara Faz Hassasiyet Çaba Çarpanı adı verilir ve bunlar projenin her aşamasını tamamlamak için gereken çaba miktarını belirler.

2.2.2.8 Cocomo II

Yazılım mühendisliği değiştikçe Boehm, Cocomo modelini geliştirmeye devam etmiştir ve Cocomo II, 2000 yılında yayınlanmıştır [25]. Cocomo II, kaynak kodu satır sayısını ve fonksiyon noktaları değerini giriş parametresi olarak kullanan fonksiyonlardan oluşur. Uygulamalar Kompozisyonu, Erken Tasarım ve Post-Tasarım olmak üzere üç farklı Cocomo II varyasyonu vardır.

Uygulama Kompozisyonu Modeli, hızlı uygulama geliştirme için Entegre Bilgisayar Destekli Yazılım Mühendisliği araçlarını kullanan projeler üzerinde çaba ve zamanlamayı tahmin etmek için kullanılır. Bu projeler çok çeşitlidir, ancak birlikte çalışabilir bileşenlerden hızla oluşması için yeterince basittir. Tipik bileşenler GUI oluşturucuları, veri tabanı veya nesne yöneticileridir. Dağıtık işlemlere ve fonksiyonel işlemlere göre hareket eder. Ayrıca finansal, tıbbi veya endüstriyel

süreç kontrol paketleri gibi alana özgü bileşenleri göz önünde bulundurarak özelleşir. Uygulama Kompozisyonu Modeli, İşlev Noktası (function point) Analizi 'ne benzer nesne noktası tahminine dayanmaktadır. Erken Tasarım Modeli alternatif sistem mimarilerinin ve operasyon kavramlarının araştırılmasını içerir. Bir yazılım projesinin ilk aşamalarında, geliştirilecek olan yazılımın boyutu ve niteliği hakkında çok az şey bilinebileceği durumlarda kullanılır. Genellikle ayrıntılı bir tahminde bulunmak için yeterli bilgi mevcut değildir. Bu nedenle model, fonksiyon noktalarına (veya mevcut olduğunda kod satırlarına) ve beş ölçek faktöründen ve yedi çaba çarpanından (maliyet faktörleri) oluşan bir dayanağa dayanmaktadır.

Post-Tasarım modeli, bir yazılım ürününün gerçek gelişim ve bakımını içerir. Üst düzey tasarım tamamlandığında ve proje hakkında detaylı bilgi mevcut olduğunda kullanılır. Tüm geliştirme yaşam döngüsünü tahmin etmektedir ve bu duruma göre Erken Tasarım Modeli ayrıntılı bir şekilde ele alınmıştır. Bu model Orta Düzey Cocomo'81'e benzer. Yeniden kullanım için ayarlanmış boyutlandırma parametresi kullanır. Ayarlanmış boyutlandırma için Kod ve/veya Fonksiyon Noktalarının Kaynak Değerleri kullanılır. Bir dizi gelişmekte olan yazılımın ekonomilerini belirleyen 15 çalışma çarpanı ve beş ölçek faktör kümesi bu modelin getirdiği ve kullandığı yeniliklerdendir. Beş ölçek faktörü Cocomo'81 modelindeki geliştirme modlarının yerini almaktadır.

Tablo 2.8 Cocomo II Bileşenleri

Özelliker	Özellikler	Çok Aşağı	Aşağı	Nominal	Yüksek	Çok Yüksek	Extra Yüksek
Örün Özellikleri	Rely	0,75	0,88	1,00	1,15	1,40	
	Size		0,94	1,00	1,08	1,16	
	Cplx	0,70	0,85	1,00	1,15	1,30	1,65
Platform Özellikleri	Time			1,00	1,11	1,30	1,66
	Stor			1,00	1,06	1,21	1,56
	Data		0,87	1,00	1,15	1,30	
	Turn		0,87	1,00	1,07	1,15	
Personel Özellikleri	Acap	1,46	1,19	1,00	0,86	0,71	
	Axep	1,29	1,13	1,00	0,91	0,82	
	Virt	1,42	1,17	1,00	0,86	0,70	
	Vexp	1,21	1,10	1,00	0,90		
	Lexp	1,14	1,07	1,00	0,95		
Proje Özellikleri	Modp	1,24	1,10	1,00	0,91	0,82	
	Tool	1,24	1,10	1,00	0,91	0,83	
	Sched	1,23	1,08	1,00	1,04	1,10	

Cocomo II'nin Efor denklemi:

$$Efor\ (ayAdam) = \ A\ x\ boyut \prod_{i}^{15} EMi\ S \qquad (2.7)$$

A sabit, boyut KLOC (Kilo Line Of Code) veya Fonksiyon Noktalarının (Function Points) değerine bakılarak elde edilen değer, (EMi- environment multiplier) çaba çarpanlarının değeri ve S (scale) ise ölçek Faktörü olarak ifade edilir.

Sabit A, üretkenlik boyutunu (standart değer 2.94) gösteren bir kalibrasyon faktörüdür, ancak şirketin geçmiş proje verileriyle kalibre edilebilir.

Ölçek faktörleri (S), beş faktöre bağlıdır: gelişme esnekliği, mimari / risk çözümü, takım uyumu, süreç olgunluğu ve süreç önceliği. Ölçek faktörleri, bir yazılım geliştirme projesinin çabasını üstel olarak etkiler. EMi maliyet faktörleridir. Çarpma

çarpanları, yazılım geliştirmenin özellikleridir ve çaba üzerinde doğrudan bir etkiye sahiptir. Çarpma Çarpanları, çok düşükten çok yükseklere kadar değişen kategorilere ayrılır. Bu kategorilere sayısal değerler verilmiştir.

2.2.3 Uzmanlık Tabanlı Teknikler

Uzmanlık tabanlı tekniklerde, yazılım maliyet tahminleri uzman görüşleri ve yargısal süreçlere dayanmaktadır. Bu teknikler, bir ilgi alanı içinde aşina olunan konu uzmanlarının bilgi ve deneyimlerini kullanır ve geçmiş tüm projelerin ve teorinin bilinen sonuçlarının bir sentezine dayanan tahminler sağlar.

Uzmanlık temelli teknikler, daha önce benzeri görülmemiş projeler için ve geçmiş projelerin nicel, ampirik verileri olmadığı durumlarda kullanılan bir yöntemdir. Uzmanlar, geçmiş proje deneyimleri ile gelecekteki projede yer alan yeni teknikler, mimariler veya uygulamalar arasındaki farkları belirleyebilir. Ayrıca, istisnai personel özelliklerini ve etkileşimlerini veya diğer benzersiz proje değerlendirmelerini de dikkate alabilirler. Tahmincinin yazılım alanında ve tahminde önemli bir deneyime sahip olması durumunda uzman kararı da nispeten doğru bulunmuştur [26].

Zayıf yönler arasında duyarlılık analizi eksikliği, tecrübeli tahmincilere bağımlılık, insan hataları ve önyargılardır. Ayrıca iyimser, kötümser ya da projenin temel yönlerine aşina olmayan tahmincilerin objektifliği bulunmayabilir. Aşağıdaki alt bölümler de uzmanlığa dayalı teknik açıklanmıştır.

2.2.3.1 Delphi Tekniği

En ünlü uzmanlık tabanlı tekniklerden biri Delphi tekniğidir. Delphi'nin eski kurucularından adını alan Delphi tekniği, toplantılar, anketler ve anketler yoluyla uzman grup arasında bir konsensüs geliştirmeyi amaçlamaktadır. İlk olarak 1940'ların sonlarında Rand şirketi tarafından teknolojinin savaş üzerindeki etkisini tahmin etmek için geliştirilmiş olmasına rağmen [27], yazılım geliştirme projeleri için maliyet tahmini dahil olmak üzere birçok başka alanda kullanılmıştır.

Orijinal Delphi tekniği grup tartışmasını engellemiştir, ancak Geniş Bant Delphi tekniği her değerlendirme turu arasında grup tartışmasını desteklemiştir [3].

Geniş bant Delphi tekniği, bir yazılım geliştirme projesinin maliyet tahmini için aşağıdaki şekilde kullanılabilir:

- Proje yöneticisi, her uzmana ürün özelliklerini ve bir tahmin formunu sunar.
- Proje yöneticisi, tahmin sorunlarını tartışacakları formlar hazırlar ve bir grup toplantısı için çağrı yapar.
- Uzmanlar anonim olarak formları doldurur.
- Proje yöneticisi tahminlerin bir özetini hazırlar ve dağıtır.
- Proje yöneticisi, özellikle uzmanların tahminlerinin büyük ölçüde değiştiği noktaları tartışmaya odaklanarak bir grup toplantısı düzenlemektedir.
- Uzmanlar tekrar anonim olarak formları doldurur ve 4 ila 6 arasındaki adımlar ortak karara varana kadar birçok tur yinelenir.

Yinelemede, uzmanlar tahmin edebilecekleri bir tahminde bulunmak için yaptıkları tahminlerin aralığını küçültürler. Eğer tam bir anlaşmanın imkânsız olduğu durum olursa, oldukça güvenilir bir tahmin olacak olanı elde etmek için ortalama puanlar kullanılabilir.

Delphi metodunu diğer metotlardan ayıran başlıca özellikleri aşağıdaki şekilde sıralanabilir:

- Katılımcıların kimliğini açıklamaması: Tahmin yöntemlerinde, özellikle de kalitatif tahmin yöntemlerinde katılımcıların kimlikleri ile yaptıkları tahminlerin kimliğin vermiş olduğu yükü taşıması ve bu yüzden tahminlerin aynı bilgiye sahip aynı kişi tarafından farklı rollerde değiştiği görülmüştür. Ayrıca grup içerisindeki güç ilişkilerinin (power relation) tahminleri etkilediği, örneğin ast/üst ilişkisine göre üstlerin kararlarından çok farklı kararları astların vermekten çekindiği, bu şekildeki farklı tahminlerin "aykırı" veya "cesur" olarak düşünüldüğü görülmüştür. Ayrıca toplumsal güç ilişkilerinin kuvvetli olduğu gruplarda, ilk tahminden sonraki tahminlerin ilk tahmine karşı bir tepki olarak anlaşıldığı da görülmüştür. Bütün bu sebeplerden dolayı bazı gruplarda kişiliğin gizli tutularak tahmin yapılmasının, katılımcıların eşit etki hakkına sahip olması, birbirini etkilememesi ve grup içerisindeki güç ilişkilerini kaldırması açısından önemi olduğu düşünülmektedir. Delphi yöntemi katılımcıların grup içerisindeki

rollerinden çok fikirlerinden emin olup olmaması ile ilgilenmektedir. Delphi yöntemi, tahmini konusunda emin olan kişiler tahminlerinde sabit kalırken emin olmayan kişilerin tahminlerini değiştireceği kabulü üzerine kuruludur.

- Bilgi akışının yapılandırılması. Tahmin sırasında, toplanan bilginin bir yönetimin süzgecinden geçme imkânı vardır. Bu sayede yönetim bir sonraki turu yönlendirebilmekte, gelen tahminleri istediği gibi birleştirebilmekte ve hatta tahminlerden aykırı olanları eleyebilmektedir. Birleştirme yöntemi olarak, ortalama, ortanca, ağırlıklı ortalama gibi farklı yöntemler kullanılabilmekte ve buna panel yönetimi karar vermektedir.
- Geri bildirimlerin bireysel kontrolü. Tahmin sürecine dahil olan uzmanların kendi tahminlerini değerlendirmesi gerekmektedir. Her turun sonunda ortalama değerin duyurulması ile tahmin eden kişilerin kendi tahminlerini değiştirme veya değiştirmeme veya ne kadar değiştireceğine karar verme imkânı oluşur.
- Yönetici rolü tanımlanmıştır. Paneli yöneten kişinin konu üzerindeki uzmanlığına bağlı olarak tahminleri yönlendirme veya yönlendirmeme imkânı vardır. Ayrıca tahmin sorusunun hazırlanması ve aynı tahminin farklı sorularla sorulmasının da katılımcılar üzerinde etki yaptığı bilinmektedir. Buna göre tahminin bir tez ve bunu karşılayan bir anti tez üzerinden üretilmesi veya tahminin sağlıklı şekilde turlar arasında devam ettirilmesi üzerinde yöneticinin etkisi bulunmaktadır.

2.2.3.2 İş Dökümü Yapısı (WBS-Work Breakdown Structure)

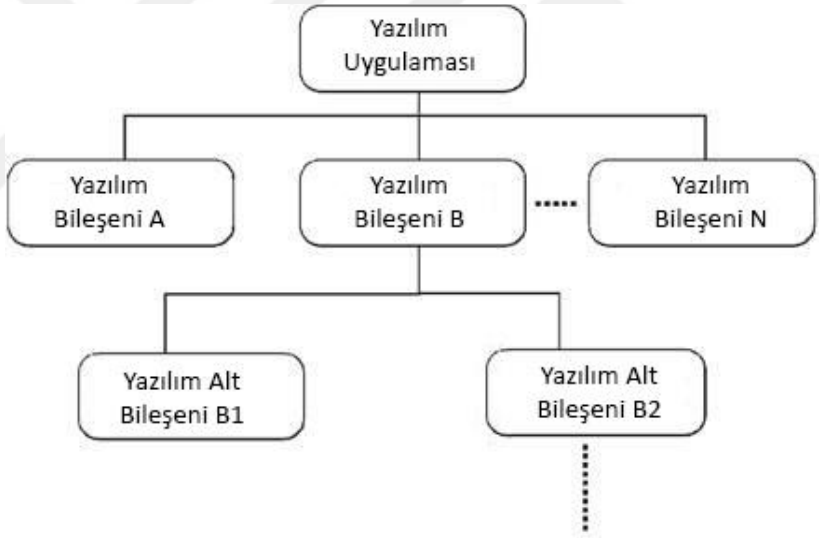
Yazılımın geliştirilmesinde neredeyse bir mühendislik uygulaması standardı olan İş Dökümü Yapısı (İDY), yazılım maliyet tahmininde de kullanılır. Projeyi, proje öğelerinin bir toplamı olarak düzenleyen ve belli bir hiyerarşiye dayalı olarak gruplandıran yöntemdir [11].

İDY, nihai amaçtan başlayarak ve bunu başarmak için gerekli tüm adımları içeren boyut, süre ve sorumluluk (sistemler, alt sistemler, bileşenler, görevler, alt görevler ve iş paketleri) açısından yönetilebilir öğelere ayrıştırmak suretiyle geliştirilmiştir. Hiyerarşideki her iniş düzeyi, proje çalışmasının giderek daha ayrıntılı bir tanımını temsil eder. En alt düzeydeki unsurlar, iş paketleri, belirli bir kişi veya kuruluşa

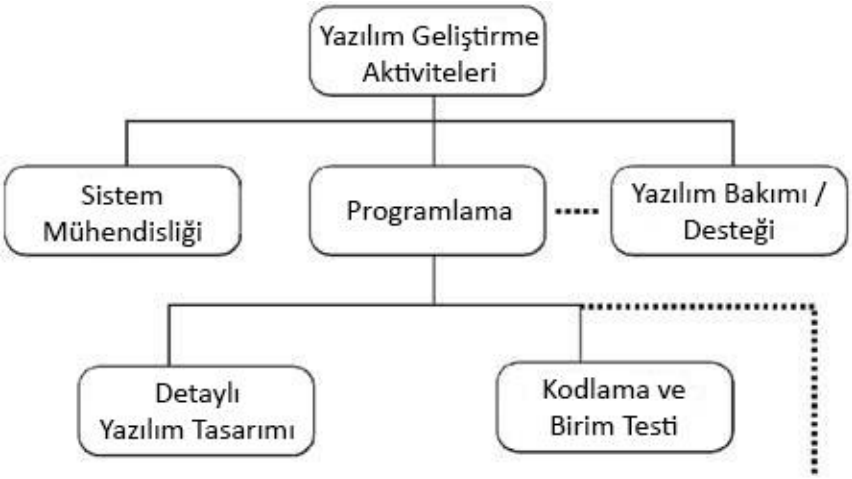
faaliyetlerin tanımlanması veya sorumlulukların atanması için mantıklı bir temel sağlar.

İDY, hangi iş ürünlerinin gerekli olduğunu ve hangi maliyetlerin hesaplandığını belirlemeye yardımcı olur. Bu tanımlamalar olmadan, yazılım maliyeti tahminleri anlamsız hale gelir. Maliyetler her bir iş paketi ile ilişkilendirilir. Toplam proje geliştirme maliyeti için aşağıdan yukarıya doğru toplam değer belirlenebilir. İDY yöntemi uzmanlık gerektirir. Bu uzmanlık ile iş paketlerinin yapı içindeki fonksiyonu ve böylece her iş paket ile ilgili maliyet tahmini yapılabilir.

Bir yazılım İDY'si, biri yazılım ürününü temsil eden, diğeri de bu ürünü üretmek için gerekli olan faaliyetleri temsil eden iki yapıdan oluşur [3]. Ürün hiyerarşisi (Şekil 2.6), çeşitli yazılım bileşenlerinin genel sisteme nasıl uygulandığını gösteren yazılımın temel yapısını ifade eder. Aktivite hiyerarşisi (Şekil 2.7), belirli bir yazılım bileşeniyle ilişkilendirilebilecek etkinlikleri gösterir.



Şekil 2.6 Ürün Hiyerarşisi



Şekil 2.7 Aktivite Hiyerarşisi

İDY'yi geliştirmek dört aşamalı bir süreçtir [28]:

- Proje hedeflerini belirlemek ve müşteriye sağlanacak ürün, hizmet veya sonuçlara odaklanmak
- Müşteriye sağlanacak ürün, hizmet ve sonuçları (teslim edilen ürünler veya ürünler) belirlemek
- Projedeki diğer çalışma alanlarını belirlemek, çalışmanın yüzde 100'ünün kapsandığından emin olmak ve çıktıları engelleyen alanları belirlemek. Ara çıktıları temsil etmek veya çıktıları tamamlamak

Her bir öğe ardışık, mantıklı alt kategorilere ayrılır. Öğelerin karmaşıklığı ve para değeri, planlama ve kontrol amaçları için yönetilebilir birimler haline getirilir.

%100 kuralı, İDY'nin geliştirilmesi, ayrıştırılması ve değerlendirilmesine yön veren en önemli ilkelerden biridir. %100 kuralı ile proje kapsamı ifade edilir. Ayrıca İDY'nin, proje yönetimi de dahil olmak üzere, tamamlanacak çalışma açısından iç, dış ve ara dönemdeki tüm çıktıları yakaladığını gösterir. Kural, hiyerarşi içindeki tüm seviyelerde geçerlidir: "çocuk-child" seviyesindeki çalışmanın toplamı "ebeveyn-parent" tarafından temsil edilen çalışmanın%100'üne eşit olmalıdır ve IDY'nin, gerçek kapsamın dışında kalan herhangi bir çalışmayı içermemesi gerekir. Proje, yani işin %100'ünden fazlasını içeremez. %100 kuralının etkinlik seviyesi için de geçerli olduğunu hatırlamak da önemlidir. Her çalışma paketindeki faaliyetlerle

temsil edilen çalışma, iş paketini tamamlamak için gerekli işin %100'üne kadar olmalıdır [Haugan, 2001].

Ayrıca İDY aşağıdaki listelenen faydaları sağlar [29].

- Ürün bileşen parçalarına ayrılır. Parçaların arasındaki ilişkiyi açıklığa kavuşturulur. Tamamlanması gereken görevlerin birbirleriyle ve nihai ürünle ilişkisi belirlenir.
- Yönetim ve teknik sorumlulukların etkili bir şekilde planlanması yapılır.
- Teknik çabaların, risklerin, kaynak tahsisatlarının, harcamaların ve maliyet / zamanlama / teknik performansın durum takibine yardımcı olur.
- Yüklenicilerin maddi gerekliliklerini karşılamada gereksiz yere kısıtlanmadığından emin olunmasına yardımcı olur.

İDY'nin başka bir kullanımı, yazılım maliyetini belirlemede ve raporlamada temel olarak hizmet vermesidir. İDY elemanlarının her birine, bir proje bütçesi ve insanların farklı proje aktivitelerine harcadıkları zaman miktarını bildirmede kullanacakları bir iş numarası verilebilir. Bir kuruluş, tüm projeleri için sürekli olarak standart bir İDY kullanıyorsa, kuruluş zaman içinde yazılım geliştirme maliyetinde kullanacak değerli bir veri tabanı oluşturacaktır. Bu veriler, kuruluşun yazılım maliyetinde oluşturacağı veri setinde ve oluşturacağı parametrik yazılım maliyet tahmini yöntemlerinde kullanılabilir. [3].

2.2.3.3 Analoji Tabanlı Yöntemler

Analoji ile tahmin, bir projeyi veya bunun bazı yönlerini benzer bir projeyle karşılaştırarak yapılır [3]. Karşılaştırma yapabilmek için benzer projelerden faydalanılır. Ayrıca birden fazla proje de bu karşılaştırmada kullanılabilir.

Örneğin, bir kuruluşun günde 5 milyon çevrim içi alışveriş işlemini gerçekleştirebilecek yazılım maliyetlerini tahmin etmek istediği varsayılsın. Kuruluş, daha önce, günde 10 milyon çevrim içi alışveriş işlemini gerçekleştirebilen başka bir müşteri için benzer yazılımlar geliştirmiş olsun. Benzerliği kullanarak, maliyetlerin, önceki projenin gerçek maliyetlerinin kabaca%50'si olduğu tahmin edilebilir. Fakat bu kabaca doğru görünse de gerçekte doğru değildir.

Analoji ile tahmin, proje düzeyinde ya da bir alt sistem düzeyinde yapılabilir. Proje seviyesi, alt sistem seviyesi, yeni proje ile tamamlanan proje arasındaki benzerlik ve farklılıkların daha ayrıntılı bir değerlendirmesini sağlama avantajına sahip olmasının yanında ayrıca sistem maliyetinin tüm bileşenlerinin göz önünde bulundurulması avantajına da sahiptir.

Analoji tabanlı maliyet tahminin temel gücü, tahminin geçmiş projelerdeki gerçek deneyime dayanmasıdır. Bu deneyim, yeni projeden belirli farklılıkları ve olası maliyet etkilerini belirlemek için kullanılabilir. Analoji tabanlı maliyet tahminin temel zayıflığı, eski projedeki kısıtlamaların, tekniklerin, personelin ve fonksiyonların yeni projede ne ölçüde temsil edildiğinin net olarak bilinmemesidir [3].

Shepperd ve Schofield, yazılım maliyet kestiriminde, analoji ile tahmin için beş aşamalı bir süreç ve yazılım aracı olan ANGEL (ANaloGy Estimation tool2) tekniğini ve aracını geliştirmiştir [30]. Bu teknik aşağıdaki adımları içerir:

- Toplanacak veriler veya özellikler tanımlanır.
- Veri toplama kuralları tanımlanır.
- Vaka (olay- işlev) tabanı doldurulur.
- Tahmin yöntemi ayarlanır.
- Yeni bir proje için çaba tahmin edilir.

ANGEL tekniğinde, organizasyonun tahmin edilmesi gereken yeni bir projesi için benzer tamamlanmış projeleri bulmaya çalışılır. Bu projeler bulunduğunda, geliştirme çabaları bilinecek ve yeni proje için çaba tahmini için bir temel olarak kullanılabilecektir. Benzerlikte, ara yüz sayısı, geliştirme yöntemi, uygulama alanı ve benzeri gibi proje özellikleri ele alınır. Açıkça kullanılan özellikler, projeleri karakterize etmek için hangi verilerin mevcut olduğuna bağlı olacaktır. Özelliklerin sayısı da esnektir. Burada önemli bir problem olarak görülen durum ise bulunan benzer projeler içerisinden benzer vakaların bulunmaması durumudur.

İlk adımda, projeler arasındaki analojileri karakterize etmek için veriler toplanır. Bu adımda göz önünde bulundurulması gereken hangi faktörlerin maliyet üzerinde doğrudan etkisinin olduğunun tespit edilmesidir. Böylece kullanılacak özellikler

kolayca tespit edilmiş olur. Bu özellikler programlama dilinde fonksiyonlar veya yazılımdaki ara yüz sayısı gibi birçok özellik olabilir.

İkinci adım, veri toplama kurallarını tanımlama adımıdır. Örgütlerde bile, çaba ile neyin kastedildiği hakkında ortak bir anlayış olmayabilir. Farklı projeler aynı özellikleri farklı şekillerde değerlendirebilir. Bu nedenle verilerin nasıl ve kim tarafında toplandığı bilgisi önemlidir. Veri toplamadan kimin sorumlu olduğunu ve verileri ne zaman toplayacaklarını belirlemek de önemlidir. Bu nedenle, bazen tutarlılık düzeyini arttırmak için projede veri toplama işi aynı kişiye verilebilir.

Üçüncü adımda ise vaka tabanı doldurulacaktır. Genel olarak, çoğu durumda, veri toplama, projeler tamamlandığında ve çaba verileri kullanılabilir hale geldikçe devam eden bir süreçtir. Bununla birlikte, veri setinin büyüklüğü ve homojenliği arasında bazı farklılıklar olduğu da görülmektedir. Bazı deneyimler, son derece farklı projeleri ayrı veri kümelerine bölme stratejisinde liyakatin esas alınması gerektiğini göstermiştir.

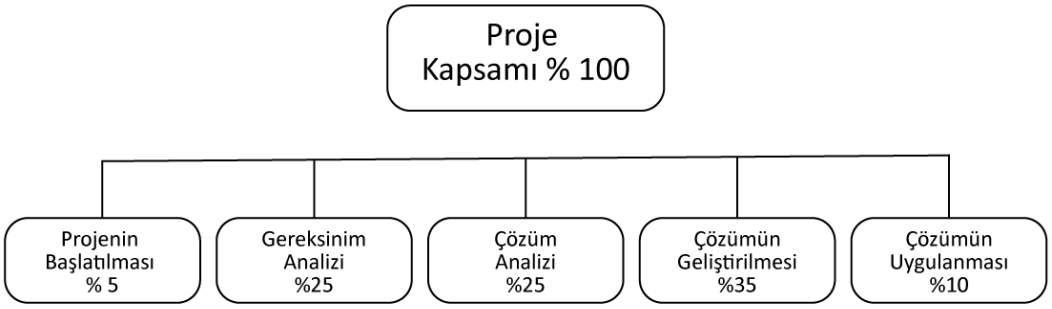
Dördüncü adım, tahmin yöntemini belirlemektir. Kullanıcının ayrıca, aranan en uygun analojiyi denemeyi ve değişkenlerin bir alt kümesinin kullanılıp kullanılmayacağını belirlemesi gerekmektedir. Çünkü bazı özelliklerin etkili analojiler bulma sürecine faydalı bir şekilde katkıda bulunmayacağından deney yapılmasını gerektirebilir. Ayarlama (tuning), tahminlerin kalitesi açısından oldukça fark yaratabilir; tipik olarak ayarlama, performansta iki kat artış sağlayabilir ve bu nedenle ANGEL aracı, bu süreç için otomatik destek sağlar.

Son adım, yeni bir proje için tahmin yapmaktır. Projeyi, tahmin sürecinin ilk aşamasında tespit edilen değişkenler açısından karakterize etmek mümkün olmalıdır. Bu değişkenlerden, ANGEL, benzer projeleri bulmak için kullanılabilir ve kullanıcı, analojilerin değerine dair öznel bir karar verebilir. Bu bağlamda ANGEL bir destek aracı olarak kullanılabilir.

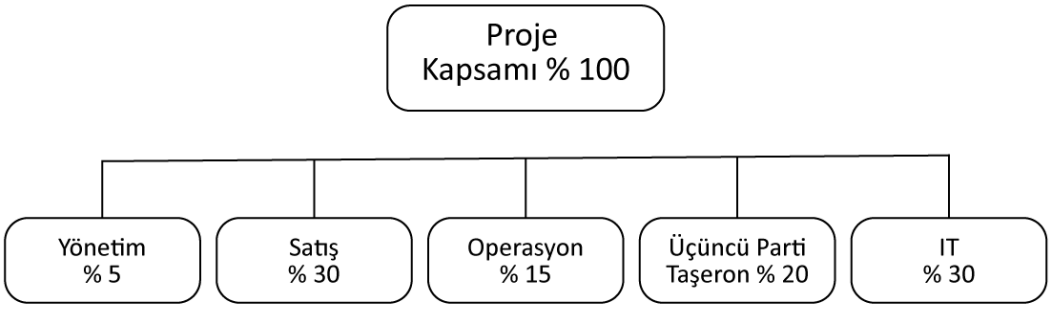
2.2.3.4 İş Dağılımı Tekniği

Bazı yazılım geliştirme projeleri, kuruluşlar içinde birçok kez tekrarlanır ve bir proje her uygulandığında benzer faaliyetler tekrarlanır. İş dağılımı tekniği bir proje için gereken eforu hesaplarken, daha önce geliştirilmiş projelerin yaşam döngüsü

boyunca harcadığı toplam efordan yaralanmaktadır [31]. İş dağılımı hem proje aşamalarından hem de her aşamada kullanılan kaynaklardan faydalanır. Bir yazılım projesinde iş dağılımının iki örneği aşağıda Şekil 2.8 ve Şekil 2.9'da sunulmuştur [31].



Şekil 2.8 İş Dağılım Tekniği



Şekil 2.9 Farklı Türlerdeki Kaynak Harcamalarının Oranları

İş dağılımı tekniğinin başarısı ve doğruluğu, geçmiş projelerin veri veya deneyimlerinin güvenilirliğine ve projelerin benzer olmasına bağlıdır. Geçmiş projelerdeki veriler zamanla kaydedilip geliştirilebiliyorsa, iş dağılımı tekniği oldukça esnek ve doğru bir Şekilde kullanılabilir. Birçok organizasyon ve uzman, bir yazılım projesinde iş dağılımı için iş kuralları ile özel kurallara da sahiptir. Örneğin NASA'da iş dağılımı istatistikleri [13], değiştirilmiş ve dönüştürülmüş yazılımlar için aşağıdaki Tablo 2.9'da verilmiştir.

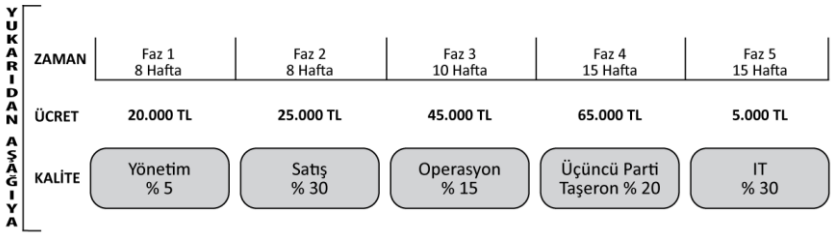
Tablo 2.9 İş Dağılım İstatistiği Örneği

Fazlar	Yeni yazılım zaman tahsisi (%)	Var olan değiştirilmiş yazılım için zaman tahsisi (%)	Değiştirilen Yazılım için zaman tahsisi (%)
Gereksinim Analizi ve Tasarım	20%	15%	5%
Detaylı Analiz, Kodlama ve Birim Test	57%	10%	5%
Yazılım Entegrasyon Testi	23%	40%	30%
Efor	100%	65%	40%

Tablo 2.9'da gösterildiği gibi yeni bir proje için zaman tahsisi gösterilmiştir [13]. Aynı şekilde var olan fakat değiştirilmesi gereken bir yazılım için zaman tahsisi gösterilmiştir.

2.2.3.5 Yazılım Maliyetinde Yukarıdan Aşağıya Yöntemi

Bir proje için yukarıdan aşağıya bir maliyet tahmini, yazılım ürününün global özelliklerini kullanarak tahmin yapmayı gerçekleştirir. Toplam maliyet daha sonra çeşitli bileşenler veya aşamalar arasında bölünür [31]. Yukarıdan aşağıya tahmin aşağıdaki Şekil 2.10'da gösterilmiştir. Yukarıdan aşağı tahmin, diğer tekniklerle birlikte kullanılabilir [31].



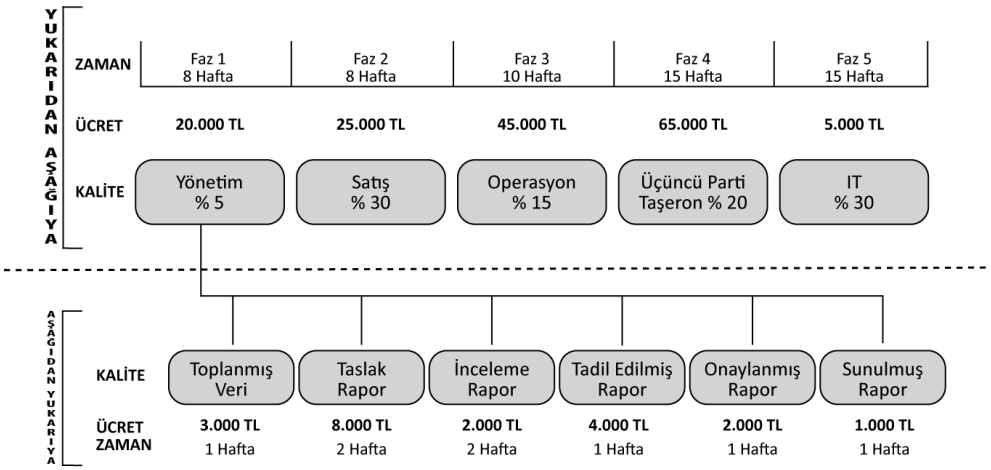
Şekil 2.10 Yukarıdan Aşağıya Tahmin Modeli

Yukarıdan aşağı tahminin ana gücü, sistem seviyesinde odak noktasıdır. Tahmin, tamamlanmış projelerin tamamındaki önceki deneyimlere dayanıyorsa, entegrasyon, kullanım kılavuzları, konfigürasyon yönetimi, vb. gibi sistem seviyesi işlevlerinin maliyetlerini göz önünde bulunduracaktır [3].

Yukarıdan aşağıya tahminlerin başlıca dezavantajları, genellikle maliyetleri artırması muhtemel zor ve düşük seviyeli işlemleri göz ardı etmesidir; bazen geliştirilecek yazılımın bileşenlerini gözden kaçırabilir. İterasyon için ayrıntılı bir temel sağlamaz. Ayrıca bileşenlerde tahmin hatalarının dengelenme şansına sahiptir fakat bu sahiplik kısıtlıdır [32].

2.2.3.6 Aşağıdan Yukarıya Yöntemi

Aşağıdan yukarı tahmin, yukarıdan aşağıya doğru tahmini tamamlayıcıdır. Aşağıdan yukarıya doğru tahminlerde, her bir bileşenin veya iş ürününün maliyeti tek tek tahmin edilir ve bu maliyetler, genel yazılım ürünü için tahmini bir maliyete ulaşacak şekilde toplanır [3]. Aşağıdan yukarıya tahmin edilmeden önce, genel yazılım ürünü ilk olarak daha küçük iş ürünlerine ya da bileşenlerine ayrılmalıdır. Bunu gerçekleştirmek için örneğin iş dökümü yapısını kullanılabilir. Aşağıdaki örnekte bir aşağıdan yukarı tahmin çözümü gösterilmektedir. Aşağıdan yukarıya tahmin yöntemi diğer tekniklerle birlikte de kullanılabilir. Aşağıdan yukarıya tahmin modeline ait çözüm Şekil 2.11'de gösterilmiştir [31].



Şekil 2.11 Aşağıdan Yukarıya Tahmin Modeli

Aşağıdan yukarıya doğru tahmin yöntemi yukarıdan aşağıya yönteminin tamamlayıcısıdır. Aşağıdan yukarıya doğru tahmin, bireysel bileşenlerin geliştirilmesi ile ilgili maliyetlere odaklanma eğilimindedir ve dolayısıyla sistem seviyesinin birçoğunu ve diğer proje maliyetlerini göz ardı edebilir. Aşağıdan yukarıya yöntemi ile tahmin yapmak yukarıdan aşağıya tahmin yapmak yöntemine

göre daha fazla çaba gerektirir. Ancak çeşitli bileşenlerle tahmin hatalarını dengeleyebildiğinden daha istikrarlı tahminler verir [3].

2.2.4 Öğrenme Tabanlı Yazılım Maliyet Tahmini Teknikleri

Öğrenme tabanlı teknikler, bir yazılım maliyeti tahmini geliştirmek için önceki ve güncel bilgileri kullanır. Bu teknikler Yapay Sinir Ağları ve Vaka-Tabanlı Akıl Yürütme yöntemleridir. Bu tez çalışmasında yapay sinir ağları tabanlı bir model geliştirilmiştir. Bu nedenle bu bölümde yapay sinir ağları ile ilgili yüzeysel bilgi sunulmuştur. Yapay sinir ağları ile ilgili ayrıntılı bilgi ve yapılan çalışmalara ait bilgiler yapay sinir ağları ile oluşturulan modelin olduğu bölümde sunulmuştur.

2.2.4.1 Vakaya Dayalı Mantık Kullanarak Yazılım Maliyet Tahmini

Vaka Tabanlı Muhakeme (VTM), analoji ile yazılım maliyet tahminine çok benzer. Analoji aslında basit bir VTM formu olarak tanımlanmıştır (Aamodt ve Plaza, 1994). VTM tahmin tekniği genellikle dört ana adımı içerir:

- En benzer vakaları, yani daha önce geliştirdiğimiz projelerin vaka verilerini toplanır.
- Tahmin problemini çözmek için vakalar tarafından temsil edilen bilgiler tekrar kullanılır.
- Önerilen çözüm gözden geçirilir.
- Gelecekteki problem çözme için faydalı olabilecek bu deneyimin bölümleri vaka veri setine eklenir.

Yazılım maliyeti tahmini bağlamında, bir VTM modeli benzer yazılım projelerinin benzer maliyetlere sahip olduğu şeklindeki benzerlik varsayımına dayanmaktadır. Öncelikle, her bir yazılım projesi, birbiriyle alakalı ve bağımsız olması gereken bir dizi özellik (çevre koşulları, kısıtlar, teknolojiler, karar, başarı, harcanan maliyet, harcanan çaba) ile tanımlanmalıdır. Ardından, aday proje ile tarihsel veri tabanındaki her bir proje arasındaki benzerlik belirlenir. Son olarak, (daha önce geliştirilen benzer) projelerin bilinen geliştirme çaba değerleri yeni vaka uyarlamasını türetmek için kullanılır (örneğin, yeni proje için bir vaka tahmini) [33].

Vaka çalışmalarının kaynağı (veri tabanı), tahmincinin kendi organizasyonu için dahili veya harici olabilir. İç kaynakların tahmin için daha faydalı olması muhtemeldir. Çünkü kendi kurumunun projelerinden faydalanması ve gelecekte uygulanacak belirli mühendislik ve iş uygulamalarını yansıtması ihtimali söz konusudur. Ancak iyi belgelenmiş vakalar benzer çalışmalarda bulunan diğer kuruluşların çalışmalarını inceleyebilir. Bu durumun da ayrıca çok yararlı olduğunu, oluşturulan veriler kullanılarak gösterilebilir [34].

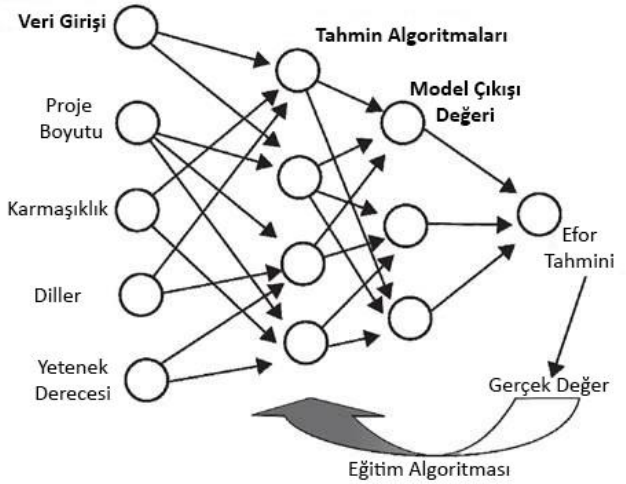
Shepperd ve Schofield [30] tarafından geliştirilen beş aşamalı süreç ve yazılım aracı ANGEL, basit bir Analoji tekniği yerine VTM tekniği olarak da kategorize edilebilir. Idri ve arkadaşları analize dayalı akıl yürütme, bulanık mantık ve dilsel niceleyicilere dayanan bir VTM tekniği oluşturmuş ve bulanık analoji yöntemini kullanmıştır [33]. Bulanık Analoji yaklaşımında hem dilsel hem de sayısal veriler bulanık kümelerle temsil edilir. Ayrıca, iki proje arasındaki benzerliklerin kümelenmesini yönlendirmek için dilsel niceleyici kullanılmıştır. Bulanık Analoji yaklaşımı her ortamın gereksinimlerine ve özelliklerine göre kolayca uyarlanabilir ve yapılandırılabilir [33].

2.2.4.2 Yapay Sinir Ağları ile Yazılım Maliyet Tahmini

Yapay Sinir ağları (YSA), örnekleri öğrenme ilkesine dayanmaktadır. Sinir ağları modelleri dört elementten oluşur. Bunlar nöron, ara bağlantı yapısını oluşturan bağlantıların ağırlıkları (weight), ara katman ve çıktılar olarak karakterize edilir [35]. Burada ara bağlantı yapısını oluşturan bağlantıların ağırlıkları ile ara katmanlar eğitim algoritması olarak da ifade edilebilir.

Sinir ağları kullanılarak geliştirilen yazılım modellerinin çoğunda geri beslemeli besleme ağları kullanılır. Nöral ağlar uygun bir nöron düzeniyle veya ağ düğümleri arasındaki bağlantılarla oluşturulur. Bu, nöron katmanlarının sayısını, her bir katmandaki nöronların sayısını ve bunların hepsinin bağlantılı olduğu yolu tanımlamayı içerir. Düğümler ve düğümler arasındaki bağlantıların ağırlık katsayıları da belirlenir. Ağ gerçek veri ile tahmin edilen veri arasındaki farkı minimum seviyeye ulaşana kadar eğitilir. Böylece ağırlık katsayı değerleri belirlenir. Eğitim verileri geçmiş projelerden oluşmalıdır. Eğitim tamamlandıktan sonra ağ

yayları için uygun ağırlıklar belirlendikten sonra, oluşturulan modelin üreteceği değerleri görmek için ağa yeni girdiler verilir [36].



Şekil 2.12 Yazılım Maliyet Tahmini için Sinirsel Ağlar [30]

Yapay sinir ağları "kara kutu" olarak çalışır ve çıktıların nasıl elde edildiğine dair herhangi bir bilgi veya muhakeme sağlamazlar. Yazılım maliyet verileri iyi bir davranış göstermediği durumlarda, parametreler arasındaki ilişkinin ve bu ilişkilerin oluşturduğu ağırlıkların yeterli olup olmaması modelin başarısını belirleyecektir.

Wittig ve Finnie YSA kullanarak bir yazılım kestirim modeli geliştirmiş ve yüksek tahmin doğruluğu (%75) elde etmiştir [37]. Ayrıca, yazılım geliştirme çabalarını tahmin etmek için kullanıldığında, model %20'lik bir hata oranı vermiştir [38].

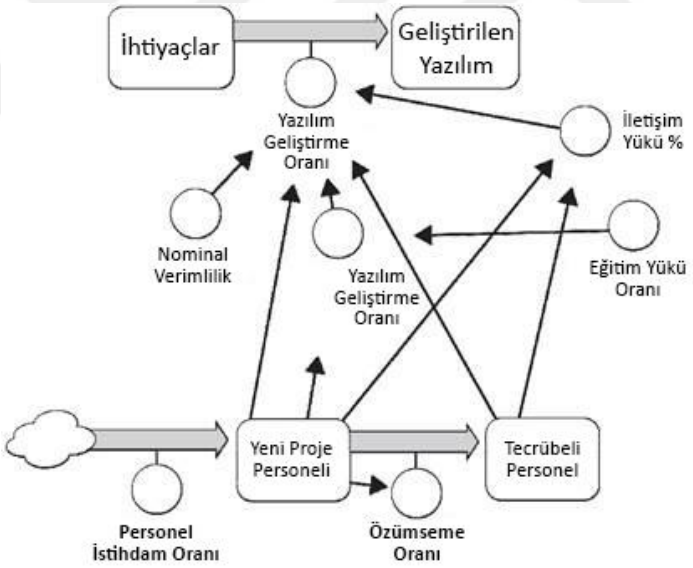
Nöral ağların oluşturulmasında temel istatiksel problemler hala varlığını korumaktadır. Bu problemlerin başında veriler arasındaki uyumsuzluk gelir. Bu nedenle verilerin uyumu oluşturulan ağın başarısını da etkileyecektir. Ayrıca karmaşık durumların çözümünde kullanılacak modellerin oluşturulması için büyük veri setlerine ihtiyaç vardır [36]. Yapay sinir ağları ile ilgili kapsamlı bilgi ve yapılan çalışmalar hakkında bilgi bu çalışmanın uygulama kısmında verilmiştir.

2.2.5 Dinamik Temelli Yazılım Maliyet Tahmini Teknikleri

Dinamik temelli teknikler, yazılım projesi çabalarının veya maliyet faktörlerinin yazılım geliştirme sürecinin süresi boyunca değiştiğini kabul eder [34]. Dolayısıyla, çaba ve maliyet faktörleri zaman açısından durağan olmaktan ziyade dinamik bir

yapıya sahiptir. Bu sadece statik modelleri ve faktörleri kullanan diğer birçok teknikle karşılaştırıldığında önemli bir farktır. Uygulamada, yazılım geliştirme sürecinde işlevsel gereklilikler, proje ekibi, bütçeler, vb. gibi maliyet ve çaba faktörleri değişmekte ve dolayısıyla proje ekibinin verimliliğinde değişikliklere neden olmaktadır. Bu durum projede kalite ve maliyetlerde oldukça olumsuz sonuçlar oluşmasına neden olabilir.

Dinamik tabanlı teknikler genellikle Jay Forrester tarafından ifade edilen sistem dinamiğine dayanır [39]. Dinamik sistemler sistemin karmaşıklığı, sistemin sorunları ve bu sorunları çerçevelemek, anlamak ve tartışmak için bir metodoloji ve bilgisayar simülasyonu sunar. Ayrıca Sistem dinamiği; modeller, pozitif ve negatif geri besleme döngüleri, stoklar, akışlar ve zamanla değişen ve geri besleme döngüleri arasındaki akış hızlarını dinamik olarak etkileyen bilgilerle değiştirilmiş ağlar olarak temsil edilir. Madachy tarafından sunulan aşağıdaki Şekil [40], Brooks Yasasını gösteren bir sistem dinamiği modelinin örneğini göstermektedir [41].



Şekil 2.13 Madachy'in Yazılım Geliştirme için Sistem Dinamikleri Modeli

Sistem dinamiği yaklaşımı aşağıdaki adımlardan oluşur [42]:

- Zaman içinde dinamik olarak problemi tanımlamak
- Oluşturulan sistemin önemli dinamiklerinin içsel, davranışsal bir görünümü tasarlamak

- Tüm gerçek sistem kavramlarını geri besleme döngülerinde ve dairesel nedensellikte birbirine bağlı sürekli nicelikler olarak düşünmek
- Sistemdeki bağımsız seviyelerin ve bunların giriş ve çıkış değerlerinin belirlemek
- Oluşturulan modeli yeni özellikler ile geliştirmek ve beslemek
- Elde edilen modelden anlayışları ve uygulanabilir politika anlayışlarını türetmek
- Model tabanlı anlayışlardan kaynaklanan değişikliklerin uygulanmak

Sistem dinamiği simülasyon modelleri matematiksel olarak bir dizi diferansiyel denklemler ile aşağıdaki sırayla temsil edilir [40].

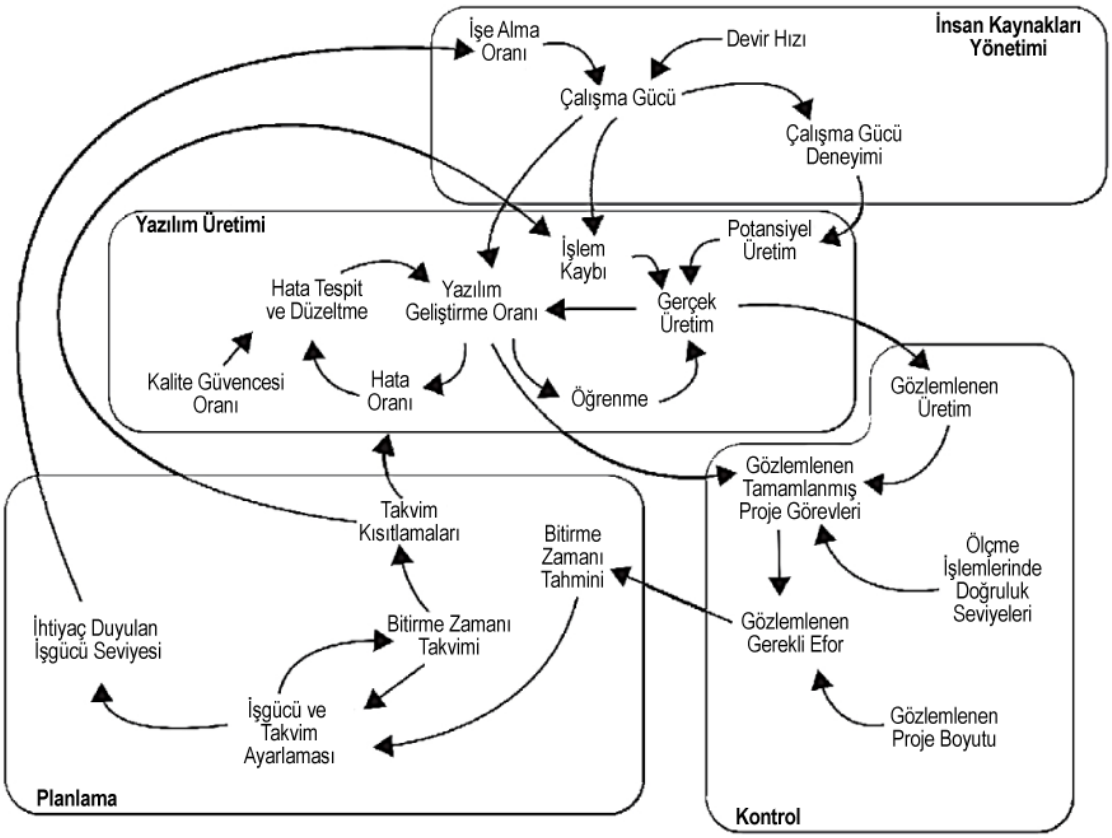
$$x'(t) = f(x, p) \tag{2.8}$$

x = modeldeki seviyeleri (durumları) açıklayan bir vektör

p = bir dizi model parametresi, doğrusal olmayan bir vektör fonksiyonu

t = zaman Sistem dinamiği tekniği,

Abdel-Hamid ve Madnick tarafından oldukça başarılı bir şekilde yazılım mühendisliği tahmininde kullanılmıştır [43]. Proje geliştirmenin ilk uygun değerlerinin tahmin ediciye açık olması koşuluyla, proje maliyetindeki değişiklikleri, personel ihtiyaçlarını ve zaman içindeki değişiklikleri tahmin edecek modeller oluşturmuşlardır. Buna ek olarak, yazılım bileşenlerinin yeniden kullanımı ile proje personelinin üretkenliği arasında başlangıçta yararlı bir ilişki bulmuşlardır. Böylece yeni bir kod geliştirmek için daha az çaba harcanmaktadır [44]. Şekil 2.14 Yazılım Maliyet Tahmini için Sistem Dinamikleri Modeli ayrıntılı bir şekilde gösterilmiştir [43].



Şekil 2.14 Yazılım Maliyet Tahmini için Sistem Dinamikleri Modeli

2.2.6 Regresyon Temelli Yazılım Maliyet Teknikleri

Regresyon tabanlı teknikler, bir ana maliyet sürücüsü (Kod satırı sayısı olabilir) olarak düşünülen değişkenlerin bir fonksiyonu olarak bir yazılım maliyet tahmini üreten bir veya daha fazla matematiksel algoritma sağlar. Regresyon teknikleri genellikle sıradan en küçük kareler yöntemi ve artık kareler yöntemini kullanır. Bu tez çalışmasında regresyon tabanlı bir maliyet modeli oluşturulmuştur. Altı farklı yöntem kullanılmıştır. Bu kullanılan regresyon yöntemleri ile ilgili ayrıntılı bilgi ve bu konuda yapılan çalışmalar ile ilgili ayrıntılı bilgi regresyon tabanlı geliştirilen yazılım maliyet tahmin modelinin anlatıldığı bölümde ayrıntılı bir şekilde açıklanmıştır. Bu bölümde yüzeysel bilgi sunulmuştur.

2.2.6.1 Sıradan En Küçük Kareler Tekniği

Sıradan en küçük kareler (OLS-Ordinary Least Square) tekniği, doğrusal bir regresyon modeli kullanarak bilinmeyen parametreleri tahmin etmek için kullanılan klasik bir matematiksel yöntemdir. İleri istatistik ve ekonometri

çalışmalarının çoğunda regresyon çözümleri sunulmuştur [45]. İstatistiksel yazılım paketlerinin basitliği ve kolay erişilebilir olmasından dolayı araştırma literatüründe kullanılan yaygın tekniklerden biridir [36]. Sıradan En Küçük Kareler yöntemi, karesel artıkların toplamını en aza indirerek veri setine uyan en iyi doğrusal modeli bulmayı amaçlamaktadır. Artıklar, veri kümesinde gözlenen ile doğrusal yaklaşımın öngördüğü yanıtlar arasındaki farktır.

Basit bir veri seti n noktalarından (veri çiftleri) oluşur, (xi, yi), i = 1, ..., n, burada xi bağımsız bir değişkendir ve yi gözlemler tarafından değeri bulunan bağımlı bir değişkendir (geçmiş Proje maliyet çıktı verileri). Model işlevi, ayarlanabilir parametrelerin vektörde tutulduğu ve f (x,) biçimine sahiptir. Amaç, modele "en iyi" uyan verilere ait parametre değerlerini bulmaktır. En küçük kareler metodu, kare artanlarının toplamı, S, en düşük değerindeyken en optimum durumdadır:

$$\text{Min } S = \sum_{I=1} r_i \qquad (2.9)$$

Kalıntılar r_i, modelin tahmin ettiği değer ile gerçek değer arasındaki fark olarak tanımlanır.

Bağımlı değişkenin gerçek değeri:

$$r_i = y_i - f(x_i, \beta) \qquad (2.10)$$

İle hesaplanır.

Kesişimi 0 olarak ve eğimi 1 olarak göstererek, doğrusal model fonksiyonu şu şekilde verilir:

$$f(x, \beta) = \beta_0 + \beta_1 x \qquad (2.11)$$

Artan kareler toplamını, en aza indirgeyen ayarlanabilir parametrelerin (optimum) değerleri basit birinci mertebeden türetme yoluyla bulunabilir. Optimum değer bulunduğunda, model fonksiyonu f (x, optimum), tahmin için kullanılabilir.

Normal En Küçük Kareler yönteminden başka varyasyonlar da vardır. Bunlar Ağırlıklı En Küçük Kareler ve Doğrusal Olmayan En Küçük Kareler yöntemlerini içerir. Ağırlıklı En Küçük Kareler metodu, karesel artıkların ağırlıklı toplamını en aza

indirmeyi amaçlamaktadır. Bu nedenle maliyet, modeldeki ayarlanabilir parametreler ile temsil edilen farklı maliyet faktörleri ile ifade edilir. Doğrusal Olmayan En Küçük Kareler yönteminde veriler, doğrusal olmayan bir model fonksiyonuna f (x,) yerleştirilmiştir.

En küçük kareler regresyonu verilerin sürekli bir fonksiyon ile ifade edilebildiği durumlar için uygundur [36]. Birçok serbestlik derecesi mevcuttur, yani tahmin edilecek parametrelerden daha fazla gözlem vardır. Veriler birbiriyle uyumludur (istatistiksel anlamda, örneğin, aykırı değerler veya önemli bir heterosik-büyüklük yoktur).

Bunlar, bu tekniğin tüm şartlarını nadiren karşılayan yazılım mühendisliği veri setleri için kullanılmasında ciddi bir kısıtlama getirmektedir. Yazılım mühendisliğinde, daha yüksek yönetim tarafından finanse edilmemesi, çeşitli geliştirme süreçlerinin bir arada bulunması ve sürecin doğru yorumlanamaması nedeniyle veri toplamak genellikle zordur. Ayrıca, çoğu zaman geçmiş projelerden eksik veya yanlış bir şekilde veri toplanabilmektedir [34]. Bu nedenle bu yönteminin kullanılacağı verilerin tutarlı olması önemlidir.

2.2.7 Maliyet Tahmini, Kıyaslama ve Risk Analizi (Nedensel Model)

Maliyet Tahmini, Kıyaslama ve Risk Analizi yöntemi (CoBRA), 1996 yılında Fraunhofer IESE (Institut für Experimentelles, Yazılım Mühendisliği) tarafından geliştirilmiştir. Ayrıca çeşitli çalışmalarda kullanılmıştır [46], [47], [48]. Sadece az miktarda verinin mevcut olduğu kuruluşlarda kullanılabilecek bu teknik, uzman güdümlü ve veri güdümlü maliyet tahmin yöntemlerinin birleştirilmesi ile oluşturulmuştur.

CoBRA'da kullanılan temel formül şu şekildedir:

$$Maliyet = Nominal\ üretkenlik\ x\ Boyut + Maliyet\ aşımı \quad (2.12)$$

Nominal üretkenlik, verimliliği optimal bir durumda belirler (tüm maliyet faktörlerinin optimal değerlerde olduğu bir projede). Maliyet yükü, optimal bir ortamda yapılmayan projelerde meydana gelen ek maliyetleri ifade eder. Ayrıca etkileme faktörleri kümesiyle (örneğin, takım kabiliyeti, gereksinim kararlılığı, vb.)

genel giderler ortaya çıkar. Temel faktörler kullanılarak proje maliyetleri üzerindeki etkilerini belirleyen bir model ortaya konulur.

Nedensel modeldeki maliyet faktörleri iki yolla olumlu veya olumsuz bir etkiye sahip olabilir. Doğrudan ilişkiler, maliyetler üzerinde doğrudan etkiye sahiptir. Maliyet faktörlerinin artması durumunda maliyet yükünün doğrudan artıracağı anlamına gelir. Dolaylı ilişkiler başka bir faktörün maliyet yüküne olan etkisini belirler. Bazı faktörler, maliyet yükü üzerinde doğrudan ve dolaylı bir etkiye sahip olabilir.

CoBRA'da, nedensel modelin geliştirilmesi ve nicelleştirilmesi uzmanlar tarafından yapılır. Faktörlerin seçimi ve nedensel ilişkilerinin belirlenmesi grup tartışmalarında yapılır. Nominal verimlilik, geçmiş projelerden elde edilen veriler kullanılarak hesaplanır.

CoBRA süreci şu şekilde özetlenebilir:

- Olası maliyet faktörlerini toplayın: Maliyet yükünü etkileyen olası faktörler kümesi toplanır.
- Maliyet faktörlerini sıralayın ve seçin: Her uzman, maliyet faktörlerini maliyet yüküne olan etkisine göre ayrı ayrı sıralar. Sıralamanın toplu sonuçları uzmanlara sunulur ve grup tartışmasında en önemli maliyet faktörleri seçilir.
- Nedensel model oluşturma: Uzmanlar arasında yapılan diğer bir grup tartışmasında, maliyet faktörleri üzerindeki faktörlerin doğrudan ve dolaylı etkileri belirlenir ve nedensel bir model geliştirilir.
- Nedensel ilişkilerin nicelleştirilmesi: Nedensel modele dayanarak, her bir uzmandan faktörlerin maliyet yükü üzerindeki etkisini ölçmek için bir anket hazırlanır. Her kararda, her bir uzmandan nominal bir olayda asgari, büyük olasılıkla azami ek yükü yüzde olarak belirtmesi istenir.
- Geçmiş projeleri analiz edin: geçmişte bir dizi proje için, boyut ve efor verileri toplanır ve uzmanlardan, masraf üst yükü sürücülerine göre onları karakterize etmeleri istenir. Bu sonuçlara dayanarak, nominal verimlilik belirlenir.

2.2.8 Makine Öğrenmesi Tabanlı Teknikler

Makine öğrenmesi algoritmaları son yıllarda yazılım maliyet tahmininde kullanılmaya başlandı. Makine öğrenmesi çözümlerinde en önemli basamak veri setleridir. Yeterli veri setlerinin olmaması bu alanda yapılan çalışmaları sınırlandırmıştır. Bu bağlamda yapılan çalışmalara baktığımızda genellikle özel olarak oluşturulan veri setleri kullanılmaktadır. Bunun yanı sıra Son yıllarda firmalarda kullanılmaya başlanan JIRA, CleverControl gibi iş takip programlarının yüksek oranda kullanılmaya başlanmasıyla bu alanda farklı ve özelleştirilmiş veri setleri elde edildi. Bu da makine öğrenme algoritmalarının daha fazla kullanılabileceği alanlar oluşturdu.

Bu tez çalışmasında geliştirilen üçüncü yöntem makine öğrenme algoritmaları tabanlıdır. Bu nedenle önerilen modelin anlatıldığı bölümde, makine öğrenme algoritmaları hakkında detaylı bilgi verilmiştir. Ayrıca makine öğrenmesi algoritmalarını kullanarak maliyet tahmini yapmaya çalışan modeller hakkında detaylı bilgi önediğimiz modelin anlatıldığı bölümde açıklanmıştır.

Bu bölümde yazılım geliştirme sürecinde maliyet tahmini yapmak için kullanılan yöntemler kategorize edilerek ayrıntılı bir şekilde ele alındı. Bu yöntemleri ortaya koyan çalışmalar hakkında bilgi verildi. Bu çalışmada özellikle regresyon, yapay sinir ağları ve makine öğrenmesi yöntemlerini kullanıldı. Bu çalışmada uygulanacak algoritmalar ve yöntemler ile ilgili detaylı bilgi önerilen yazılım maliyet kestirim modellerin anlatıldığı bölümde verilecektir.

3
Scrum

Bu tez çalışmasında geliştirildiğimiz üçüncü yazılım maliyet tahmini modeli yazılımın geliştirildiği metodolojiyi dikkate alarak maliyet tahmini yapmaktadır. Geliştirdiğimiz bu model ile Scrum metodolojisi ile geliştirilen yazılımlar için maliyet kestirimi yapılmaya çalışılmaktadır. Bu nedenle bu bölümde Scrum metodolojisi hakkında detaylı bilgi sunulmaya çalışıldı.

Scrum, yinelemeli ve Çevik yazılım geliştirme metodolojisinin bir türüdür [49]. Scrum geliştirme sürecinde müşterilerin gereksinimlerindeki hızlı değişimlere hızlı bir şekilde cevap vermeyi amaçlar. Bu yaklaşım "ampirik yaklaşım" olarak tanımlanmaktadır [49]. Sorunun tam olarak anlaşılamayacağını veya tanımlanamayacağını kabul eder. Takımın yeteneğini maksimize etmeye odaklanır. Müşteriye ürünü hızlı bir şekilde teslim etmeyi ve ortaya çıkan yeni gereksinimlere ve değişikliklere hızlı cevap vermeyi amaçlar [49].

Scrum hızlı ve basit bir işlemdir. Bu da işlemin yararlı bir iş elde etmek için mümkün olan en verimli sürenin maksimuma çıkarılması ve mümkün olduğu kadar işlerin küçük iş parçalarına ayrılması anlamına gelir. Scrum anlaşılması kolay ve ustalaşması zordur [50].

3.1 Geçmişi

Jeff Sutherland ve Ken Schwaber, 1995 yılında ABD'de Austin'deki Oopsla konferansında Scrum sürecini duyurdu ve "SCRUM Yazılım Geliştirme Süreci" adlı makalesini yayınladı [51]. Takeuchi ve Nonaka adlı yazarların "Yeni Ürün Geliştirme Oyunu" adlı makalesinden "Scrum" terimini kullanılmaya başlandı. Araştırma, yeni ve karmaşık ürünlerin geliştirilmesinde kendi kendini organize eden ve amaçlarını kendileri tarafından belirlenen küçük ekiplerin olağanüstü başarı ortaya koyduğunu gösterdi. En iyi takımların, kendi ortak hedeflerine ulaşmak için sorunlar ile en iyi nasıl baş edebileceklerine dair kendi taktiklerini tasarlayanların olduğu

gözlemlendi. 2001 yılının şubat ayında Jeff ve Ken, Çevik Yazılım Geliştirme için Manifestoyu oluşturan 17 yazılım geliştirme liderinden biriydi [49]. 2002'de Ken Schwaber, son derece başarılı sertifikalı (Certified) Scrum Uzmanlarının (Scrum Master) ortaya çıkması için Mike Cohn ve Esther Derby ile Scrum Alliance'ı kurdu. 2006 yılında, Jeff Sutherland kendi Scrum Şirketini kurmuş ve sertifikalı Scrum kurslarını sunmaya ve öğretmeye devam etmiştir. Ken, 2009 sonbaharında Scrum İttifak'tan ayrıldı ve Scrum'un kalitesini ve etkinliğini daha da geliştirmek için Scrum.org'u kurdu [52].

2010 yılında Scrum rehberinin ilk yayını ,2011 ve 2013 yıllarındaki artan güncellemeleri ile Jeff ve Ken, dünya çapında tanınan Scrum belgesini oluşturdu [53].

1995 yılında ilk yayınından bu yana, Scrum dünya çapında birçok yazılım geliştirme şirketi tarafından benimsenmiştir. Çevik yazılım geliştirme için en çok uygulanan çerçeve (framework) olarak kabul edilmektedir.

Günümüzde Scrum sadece yazılım geliştirme için değil, aynı zamanda imalat, pazarlama, operasyon ve eğitim alanlarında da kullanılmaktadır.

3.2 Hızlı Bildiri

11-13 Mayıs 2001'de, on yedi kişi ağır yazılım geliştirmeye alternatif ve ihtiyaca cevap verecek bir model ortaya koymak için toplandı. Bu toplantı sonucunda Aşırı Programlama (Extreme Programming), SCRUM, DSDM (Adaptive Software Development), Kristal (Crystal), Özellik Tabanlı Geliştirme (Feature-Driven Development) ve Pragmatik modellerden oluşan Çevik model duyuruldu. Çevik Yazılım'ın (Agile Manifesto) on iki ilkesi vardır [6]:

- En büyük önceliğimiz, değerli yazılımların hızlı ve eksiksiz teslimi ile müşteriyi memnun etmektir.
- Değişen ihtiyaçlar karşısında, Çevik süreçler müşteriye rekabet için avantaj sağlar.
- Çalışan yazılımı hızlı bir şekilde kısa zaman aralıklarında sunmak.
- Müşteriler ve geliştiriciler proje boyunca günlük olarak birlikte çalışmalıdır.

- Motive bireyler etrafında projeler oluşturun. Onlara ihtiyaç duydukları ortamı ve desteği verin ve işi yapmak için onlara güvenin.
- Bir gelişim ekibine bilgi aktarmanın en etkili yöntemi yüz yüze iletişimdir.
- Çalışan yazılım, ilerlemenin birincil ölçütüdür.
- Çevik süreçler sürdürülebilir kalkınmayı teşvik eder. Sponsorlar, geliştiriciler ve kullanıcılar süresiz olarak sabit bir hızda devam edebilmelidir.
- Teknik mükemmellik, iyi tasarım ve sürekli dikkat çevikliği artırır.
- Basitlik- yapılan iş miktarını maksimize etme sanatıdır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize eden ekiplerden ortaya çıkar.
- Ekip nasıl daha etkili hale geleceğini ve ardından da davranışını buna göre ayarlamak için sık sık toparlanır.

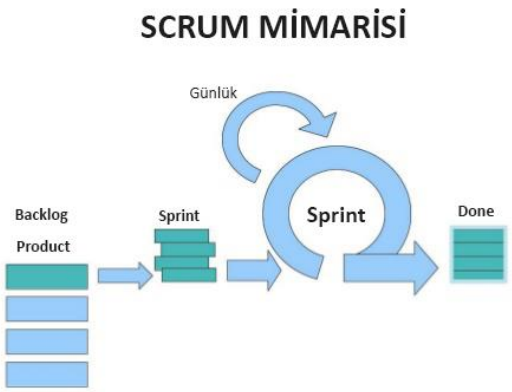
3.3 Scrum Teorisi

Scrum, bilginin deneyimden geldiğini ve bilinenlere dayanarak kararlar alındığını iddia eden deneysel bir süreçtir [51]. Scrum'ın her uygulamasında üç önemli faktör desteklenmektedir: şeffaflık, denetim ve uyum.

Şeffaflık: Yazılım geliştirme sürecinin önemli yönleri, sonuç için tüm paydaşlar tarafından görülebilir olmalıdır. Şeffaflık, sürecin tüm katılımcılar tarafından anlaşılması ve hangi aşamada olduğunun anlaşılması için gereklidir ve ortak bir dil gerektirir.

İnceleme: Scrum kullanıcıları, istenmeyen değişiklikleri tespit etmek ve Scrum da ortaya konan Hedeflere ulaşılıp ulaşılmadığını anlamak için Scrum eserlerini sık sık incelemeli ve ilerlemelidir.

Uyarlama: Bir sürecin bir veya daha fazla yönünün saptığını tespit edilirse; Daha fazla sapmayı en aza indirmek için işlemler (önlemler) ayarlanmalıdır.



Şekil 3.1 Scrum Metodolojisinin Genel Çalışma Şekli

Şekil 3.1'de Scrum metodolojisin genel mimarisi gösterilmiştir. Şekilden de anlaşılacağı üzere yapılacak olan işler Scrum iş listesine (Product Backlog) atılır. Daha sonra ürün listesindeki işler, iş listesinden ilgili sprint'e atılır. İlgili sprintin günlük üzerinde çalışılan işleri ile ilgili olarak günlük toplantılar (Daily Scrum) yapılır. Sprint sonunda ise işler Tamamlandıya (Done) çekilir [54].

3.4 Roller

Scrum takımında üç ana rol vardır. Bu roller Ürün Sahibi (Product Owner -PO), Scrum Uzmanı (Scrum Master -SM) ve Geliştirme Ekibi (Development Team, DT) rolleridir. Scrum Takımları kendi kendini organize eder ve çapraz işlevlidir. Kendi kendini organize eden ekipler, çalışmalarını en iyi şekilde nasıl gerçekleştireceklerini belirler. İşlevsel takımlar, takımın bir parçası olmayan diğer kişiler ile çalışmalarında başarılı olmak için gereken tüm yetkinliklere sahiptir [49].

Scrum ekipleri, ürünleri artırımlı ve kademeli olarak teslim eder. Bu artan teslimatlar, çalışan ürünün yararlı sürümleri olmalıdır.

3.4.1 Ürün Sahibi (Product Owner -PO)

Ürünün değerini ve geliştirme ekibinin çalışmasını maksimize etmekten PO sorumludur [49]. PO bir kişidir, komite değildir. PO paydaşları temsil eder ve müşterinin sesidir.

"Product Owner, Geliştirme Takımı tarafından geliştirilen işin ve ürünlerin değerini maksimuma çıkarmaktan sorumludur." – Scrum Guide

- Ürünün özelliklerini tanımlar.
- İçerik ve sürüm tarihine karar verir.
- Ürünün karlılığı için sorumludur.
- Market değerlerine göre özellikleri önceliklendirir.
- Her iterasyon için önceliklendirme ve özellikleri ayarlamadan sorumludur.
- İş sonuçlarını kabul veya reddeder.

Organizasyonda başarılı olabilmek için, tüm organizasyon kararlarına karşı saygılı olmalıdır. Geliştirme ekibine ne yapması gerektiğini kimse söyleyemez ama geliştirme ekibi de kendi başına harekete geçemez.

3.4.2 Geliştirme Ekibi

Her sprintin sonunda "Tamalandıya", ürünün salınabilir bir artışını ekleme sorumluluğu olan bir grup profesyoneldir. Karar verilirse, üründe yapılan artış serbest bırakılabilir(release). Geliştirme ekibi kendi kendini organize eder ve çapraz işlevlidir. Farklı takımlardan yardım almadan kendi başlarına ürün artışları oluşturabilirler [55].

Geliştirme ekibi üyelerinin, geliştirici dışında, analist veya tester gibi bir unvanları yoktur. Hepsi geliştiricidir. Ekip üyeleri bazı uzmanlık ve yeterlilik alanlarına sahip olabilir, ancak hesap verebilirlik her zaman geliştirme ekibine aittir [55].Takım 3 ila 9 üyeden oluşabilir. Ekip, çevik ve önemli çalışmaları tamamlamak için yeterince büyük kalabilecek kadar küçük olmalıdır. Daha küçük ekipler ürün artışı kazanmakta zorluk çekebilirken, daha büyük takımlar koordinasyonda zorluk yaşayabilirler. Sadece sprintler arasında üyeler değişebilir. Üyeler tam zamanlı olmalı. (İstisnai durumlar olabilir örneğin Veri tabanı Yöneticisi)

3.4.3 Scrum Uzmanı (Scrum Master)

Scrum takımındaki Scrum Uzman'ı, Scrum teorisini ve bu teorinin en iyi uygulamalarını ve kurallarının önemini vurgulayarak, Scrum'ın anlaşılmasının ve iyi bir şekilde uygulanmasının sağlanması sorumluluğu taşıyan kişidir [56].

Scrum Uzmanı, Scrum Ekibinde bir proje yöneticisi değil ve geleneksel bir takım lideri değildir. Hizmetkâr bir lider veya ekibin işinin kolaylaştırıcısıdır. Geliştirme ekibinin arttırdığı ürünün değerini artırmaya yardımcı olan kişidir.

SM'nin ayrıca Scrum Takımının PO'suna karşı farklı sorumlulukları vardır.

SM'nin PO'ya karşı sorumlulukları [49]:

- Etkili PBL (Project Based Learning) Yönetimi için teknikler bulur. PBL projeye özgü ortaya konacak yönetim modelidir.
- Scrum ekibinin İş listesindeki iş parçalarının (PBI- Product Backlog Item) ihtiyacını anlamasına yardımcı olur.
- Çevikliği anlamak ve pratik yapmak.
- Gerekli ve talep edilen Scrum Etkinliğini düzenler.

SM'nin Geliştirme Ekibine karşı sorumlulukları:

- Ekibin üretkenliğini, işlevselliğini ve motivasyonunu artırmaya çalışır.
- Geliştirme Ekibinin kendi kendini organize etmesi ve çapraz işlevsel olması için koçluk yapar.
- Problemleri ve özellikle de ekibi engelleyen her şeyi ortadan kaldırmaya çalışır. Böylece Geliştirme Ekibinin ürün devamlılığını ve sürekliliğini sağlar.

SM'nin Organizasyona karşı sorumlulukları:

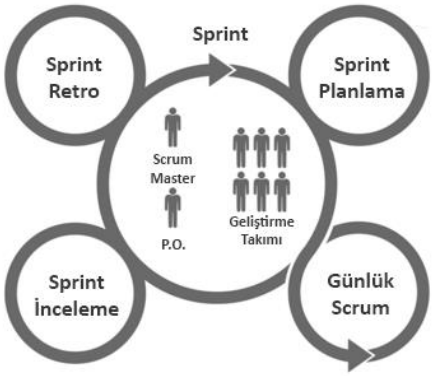
- Scrum'ın organizasyon içinde benimsemesinde birinci derecede rol alır.
- İşçilerin ve paydaşların Scrum'ı anlamalarına ve Scrum'ı ampirik bir şekilde benimsemelerine yardımcı olur.
- Organizasyondaki Scrum'ın etkinliğini artırmak için diğer Scrum uzmanları ile iş birliği yapar.

Scrum uzmanını iyi yapan altı özellik var. Bunlar; sorumlu, mütevazı, işbirlikçi, kararlı, etkili ve bilgili olmalarıdır [49].

3.5 Olaylar

Tüm Scrum Etkinlikleri, her olayın düzenlilik oluşturmak ve tanımlanmamış toplantılara olan ihtiyacı en aza indirgemek için maksimum süreye sahip olduğu anlamına gelen, zamana bağlı olaylardır. Örneğin, Günlük Scrum 15 dakika ile

sınırlıdır [54]. Sprint başladıktan sonra, süresini kısaltmak veya uzatmak için bir yol yoktur. Amaçları yerine getirilirse, diğer olaylar bitebilir.



Şekil 3.2 Scrum Metodolojisinin Olayları

Scrum'daki her etkinlik bir fırsattır ve üretkenliği arttırmak için uyarlar ve denetler. Herhangi bir olay başarısız olursa, şeffaflık azalır ve denetim için fırsat kaybedilir. Şekil 3.2'de Scrum metodolojisinin olayları gösterilmiştir.

3.5.1 Koşum (Sprint)

Sprint; sprint planlama, günlük Scrum, geliştirme çalışmaları, sprint gözden geçirme (Sprint Review) ve Sprint Retrospektifi (Sprint retrospective) gibi etkinliklerden oluşan ana etkinliktir [49]. Her sprint bir ay veya en sık iki hafta süren zaman kutulu bir olaydır. Bir ay ile sınırlandırılmalıdır çünkü zaman dilimi genişlerse karmaşıklık ve risk artabilir.

Sprint başlamadan önce bir sprint zamanı seçilmelidir. Bir sprint başladıktan sonra değiştirilemez.

Her sprintin bir hedefi vardır ve tehlike altında olmamalıdır. Sprint'in sonunda kullanılabilir ve potansiyel olarak değiştirilebilir ürün artışı oluşturulmalıdır.

Sprintler sadece PO tarafından çıkarılabilir, ancak paydaşlar, geliştirme ekibi veya Scrum uzmanı tarafından etkilenebilir. Sprint, hedefinin geçerliliğini yitirirse o sprint iptal edilir. Sprint iptalleri çok nadirdir ve çoğu zaman Scrum takımı için

travmatiktir, çünkü herkes başka bir sprint planlamada yeniden gruplanmak zorundadır [49].

3.5.2 Sprint Planlama

Sprintler detaylı bir şekilde planlanmalıdır. Bu plan Sprint iş listesi (Sprint Backlog) ve sprint hedefi oluşturmak için Scrum takımı tarafından oluşturulur. Şekil 3.3'de Scrum Metodolojisinde Sprint Planlama ait klasik bir toplantı görseli konmuştur.

Sprint planlama ayrıca bir aylık planlama için sekiz saat ile sınırlıdır. Scrum Uzmanı, etkinliğin gerçekleşmesini ve katılımcıların amacını anlamasını ve zaman Tablosinde hedefin tutulmasını sağlar [48].

Sprint iş listesi, Sprint hedefini tamamlamak ve başarmak için geliştirme ekibi tarafından seçilen PBI'lardan oluşur. PBI, Sprint sırasında yapılacak işin açık ifadesidir.

Şekil 3.3 Scrum Metodolojisinde Sprint Planlama

Sprint planlama, "üründe Artışı sağlamak için ne kadar iş gerekiyor?" sorusuna cevap vermelidir. Bundan sonra sprintte gerçekleştirecek olan iş için ilgili öğeler seçilir.

Öğeler PBI'dan seçildiğinde, geliştirme ekibi öğeyi "tamamlandı" olarak tamamlamak için bir kriter belirler.

Takım iş listesinden tamamlayabilecekleri işleri seçer. Sprint iş listesi oluşturulur. İşler tanımlanır ve her biri değerlendirilir. (1-16 saat). Bu durum yüksek seviye tasarım olarak kabul edilir.

3.5.3 Günlük Scrum (Daily Scrum)

Günlük Scrum, Geliştirme Takımı'nın bir sprint sırasında gelecek 24 saat için planlaması için 15 dakikalık bir aktivitedir [49]. Karmaşıklığı azaltmak için her gün aynı yerde gerçekleşir. Etkinlik sırasında, her Geliştirme Ekibi üyesi şunları açıklar.

Şekil 3.4 Scrum Metodolojisinin Günlük Toplantı (Daily Scrum)

- Dün üzerinde çalıştığım iş ile ilgili olarak ne yaptım?
- Bugün ne yapacağım?
- Geliştirme Ekibi verimliliğini düşüren engellerim var mı?

Günlük Scrum sırasında herhangi bir engel (blok, risk veya sorun) ortaya çıkar ve Scrum uzmanı tarafından belirtilir ve ekibin Scrum tahtasında gösterilir. Şekil 3.4'te günlük Scrum bir görselle ifade edilmiştir.

Scrum Uzmanı toplantının düzenleyicisidir. Geliştirme Ekibinin toplantıya katılmasını sağlar ve toplantının süresini 15 dakikaya kadar sınırlar.

3.5.4 Sprint İnceleme

Bir aylık sprint için, kısa sprintler için dört saatlik bir toplantıdır, olay genellikle daha kısadır. Bir sprint'in sonunda, Scrum ekibi ve paydaşlar sprintte neler yapıldığını görebilirler. Artış PO tarafından sunulur. Neyin "yapıldığını" ve neyin

“yapılmadığını” burada ortaya koyar. PO geribildirim ve iyileştirme fikirlerini ifade eder. Geliştirme ekibi artırımın sorularına cevap verir.

Sprint incelemesinin sonunda, ürün istek listesi PO tarafından revize edilir ve katılımcılar bundan sonra ne yapacakları konusunda iş birliği yaparlar. Bir sonraki Sprint planlamanın maddelerini yüzeysel olarak ortaya çıkarmayı kolaylaştırır [49].

3.5.5 Sprint Retrospektifi

Sprint retrospektifi, sprintin sonunda gerçekleşen bir toplantıdır. Aynı zamanda ayda bir sprint için gerçekleştirilen 3 saatlik bir olaydır. Kısa sprintler için genellikle daha kısadır. Scrum Uzmanı, etkinliği kolaylaştırır ve Scrum sürecinin hesap verebilirliğini arttırmak adına geliştirme ekibi de katılır [49].

Sprint retrospektifi, denetim ve adaptasyon için resmi bir fırsat sunar. Geliştirme ekibine daha da üretken hale gelmeleri için fırsat sunar. Scrum uzmanı ayrıca toplantıyı organize eder ve sprintin amacını sağlar.

Bu etkinliğin amacı:

- Son sprint üzerinde yapılanları yansıtmak
- Sürekli süreç iyileştirme eylemlerini tanımlamak ve kabul etmek.

Bu toplantının amacına ulaşması için sprint retrospektifleri için bazı kurallar vardır. Her takım üyesi, son sprintteki her şey hakkındaki fikirlerini şu yönergelere göre ifade eder:

- Tekerlek / Denizyıldızı
- Yelkenli
- Çılgın Memnuniyetlik
- Serin Duvar
- Yalın Kahve
- Sorular Retrospektif
- 4LS
- Memnuniyet Hektogramları
- Çemberler
- Çeyrek (Quartering)

- Retrospektif Eylemleri Yönetme

Bunlar arasında, Deniz Yıldızı ve Çılgın memnuniyetlik teknikleri en yaygın kullanılanlardır. Denizyıldızı modeli Scrum tahtasını beş kategoriye ayırır:

Yapmaya Devam Et: Ekip üyeleri için Sprint sırasında sevdikleri iyi şeylere odaklanmak iyi bir başlangıç noktasıdır. Deniz yıldızı tekniği aşağıda ifade edilen elemanlardan oluşmaktadır.

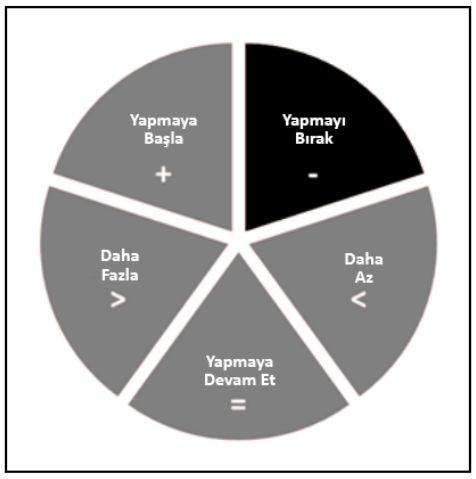
Az: Her takım üyesi daha çevik olmak için kullanılmaması veya yapılmaması gerektiği fikrine sahiptir.

Daha Fazlası: Bu daha fazla olursa, daha fazla üretkenlik sağlar.

Yapmayı Durdurun: Ekip üyeleri bunu yapmaktan vazgeçmeye karar verdiler çünkü yararlı değil ya da takımın başarısı için bir engel.

Yapmaya Başlama: Bu olursa, daha verimli, Çevik ve etkili bir geliştirme ortamı elde edilebilir.

İkincisi, çılgın memnuniyet modelidir. Burada Scrum panosu 3 alana ayrılmıştır:



Şekil 3.5 Retrospektif – Deniz Yıldızı Tekniği

Deli- Mad hayal kırıklığı, takımı rahatsız eden ve / veya çok zaman harcayan şeyler

Üzücü- Sad hayal kırıklıkları, işin beklenilen şekilde çalışmaması

Sevindim-Glad zevkler, takımı mutlu eden şeyler

Şekil 3.5'te deniz yıldızı yönergesi ayrıntılı bir şekilde gösterilmiştir.

3.6 Uzantılar

Uzantılar Scrum'daki temel uygulamalar olmasa da bir organizasyonun daha verimli ve etkili bir ürün geliştirmeye sahip olması için yararlı olan yazılım geliştirme uygulamalarıdır. 2012 yılında Scrum.org tarafından uzantılar kısmı askıya alındı [52]. Fakat daha sonra tekrar web sitesinde yayınlandı.

3.6.1 İş Listesi İyileştirme (Backlog Refinement)

İş listesi iyileştirme, PBI'ların devam eden sürecini gözden geçirme ve kontrol etme süreci olup, uygun bir şekilde önceliklendirilmeleri ve geliştirme ekibi için açık ve yürütülebilir olmalarının sağlanmasıdır.

İş listesi artırımı çekirdek bir Scrum uygulaması değildir. Ancak sprint kapasitesinin %10'una kadar önerilen bir süre ile bir sprinte giren iş listesi maddelerinin kalitesinin korunması için tercih edilmiştir [52].

3.6.2 Scrum'ın Scrum'ı (Scrum of Scrums- SoS)

Bir SoS Toplantısı, aynı ürün üzerinde çalışan birden fazla takım için Scrum'ı ölçeklendiren bir tekniktir. Bu toplantıda, Scrum ekipleri, projeyi ortaklaşa koordinasyon ve entegrasyon projelerinde nasıl sunacaklarına odaklanmak için karşılıklı bağımlılıklarını tartışırlar. Her Scrum takımından uzman, bazen gerekirse bir temsilci vardır; Scrum takımlarının diğer üyeleri bu toplantıya katılabilir. Bu SoS toplantısında, günlük Scrum'da olduğu gibi, Scrum Takımları temsilcileri aşağıdakiler hakkında konuşurlar [54]:

- Ekibimiz ne yaptı?
- Ekibimiz ne yapacak?
- Ekibimi yavaşlatan herhangi bir engel var mı?
- Başka bir takımı yavaşlatan herhangi bir engel var mı?

3.7 Eserler

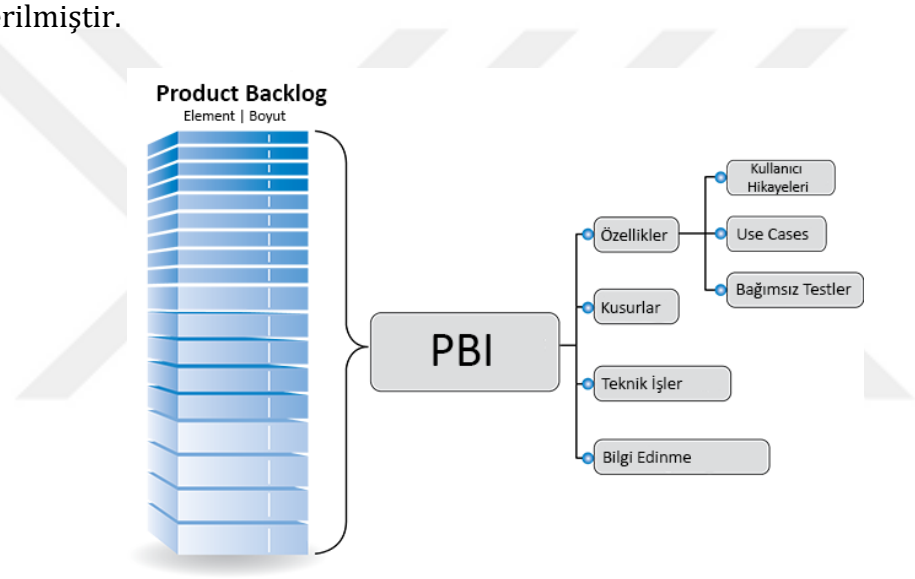
Scrum'daki eserler, herkesin kolayca anlayabilmesi ve denetim ve uyum için fırsat sağlaması için bilginin şeffaflığını maksimize etmeyi amaçlamaktadır. Scrum dört eseri tanımlar. Bunlar Ürün istek listesi, Sprint iş listesi, artış ve "bitti" adlı eserlerdir [56].

Scrum Eserleri aşağıda listelenmiştir;

- Ürün İş Listesi (Product Backlog)
- Sprint İş Listesi (Sprint Backlog)
- Ürün Parçası (Increment)
- Bitti (Ürün)

3.7.1 Ürün İş Listesi (Product Backlog)

Geleneksel metodolojideki iş listesi- fonksiyonel tasarım dokümanı arası bir dokümandır. Projede yapılacak işler mümkün olduğunca basit ifadelerle listelenir. İçeriği her sprintten sonra güncellenir [56]. Şekil 3.6'da ürün iş listesinin yapısı gösterilmiştir.



Şekil 3.6 Scrum Metodolojisinin İş Listesi (Product Backlog Item)

- Yapılacak projenin bütün gereksinimlerini içeren dokümandır.
- Proje sonunda elde edilecek yazılımdan beklenen özellik ve fonksiyonları tanımlar.
- PO bu iş listesinin içeriğini müşteri ve (varsa) sponsor beklentilerini de göz önüne alarak belirler.
- İhtiyaç ve istekler öncelik sırasına göre bu iş listesinde listelenir.
- Proje devam ederken ürün sahibi bu listeyi sürekli güncelleyebilir. Yeni özellikler ekleyip, listedeki fonksiyonların önceliklerini değiştirebilir.

- İş listesinden bulunan işlerin ne kadar sürede yapılabileceği tahmini olarak yazılır. Bu tahminler yapılırken geliştirme takımının öngörüsüne başvurulur. x günde/adam-ay'da biter diye iş listesine yazılır. Bu öngörüler öncelik belirlemeye kaynak oluşturur.
- Kısaca bu iş listesini oluşturan maddelerin önceliğini ve içeriğini PO belirler, süreleri geliştirme takımı belirler.
- PO süre tahmininden memnun olmazsa, maddenin içeriğini veya önem sırasını değiştirebilir. Buna bağlı olarak da takım yeni bir süre tahmininde bulunur.
- İş listesinin öncelikli işleri sprint iş listesinde yer alır. Önceliği düşük olanlar sonraki sprintlere bırakılır. Proje takımı sprinte başladıktan sonra bir daha iş listesine bakmaz. Sprint iş listesine odaklanır.

Sprint bittikten sonra ürün iş listesine dönülür, gerekli güncelleme yapılır, sprint boyunca bir kenara yazılan maddeler eklenir.

3.7.2 Sprint İş Listesi (Sprint Backlog)

Sprint iş listesi, bir sonraki sprint için seçilen bir PBI listesidir ve ayrıca sprint artışını sağlamak ve Sprint hedefini korumak için bir plandır. Sprint iş listesi öğeleri geliştirme ekibinin kapasitesi düşünülerek seçilmelidir. Ekip, Sprint iş listesinin kalemlerinin bir kısmını bir sprintte işleyemezse, Sprint iş listesine eklememelidir. Çünkü bu durum geliştirme ekibinin hızını azaltır [49].

Geliştirme ekibi tarafından seçilen kalemler, bu kalemleri arttırmaya yönelik görevler kararlaştırılmalı ve geliştirme ekibi tarafından büyüklük ve saatler verilmeli ve Scrum takip uygulamasında tutulmalıdır. Geliştirme ekibinin kendi kendini organize etmesini sağlar.

Yeni çalışma gerektiğinden, yalnızca geliştirme ekibi bunu sprint iş listesine ekleyebilir ve sprint sırasında yalnızca geliştirme ekibi sprint iş listesini değiştirebilir [52].

Sprint iş listesi şeffaftır ve herkese görünür durumdadır. Geliştirme ekibi tarafından ne yapılacağının gerçek zamanlı bir resmidir. Sprint'in nasıl gittiğini anlamak için, Sprint iş listesinde kalan çalışmayı gösteren genel olarak görüntülenen bir grafik

olan Sprint grafiği bulunmaktadır. Scrum Uzmanı, sprintin başlangıcında ve gün geçtikçe ideal çalışma grafiğini çizerek, bu Tablode kalan çalışmayı günceller. Herkes, sprint 'in nasıl gittiğini ve ekibin yapacağı ekstra çalışmaların olup olmadığını anlayabilir [56].

3.7.3 Bitti 'nin Tanımı (Done of Done- DoD)

Bir PBI veya bir artış "Bitti" olarak tanımlandığında, herkes "Bitinin" ne anlama geldiğini anlar. Farklı Scrum Takımlarında farklı olabilse de üyelerin kendi takımlarında ortak bir "Bitti" anlayışı olması gerekir. Genel Bitti kriterleri aşağıdakilerden herhangi birini içerebilir [56]:

- İhtiyaç analizinin tamamlanması
- Tüm belgelerin tamamlanması
- Yorumlayan diğer takım üyeleri bitti kararı
- Tamamlanan gelişim
- Tamamlanmış birim testi
- Kalite güvence testlerinin tamamlanması
- Kullanıcı kabul testlerinin tamamlanması
- Düzeltilen konuların listesi
- Paydaşlara ve / veya iş temsilcilerine başarılı bir şekilde gösterme.
- Ürünün Başarılı Dağıtımı.

İlk Sprint başlamadan önce, Bittinin Tanımı, Scrum ekibi tarafından belirlenmelidir. DoD belirlendiğinde, ekip bir görevin tamamlanmasını ve sonucun DoD'ye uygun olmasını sağlayabilir. Net bir DoD kritiktir çünkü standardizasyona yardımcı olur. Ayrıca ürünün daha iyi kalitede olmasına yardımcı olur. Ekibe "Bitti" standartlarını hatırlatmak için Scrum tahtada DoD tutulabilir.

3.7.4 Üründe Artış

Artış, bir sprint sırasında tamamlanan tüm Ürün İstek Listesi öğelerinin toplamıdır. Bir sprint 'in sonunda, yeni artış, "Bitti" olmalıdır. Bu da çıkartılan ürünün kullanılabilir durumda olmasını ve Scrum ekibinin "Bitti" tanımının karşılandığı anlamına gelir. Ürün olup olmadığına bakılmaksızın kullanılabilir durumda

(gönderilebilir ürün) olmalıdır. Son olarak ürün sahibi ürünü serbest bırakmaya (release) karar verebilecektir [49].

3.7.5 Hikâye Noktası (Story Point)

Bu tez çalışmasında en çok karşılaşacağımız kavramların başında Story Point kavramı gelmektedir. Story Point, yapılacak işin ne kadar zor olduğunu belirtmek için kullanılır. Üretkenliği veya işin ne kadar zaman alacağını göstermez! Aynı zorluktaki iki iş, farklı sürelerde bitebilir. Ama eğer işin karmaşıklık ya da büyüklüğünü tespit edebiliyorsak işin ne kadar sürede bitebileceğini diğer bir ifade ile eforunu tespit edebiliriz. Story Point değerlerini göstermede birçok farklı metot vardır [56]. Bu gösterimlerden bir tanesi t-shirt gösterimidir. Bu yöntemi, t-shirt'lerin S, M, L, XL gibi işaretlenmesi olarak da düşünebilirsiniz. Kullanılan bir başka gösterim ise aritmetik (1 ,2 ,3…) gösterimdir. Bu yöntemde de çok fazla tercih edilmemektedir. Bunun nedeni ise sayıların birbirine çok yakın olmasıdır. Örneğin yapılacak bir işin değeri belirlenirken üyelerden biri 5 değerini verirken diğeri 6 değerini verebilir. Burada değerler birbirine çok yakın olduğu için net ve istenen kararlar alınmayabilir. Story Pointleri göstermede en çok kullanılan yöntem ise Fibobacci yöntemidir. Bu yöntemde Fibonacci serisine benzer 0, 1, 2, 3, 5, 8, 13, 20, 40, 100 gibi rakamlar kullanılır, aradaki sayılar kullanılmaz. Bunun iki önemli nedeni var birincisi aradaki sayılar olursa çok fazla karmaşa olur, 5 mi yoksa 6 mı şeklinde, bunun yerine 5 mi 8 mi ayırımı daha kolay karar verilebilir bir durumdur. Diğer bir neden işlerin büyüklükleri ile o işteki belirsizlik veya belirsizlik çıkma olasılığı da doğrusal olarak artar o yüzden sayılar arasındaki farklar da giderek artıyor, mesela 40 ile 100 arasında ciddi bir fark var. Bu büyüklükteki sayılar genelde belirsizlik içeren ve bölünemeyen kullanıcı hikayeleri'leri (user story) için kullanılır. Bu çalışmada hem aritmetik hem de hem de Fibonacci gösterimleri kullanılmıştır.

Bu tez çalışmasında yazılım maliyet tahmini yapmak için geliştirilen üçüncü yöntem Scrum ile geliştirilen yazılımlar için maliyet tahmini yapmaktır. Bu nedenler bu bölümde Scrum mimarisi açıklandı. Scrum hakkında detaylı bilgi sunuldu.

4
Veri Setleri

Bu çalışmamız da üç farklı yazılım maliyet kestirim modeli geliştirilmiştir. Bu nedenle de birçok veri seti kullanılmıştır. Çalışmamızın ilk safhasında yapay sinir ağları modeli kullandık. Yapay sinir ağları modelini geliştirirken Cocomo81 veri setini kullandık. Çalışmamız da ikinci yöntem olarak en küçük kareler tabanlı regresyon yöntemlerini kullandık. En küçük kareler tabanlı regresyon modelimizde ise Cocomo81, Cocomonasa1 ve Cocomonasa2 veri setini kullandık. Çalışmamızın üçüncü safhasında ise Scrum yazılım geliştirme metodolojisinde yazılım efor tahmini yapmaya çalıştık. Bu yöntemde ise Scrum için özelleştirilmiş (customized) veri setini kullandık.

4.1 Promise Veri Setleri

Promise web sitesi üzerinde, açık kaynak kodlu olarak ulaşılabilecek ve efor tahmininde kullanılabilecek veri setleri mevcuttur [57]. Önerilen model için bu veri setlerinden Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri kullanılmıştır. Bu veri setleri yapay sinir ağları modeli ve en küçük kareler tabanlı regresyon modeli oluşturulurken kullanıldı. Cocomo81 veri setinde 63 tane proje örneğine ait yazılım maliyeti değerleri bulunmaktadır. Cocomonasa1 veri setinde ise 60 tane projeye ait yazılım maliyet değerleri mevcuttur. Cocomonasa2 ver seti içerisinde 94 tane projeye ait yazılım efor sonuçları mevcuttur [57]. Çalışmada daha sağlıklı sonuç elde edebilmek için tüm değerler normalize edilmiştir. Normalizasyondan sonra veri üzerinde model oluşturularak analizler yapılmıştır. Bu üç veri setinde 15 tane giriş değeri (Feature) bulunmaktadır. Bu nedenle bu veri setlerindeki parametreleri açıklarken Cocomonasa2 veri setini açıklamak yeterli olacaktır. Aşağıda Tablo 4.1'de veri setinin bir örneği gösterilmektedir.

Aşağıdaki Tablo 4.1'de Cocomonasa2 veri setine ait veriler mevcuttur. Bu veri setinde rely, data, cplx, time, stor, virt, turn, acap, aexp, pcap, vexp, lexp, modp, tool, sced, equivphyskloc, act_effort gibi sütünların olduğu görülmektedir. Rely, data,

cplx, time, stor, virt, turn, acap, aexp, pcap, vexp, lexp, modp, tool, sced, equivphyskloc giriş değerlerini oluştururken act_effort ise çıkış değeri olan gerçek eforu ifade etmektedir [57].

Tablo 4.1 Cocomo Nasa2 Veri seti Örneği

Mode	semi-detached	semi-detached	semi-detached	semi-detached	semi-detacheed
rely	h	h	h	h	h
data	l	l	l	l	l
cplx	h	h	h	h	h
time	n	n	n	n	n
stor	n	n	n	n	n
virt	l	l	l	l	l
turn	l	l	l	l	l
acap	n	n	n	n	n
aexp	n	n	n	n	n
pcap	n	n	n	n	n
vexp	n	n	n	n	n
lexp	h	h	h	h	h
modp	h	h	h	h	h
tool	n	n	n	n	n
sced	l	l	l	l	l
kloc	25,9	24,6	7,7	2,2	66,6
act_effort	117,6	117,6	31,2	8,4	352,8

Tablo 4.1'de Cocomonasa2 veri setinin bir örneği verilmiştir. Veri seti, çalışmalarda 2 bölüme ayrılmış, %75'lik bölümü eğitim için, kalanı ise test için kullanılmıştır. Kümelerin oluşturulması için bir rastgele fonksiyonu kullanılmıştır. Bu fonksiyona göre, tüm küme içerisinden belli sayıdaki satırlar 2 yeni alt kümeyi oluşturmuştur. Özellikle eğitim aşamasında veri kümesinin içeriği ile beraber, verilerin sırası da sonuçları etkileyebileceğinden, daha modüler bir yapıya ulaşmak için rastgele elemanların seçilmesi önemlidir. Özellikle YSA oluştururken başlangıç değerleri ve kümeler, sonuçta oluşan alt kümeleri çok fazlaca etkilemektedir. Bu nedenle eğitim

ve test veri kümelerinin oluşturulmasında sıra gözetilmeden, rastgelelik sağlayan metot tercih edilmiştir.

Tablo 4.1'de giriş parametrelerinin kısaltılmış halini görmekteyiz. Bu kısaltmaların ne olduğunu ayrıntılı bir şekilde ele alacağız. Ayrıca bu giriş parametrelerinin değerlerinin karşısında çok düşük (vl-very low), düşük (l-low), nominal (n-nominal), yüksek(h-high), çok yüksek (vh-very high), ekstra yüksek (xh-extra high) gibi değerlerin olduğu görülmektedir [57]. Bu sayısal olmayan değerlerin karşısında ise sayısal karşılıkları yapılan birçok çalışma neticesinde ortaya konulmuştur [3]. Bu değerler ise Tablo 4.2'de listelenmiştir.

Tablo 4.2 Veri Seti Parametreleri Aldığı Değerlerin Sayısal Karşılığı

Özelliker	Çok Aşağı	Aşağı	Nominal	Yüksek	Çok Yüksek	Ekstra Yüksek
Required Software Reliability	0,75	0,88	1	1,15	1,4	
Database size		0,94	1	1,08	1,16	
Product Complexity	0,7	0,85	1	1,15	1,3	1,65
Execution Time Constraints			1	1,11	1,3	1,66
Main Storage Constraints			1	1,06	1,21	1,56
Virtual Machine Volatility		0,87	1	1,15	1,3	
Computer Turnaround Time		0,87	1	1,07	1,15	
Analyst Capability	1,46	1,19	1	0,86	0,71	
Applications Experience	1,29	1,13	1	0,91	0,82	
Programmer Capability	1,42	1,17	1	0,86	0,7	
Virtual Machine Experience	1,21	1,1	1	0,9		
Programming Language Experience	1,14	1,07	1	0,95		
Modern Programming Practices	1,24	1,1	1	0,91	0,82	
Use of Software Tools	1,24	1,1	1	0,91	0,83	
Require Development Schedule	1,23	1,08	1	1,04	1,1	

Güvenirlik (**RELY Relaibility**): Yazılım ürünün kullanıcıyı tatmin edecek şekilde SRS dokümanında istenilen fonksiyonları gerçekleştirmesini ifade eder. Bu parametrenin değerini belirlerken birçok yazılım geliştirme aktivitesi göz önünde bulundurulur. Rely parametresinin değeri aşağıdaki durumlar dahilinde belirlenir [58].

Very-Low (Çok Düşük): bir yazılım arızasının etkisi yazılım geliştiriciler tarafından kolayca görülüp düzeltilebiliyorsa rely parametresinin değeri "*çok düşük*" olur.

Low (Düşük): Bir yazılım arızasının etkisi, kullanıcılar için düşük düzeyde ve kolayca düzeltilebilir seviyede ise *Düşük* değerini alır

Nominal (Normal): Bir yazılım arızasının etkisi, kullanıcılar için ölçülü bir seviyede ise, fakat bu hatanın düzeltilmesi ekstra yük gerektiriyorsa bu durumda rely parametresi "*nominal*" değer alır.

High (Yüksek): Biz yazılım hatası ciddi finansal bir probleme sebebiyet veriyorsa ve kişilerin güvenini zedelemeye yol açabiliyorsa bu durumda rely parametresi "*yüksek*" değerini alır.

Very High (Çok Yüksek): Eğer bir yazılım arızası insan ölümüne sebebiyet verebilecek bir durumda ise rely parametresi "*çok yüksek*" değerini alır.

Extra High (Ekstra Yüksek): "*çok yüksek*" ile ayni değeri alır

Veri Boyutu (Data -Data Size): Geliştirilen yazılım ürününün veri tabanı büyüklüğüdür. Bir yazılım ürününü geliştirmek için gereken işgücü miktarı, açıkça veri tabanı ile doğrudan alakalıdır. D/P = (Database size in bytes or characters) / (Program size in SLOC seklinde formüle edip buna bağlı olarak data parametresinin değerini belirleyebiliriz [58].

Very-Low (Çok Düşük): D/P'nin varsayılan değeri olarak gelir.

Low (Düşük): D/P <10

Nominal (Normal): 10 <= D/P <= 100

High (Yüksek): 100 <= D/P <= 1000

Very High (Çok Yüksek): D/P> 1000

Extra High (Ekstra Yüksek): "çok yüksek" ile ayni değeri alır

Karmaşıklık (Cplx-Product Complexity): Bir yazılımın karmaşıklığını ifade eder. Karmaşıklık daha çok sübjektif bir konu olduğu için, bu durumu ortadan kaldırmak ve cplx parametresinin değerini belirlemek adına bir kontrol fonksiyonu belirlenmiştir. Bu kontrol fonksiyonu; kontrol, hesaplama, cihaz bağımlılığı ve veri yönetim operasyonlarından oluşmaktadır [58].

Cplx - ***(Hesaplama Operasyonları- Computational Operations) için***

Very-Low (Çok Düşük): basit operasyonlar için bu değeri alır. Örneğin A = B + C x (D-E).

Low (Düşük): Orta seviye operasyonlar için bu değeri alır. D = sqrt (B^2 - 4.0 x A x C

Nominal (Normal): Standart matematik ve istatistik operasyonları için bu değeri alır. Örneğin matris ve vektör işlemleri gibi

High (Yüksek): basit numerik analiz gibi işlemler için kullanılır. Örneğin diferansiyel denklemler gibi

Very High (Çok Yüksek): Zor numerik analiz yöntemleri gibi durumları ifade eder. Parçalı diferansiyel denklemler gibi

Extra High (Ekstra Yüksek): hassas durumların gerektirdiği ve ölçümlerin yapıldığı buna bağlı olarak skolastik çözümlerin sunulduğu durumları ifade eder.

Cplx - ***(Cihaz Bağımlı Operasyonları- Device-Dependent Operations) için***

Very-Low (Çok Düşük): Basit bir formatta basit okuma ve yazma işlemleri için bu değeri alır.

Low (Düşük): İşlemecilerin belireli kavramlar üzerinde özelleştirilmediği ve aranmadığı durumları ifade eder. I/O işlemleri GET/PUT seviyesinde yapılır ve üst üste yazma yoktur.

Nominal (Normal): I/O işlemlerinde cihaz seçebilme imkânın olduğu, durum kontrolünü olduğu ve hata işlemlerini yapıldığı durumlarda bu değeri alır.

High (Yüksek): I/O işlemlerinin fiziksek ortamlarda yapıldığı durumlarda bu değeri alır. (Fiziksel depolama adres çevirileri; seeks, reads …)

Very High (Çok Yüksek): Program kesmelerinin(interrupt) çalışması, maskelenmesi ve tanımlanmasının olduğu durumlarda bu değeri alır. Ayrıca protokollerin oluşturulduğu durumlarda da bu değeri alır.

Extra High (Ekstra Yüksek): Mikro programlama durumlarında ve Cihaz zamanlamasına bağlı kodlama durumlarında bu değeri alır.

Cplx - ***(Veri Yönetim Operasyonları- Data Management Operations) için***

Very-Low (Çok Düşük): Hafızada gerçekleştirilen basit dizi işlemleri için bu değeri alır.

Low (Düşük): sadece bir dokuman ve onun bir tane alt dosya işlemin yapıldığı durumlarda bu değeri alır. Burada ara dosya yoktur. Dosya düzenleme yoktur.

Nominal (Normal): birçok girişin olduğu sadece bir tane çıkışın olduğu orta seviyede işlemlerin gerçekleştiği durumları ifade eder.

High (Yüksek): karmaşık veri yapılarının oluşturulduğu ve kayıt işlemin yapılabildiği ayrıca bir data akışının olduğu durumlarda bu değeri alır.

Very High (Çok Yüksek): Dosyaların oluşturulduğu, komut satırından işlemlerin yapılabildiği ve arama optimizasyonun olacağı durumlarda bu değeri alır.

Extra High (Ekstra Yüksek): yüksek bağımlılığın olduğu, dinamik yapıların oluşturulacağı ve doğal dil isleme yapılarının oluşturulacağı durumlarda bu değeri alır.

Analistlerin Yetkinliği (Acap-Analyst Capability): Analistler, gereksinimlerin ve ön tasarım özelliklerinin geliştirilmesi ve onaylanmasında rol alırlar. Analistler detaylı tasarım ve kodlama aktivitelerinde danışmanlık hizmeti verirler. Analistler entegrasyon ve testlerde ağırlıklı olarak rol alırlar. Analist Yetkinliği değeri, yazılım analistlerinin genel nüfusuna göre yüzdelikler cinsinden ifade edilir. Burada göz önünde bulundurulması istenen ana unsurla yetenek, verimlilik, kusursuzluk ve iletişim ve yönteme becerisidir. Burada tecrübe göz önünde bulundurulmaz. Burada kişiden ziyade takım başarısına odaklanılır [58].

Very-Low (Çok Düşük): %15

Low (Düşük): %35

Nominal (Normal): %55

High (Yüksek): %75

Very High (Çok Yüksek): %90

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Uygulama Deneyimi (Aexp-Application Experience): yazılım ürününü geliştiren proje ekibinin eşdeğer uygulama deneyim düzeyini temsil eder [58].

Very-Low (Çok Düşük): deneyim <4 ay

Low (Düşük): Deneyim>= 1 yıl

Nominal (Normal): Deneyim>=3 yıl

High (Yüksek): Deneyim>= 6 yıl

Very High (Çok Yüksek): Deneyim>= 12 yıl

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Programcı Yetkinliği (Pcap-Programmer Capability): Yazılım urunu üzerinde çalışan programcıların yetkinliğini ifade eder. Programcıların yetkinliği, programcıların toplam nüfusuna göre yüzde cinsinden ifade edilir. Burada göz önünde bulundurulması istenen ana unsurla yetenek, verimlilik, kusursuzluk ve iletişim ve yönteme becerisidir. Burada tecrübe göz önünde bulundurulmaz. Burada kişiden ziyade takım başarısına odaklanılır [58].

Very-Low (Çok Düşük): %15

Low (Düşük): %35

Nominal (Normal): %55

High (Yüksek): %75

Very High (Çok Yüksek): %90

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Sanal Makine Deneyimi (Vexp- Virtual Machine Experience): proje ekibinin yazılım ürününün görevlerini yerine getirmek için çağırdığı donanım ve yazılım

kompleksi ile olan deneyimini temsil eder. Örneğin bilgisayarlar, işletim sistemleri, veri tabanı yönetim sistemleri (burada yazılım dili göz önünde buldurulmaz) [58].

Very-Low (Çok Düşük): Deneyim <Ay

Low (Düşük): 1 ay <Deneyim <4 Ay

Nominal (Normal): 4 ay <Deneyim <1 Yıl

High (Yüksek): 1 Yıl <Deneyim <3 yıl

Very High (Çok Yüksek): "Yüksek" ile ayni değeri alır.

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Programlama Dili Deneyimi (Lexp- Programming Language Experience): Bu, yazılım ekibini geliştiren proje ekibinin programlama dili deneyim düzeyini temsil eder. Programlama dili deneyimi, proje takımının kullanılacak programlama dili ile eşdeğer deneyim süresi açısından tanımlanır [58].

Very-Low (Çok Düşük): Deneyim <Ay

Low (Düşük): 1 ay <Deneyim <4 Ay

Nominal (Normal): 4 ay <Deneyim <1 Yıl

High (Yüksek): 1 Yıl <Deneyim <3 yıl

Very High (Çok Yüksek): "Yüksek" ile ayni değeri alır.

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Bilgisayar Dönüş Süresi (Turn-Turnaround Time): Yazılım Geliştirici tarafından yapılan bir işin, yayına alınıp sonucunun yazılımcıya iletilmesine kadar geçen ortalama süre olarak ifade edilir. "Avg Turn" ortalama dönüş(turn) olarak ifade edilir [58].

Very-Low (Çok Düşük): Düşük ile aynı değeri alır.

Low (Düşük): avg turn <4 saat

Nominal (Normal): 4 saat <avg turn <12 saat

Very High (Çok Yüksek): avg turn >12 saat

Extra High (Ekstra Yüksek): "Çok Yüksek" ile aynı değeri alır.

Modern Programlama Kullanımı (Modp–Use of Modern Programming): Yazılım geliştirmede kullanılan modern programlama uygulamalarının derecesini temsil eder [58].

Very-Low (Çok Düşük): hiç kullanılmadığı durumlarda

Low (Düşük): yeni kullanılmaya başlandığında

Nominal (Normal): kısmi olarak kullanılmaya başlandığında

High (Yüksek): genel olarak kullanılmaya başlandığında

Very High (Çok Yüksek): rutin olarak kullanıldığında

Extra High (Ekstra Yüksek): "*çok yüksek*" ile aynı değeri alır

Çalıştırma Süresi Kısıtlamaları (Time-Execution Time Constraint): Yazılım ürününe uygulanan yürütme süresi kısıtlamasının derecesini yansıtır. Derecelendirme, kullanılabilir yürütme süresi cinsinden ifade edilir [58].

Very-Low (Çok Düşük): *Düşük* ile aynı değeri alır.

Low (Düşük): Nominal ile aynı değeri alır.

Nominal (Normal): <=%50 kullanılabilir yürütme süresi kullanımı

High (Yüksek): <=%70 kullanılabilir yürütme süresi kullanımı

Very High (Çok Yüksek): <=%85 kullanılabilir yürütme süresi kullanımı

Extra High (Ekstra Yüksek): <=%95 kullanılabilir yürütme süresi kullanımı

Yazılım Araçlarının Kullanımı (Tool-Use of Software Tools): yazılım ürününün geliştirilmesinde kullanılan yazılım araçlarının kullanım derecesini gösterir [58].

Very-Low (Çok Düşük): Temel araçlar, ör. assembler, linker, monitor, hata ayıklama yardımcıları. *Low (Düşük):* Daha verimli araçların kullanılmaya başlandığında. Örneğin, Üst Düzey Dil derleyici, makro derleyici, kaynak düzenleyici, temel kütüphane yardımcıları, veri tabanı yardımcıları

Nominal (Normal): Gerçek zamanlı işletim sistemleri, veri tabanı yönetim sistemi, etkileşimli hata ayıklayıcılar, etkileşimli kaynak editörü gibi araçlar kullanıldığında

High (Yüksek): Sanal işletim sistemleri, veri tabanı tasarımına yardımcı programlar, programlama tasarım dili araçları, performans ölçümü araçları, analiz araçları, program akışı araçları ve test analizörü gibi araçların kullanılmaya başlanmasında.

Very High (Çok Yüksek): Konfigürasyon yönetimi destekleyen araçlar, programlama destek kütüphanesi, entegre dokümantasyon sistemi, proje kontrol sistemi, genişletilmiş tasarım araçları, otomatik doğrulama sistemi gibi ileri düzey araçların genel olarak kullanılmaya başlanmasında

Extra High (Ekstra Yüksek): "*çok yüksek*" ile aynı değeri alır.

Ana Depolama Üzerindeki Kısıtlamalar (Stor-Main Storage Constraint): Yazılım ürünü tarafından kullanılması beklenen ana depolama yüzdesini ve ana depolama kaynaklarını kullanan tüm alt sistemleri yansıtır [58].

Very-Low (Çok Düşük): *Düşük* ile aynı değeri alır.

Low (Düşük): Nominal ile aynı değeri alır.

Nominal (Normal): <=%50 kullanılabilir depolama alanı kullanımı

High (Yüksek): <=%70 kullanılabilir depolama alanı kullanımı

Very High (Çok Yüksek): <=%85 kullanılabilir depolama alanı kullanımı

Extra High (Ekstra Yüksek): <=%95 kullanılabilir depolama alanı kullanımı

Sanal Makine Volatilitesi (Virt- Virtual Machine Volatility): Geliştirilecek yazılım ürününün altında yatan sanal makinenin oynaklık seviyesini yansıtır. Sanal makine, görevlerin yerine getirilmesi için donanımın ve yazılımın bir bütün olarak çalışması şeklinde ifade edilir. Derecelendirmesi ise majör ve minör değişikliklerin göreceli sıklığı şeklinde tanımlanır [58].

Büyük değişiklik: Geliştirilmekte olan rutinlerin kabaca%10'unu önemli derecede etkiler.

Küçük değişiklik: Geliştirilmekte olan rutinlerin kabaca%1'ini önemli derecede etkiler.

Very-Low (Çok Düşük): *Düşük* ile aynı değeri alır.

Low (Düşük): Her 12 ayda bir büyük bir değişiklik oluyorsa

Nominal (Normal): Her 6 ayda bir büyük bir değişiklik ve iki haftada bir küçük bir değişiklik oluyorsa

High (Yüksek): Her 2 ayda bir büyük bir değişiklik ve her hafta küçük bir değişiklik oluyorsa

Very High (Çok Yüksek): İki haftada bir büyük bir değişiklik ve iki günde bir küçük bir değişiklik oluyorsa

Extra High (Ekstra Yüksek): "*çok yüksek*" ile aynı değeri alır.

Takvim Üzerindeki Kısıtlamalar (Sced-Schedule Constraint): Bir yazılım ürününü geliştiren proje ekibine uygulanan kısıtlamaların seviyesini temsil eder. Derecelendirmeler, belirli bir çaba gerektiren bir proje için nominal zaman göre zaman Tablosindeki sapmaların yüzde olarak ifade edilmesi olarak tanımlanır [58].

Very-Low (Çok Düşük): Nominal değerin%75'i

Low (Düşük): Nominal değerin%85'i

Nominal (Normal): Nominal değerin%100'ü

High (Yüksek): Nominal değerin%130'u

Very High (Çok Yüksek): Nominal değerin%160'ı

Extra High (Ekstra Yüksek): "*Çok yüksek*" ile aynı değeri alır.

```
Eforu    | acap | analysts capability
azaltmak | pcap | programmers capability
için     | aexp | application experience
bunları  | modp | modern programing practices
arttır   | tool | use of  software tools
         | vexp | virtual machine experience
         | lexp | language experience
---------+------+---------------------------
         | sced | schedule constraint
---------+------+---------------------------
Eforu    | stor | main memory constraint
azaltmak | data | data base size
için     | time | time constraint for cpu
bunları  | turn | turnaround time
azalt    | virt | machine volatility
         | cplx | process complexity
         | rely | required software reliability
```

Şekil 4.1 Yazılım Parametreleri ile Efor Arasındaki İlişki [58]

Şekil 4.1'de yazılım parametreleri ile yazılım eforu arasındaki ilişki gösterilmiştir. Analist Yetkinliği, programcı yetkinliği, uygulama geliştirme deneyimi, modern programlama dili deneyimi, yazılım araçlarını kullanabilme yetkinliği, sanal makine deneyimi ve programlama dili deneyimi arttıkça yazılım eforunun azalacağı görülmektedir. Diğer yanda Ana hafıza üzerindeki kısıtlamalar, veri tabanı boyutu, işlemci üzerindeki kısıtlamalar, işlemlerin karmaşıklığı ve yazılım güvenirliği arttıkça yazılım maliyetinin bu veri seti üzerinde çeşitli işlemeler yaptık. Bu işlemler çevresel faktörlerin ortadan kaldırılması ve sadece kod satısı sayısı ile efor arasında bir ilişki kuracak şekilde verilerin düzenlenmesini içermektedir. Veri setinin oluşan yeni formu üzerinde modeller geliştirildi.

4.2 Scrum Veri Seti

Scrum, yinelemeli ve artımlı bir çevik yazılım geliştirme metodolojisidir ve geliştirme sürecinde müşterilerin gereksinimlerindeki hızlı değişimlere hızlı bir şekilde cevap vermeyi sağlar [56]. Bu yaklaşım "ampirik yaklaşım" olarak tanımlanmaktadır. Scrum yaşam döngüsünde sprintler mevcuttur. Her bir sprint sonucunda projenin bir iterasyounu ya da artırımı yapılmaktadır. Scrum'da temel olarak üç tane temel safha vardır. İlk safha "ürün iş listesi", ikincisi ise "sprint iş listesi" safhası ve son olarak ta "ürün bitti" safhası vardır [49]. Ürün iş listesinde yapılacak işler listelenir. Eğer ürün iş listesinde yapılacak olan işlerin süresi doğru bir şekilde belirlenebilirse yapılacak olan işin gerçek eforu ortaya konmuş olur. Bu çalışmamız da Scrum metodolojisi ile geliştirilen projeler için maliyet tahmini yapmaya çalıştık. Yazılım efor tahminini asıl olarak ürün iş listesinde bulunan işler belirlemektedir. Ürün iş listesindeki her bir iş için Story Point'ler belirlenir. Story Pointlerin toplamı projenin gerçek eforunu belirler. Bu nedenle asıl önemli olan Story Pointleri doğru bir şekilde belirleyebilmek. Sprint planlama toplantısında bir sprint içine girebilecek işler belirlenir. Bu toplantıda her bir iş için Story Pointler takım üyeleri tarafından belirlenir. Bu puanlama delphi yöntemine benzemektedir. Story Pointler belirlenirken özelleştirilmiş Fibonanci serisi kullanılır. Story Pointler belirlenirken riskler ve yapılacak işin karmaşıklığı göz önünde bulundurulur. Böylece doğru efor tahmini yapılmaya çalışılır. Bu çalışmamızda MongoDB, JIRA, JBOSS, APACHE ve SPRING ortamında geliştirilen projelerden oluşan Scrum

ortamında oluşturulan veri setleri kullanıldı [59]. Bu veri setlerini öncelikle 4 tane ana safhaya ayırdık. Her bir safhada yapılacak işler ve sprintler belirlendi. Son aşama olarak ta bu sprintlerde yer alan işler için Story Pointler belirlendi.

Tablo 4.3 Apache Veri Seti 0-30 fazı

Storypoint	1	1	2	1	1
Type	Task	Improvement	Bug	Bug	Bug
Priority	Minor	Major	Major	Minor	Minor
No_comment	1	1	0	0	0
No_affectversion	0	0	1	0	0
No_fixversion	0	0	0	0	0
No_issuelink	0	2	1	1	0
No_blocking	0	0	0	0	0
No_blockedby	0	0	0	0	0
No_fixversion_change	1	1	0	1	0
No_priority_change	0	0	0	0	0
No_des_change	0	0	0	0	0
Gunning_fog	Medium	Medium	Hard	Hard	Medium

Tablo 4.3' de projenin ilk safhasında yapılacak işler belirlendi. Bu işlerin ilgili sprintleri belirlendi. Yapılacak işler sprintlere atıldıktan sonra Scrum gözden geçirme toplantısında her bir iş için Story Pointler belirlendi. Bu safha projenin ilk safhasıdır. Projenin yaklaşık olarak %30'luk kısmını oluşturmaktadır. Bu oran değişebilir. Bu oran belirlenirken teslim edilecek iş miktarı göz önünde bulundurulur. Burada ürün iş listesinin ilk aşaması da tamamlanmış oldu. Bundan sonra ikinci safhada yapılacak işler belirlenir. Tablo 4.4'de ise projenin %20'lik kısmını oluşturan yapılacak işler belirlenir. Bu işler ilgili sprintlere atılır. Yapılacak işler ilgili sprintlere atıldıktan sonra Scrum gözden geçirme toplantısında Story Point ataması gerçekleştirilir. Ayrıca birinci safhadan gelen ve tamamlanamayan işler bu safhada tekrar puanlanır. Bu safhada yapılması planlanan işler ilk safhadan kaynaklanan problemlerden dolayı iyileştirilir.

Tablo 4.4 Apache Veri Seti 30-50 fazı

Storypoint	1	1	2	1	1
Type	Task	Improvement	Bug	Bug	Bug
Priority	Minor	Major	Major	Minor	Major
No_comment	1	1	0	0	2
No_affectversion	0	0	1	0	0
No_fixversion	0	0	0	0	0
No_issuelink	0	2	1	1	1
No_blocking	0	0	0	0	0
No_blockedby	0	0	0	0	0
No_fixversion_change	1	1	0	1	0
No_priority_change	0	0	0	0	0
No_des_change	0	0	0	0	0
Gunning_fog	Easy	Easy	Hard	Hard	Easy

Tablo 4.4'de ise projenin üçüncü safhasını oluşturan işler bulunmaktadır. Bu safha projenin %20'lik kısmını oluşturur. Bu oran ifade ettiğimiz gibi değişebilir. Üçüncü safhada yapılacak işler belirlendikten sonra ilgili sprintelere atılır. Bir önceki safhadan gelen işler varsa tekrar puanlanır. Ayrıca bu safhada yapılacak ve belirlenmiş olan işler puanlanır. Daha önceki safhalarda yapılan işler "done'a" çekilir. Böylece ürün iş listesi güncellenmiş olur.

Tablo 4.5 Apache Veri Seti 50-80 fazı

Storypoint	1	1	2	1	1
Type	Task	Improveme	Bug	Bug	Bug
Oriority	Minor	Major	Major	Minor	Major
No_comment	1	1	2	2	2
No_affectversion	0	0	1	0	0
No_fixversion	0	0	0	0	5
No_issuelink	0	2	1	1	4
No_blocking	0	0	0	0	0
No_blockedby	0	0	0	0	0
No_fixversion_change	1	1	0	1	0
No_priority_change	0	0	0	0	0
No_des_change	0	0	0	0	0
Gunning_fog	Medium	Medium	Hard	Hard	Medium

Tablo 4.6 Apache Veri Seti 80-100 fazı

Storypoint	1	1	1	1	1
Type	Task	Improvement	Bug	Bug	Bug
Priority	Minor	Major	Minor	Minor	Major
No_comment	1	3	2	2	2
No_affectversion	0	0	0	0	0
No_fixversion	0	0	0	0	5
No_issuelink	0	2	1	0	4
No_blocking	0	0	0	0	0
No_blockedby	0	0	0	0	0
No_fixversion_change	1	1	1	1	0
No_priority_change	0	0	0	0	0
No_des_change	0	0	0	0	0
Gunning_fog	Medium	Medium	Hard	Medium	Medium

Tablo 4.6'da ise projenin son safhasını oluşturan işler listelenmektedir. Daha önceki safhalardan kalan işler burada tekrar ele alınır. Scrum gözden geçirme toplantısında

yapılacak işler ve bir önceki safhalardan gelen işler puanlanır. Daha önceki safhalardan tamamlanmış işler "Yapıldıya" çekilir. Böylece ürün iş listesi güncellenmiş olur.

Tablo 4.3, Tablo 4.4, Tablo 4.5 ve Tablo 4.6 Apache ortamında Scrum metodoloji ise geliştirilen projelere ait küçük bir örnek sunulmuştur. Veri setinin orijinal halinde 5827 adet örnek mevcuttur. Bu veri setinin benzeri Jira, JBoss, Spring ve MongoDB ortamında geliştirilmiştir. Burada Jira ve Apache veri setlerinden örnekler sunulmuştur.

Veri setini oluşturan parametreler Tablo 4.7'de gösterilmiştir. Veri seti; Priority, Number of comments, Number of affect versions, Number of fix versions, Issue links, Number of blocking issues, Number of blocked issues, changing of fix versions, Changing of priority, Changing of description ve Complexity of description özelliklerinden oluşup, yazılım efor tahminin kullanılacak temel özelliklerdir. Tablo 4.7'de ayrıntılı bir şekilde bu özellikler açıklanmıştır.

Tablo 4.7 Veri Setini Oluşturan Özellikler

	Özellik	**Açıklaması**
type	Tür	Yapılacak işin Türü
priority	Öncelik	Yapılacak işin Önceliği
no_comment	Yorum Sayısı	Yapılacak iş ile ilgili yorum sayısı
no_affectversion	Etkilenen sürümlerin sayısı	Bir sorunun bulunduğu sürümlerin sayısı
no_fixversion	Düzeltme sürümleri sayısı	Bir sorunun olduğu veya çıkacağı sürüm sayısı
no_issuelink	İş bağlantıları	Yapılacak Bir işin bağlantı sayısı
no_blocking	Engelleyen işlerin sayısı	Yapılacak isin Çözülmesi için bu isi engelleyen İşlerin sayısı
no_blockedby	Engellenen islerin sayısı	Yapılacak iş tarafından engellenen iş sayısı
no_fixversion_change	Düzeltme sürümlerinin değiştirilmesi	Düzeltme sürümünün değiştirilme sayısı
no_priority_change	Önceliğin değiştirilmesi	Bir İş'in önceliği değiştirilme sayısı
no_des_change	Açıklamanın değiştirilmesi	Bir işin açıklamasının değiştirilme sayısı
gunning_fog	Açıklamanın karmaşıklığı	Okuma yeteneği endeksi (Gunning Fog [21]), kolay ve zor bir şekilde kodlanan bir açıklamanın karmaşıklık seviyesini gösterir

Tür (Type – İş Türü): Yapılacak işin türünü belirler. Bu türler Bug, Documentation, Epic, Improvment, New Feature, Story, Sub-Task, Task, Technical Task, Test, Umbrella ve Wish değerini almaktadır.

Öncelik (Priority): Yapılacak işin Önceliği belirler. Minor, Major, Blocker, Critical, blocker ve trival değerlerini alabilmektedir.

Açıklamanın karmaşıklığı (Description Complexity): Okuma yeteneği endeksi (Gunning Fog [60]), kolay ve zor bir şekilde kodlanan bir açıklamanın karmaşıklık seviyesini gösterir. Easy, Medium ve Hard değerlerini almaktadır.

Tablo 4.8 JIRA Veri Seti 0-30 fazı

Storypoint	1	1	1	3	3	1
Type	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task
Priority	Minor	Minor	Minor	Minor	Minor	Minor
No_comment	0	0	0	0	0	0
No_affectversion	0	0	0	1	0	0
No_fixversion	2	2	2	1	0	0
No_issuelink	0	0	0	1	0	0
No_blocking	0	0	0	0	0	0
No_blockedby	0	0	0	0	0	0
No_fixversion_change	0	0	0	0	0	0
No_priority_change	0	0	0	0	0	0
No_des_change	0	0	0	0	0	0
Gunning_fog	Easy	Easy	Easy	Medium	Easy	Easy

Tablo 4.9 JIRA Veri Seti 30-50 fazı

Storypoint	1	1	1	3	3	1
Type	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task	Dev. Sub-task
Priority	Minor	Minor	Minor	Minor	Minor	Minor
No_comment	0	0	0	0	0	0
No_affectversion	0	0	0	1	0	0
No_fixversion	2	2	2	1	0	0
No_issuelink	0	0	0	1	0	0
No_blocking	0	0	0	0	0	0
No_blockedby	0	0	0	0	0	0
No_fixversion_change	0	0	0	0	0	0
No_priority_change	0	0	0	0	0	0
No_des_change	0	0	0	0	0	0
Gunning_fog	Easy	Easy	Easy	Easy	Easy	Easy

Tablo 4.10 JIRA Veri Seti 50-80 fazı

storypoint	1	1	1	1	3	1
Type	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task	Dev. Sub-task
Priority	Minor	Minor	Minor	Minor	Minor	Minor
No_comment	0	0	0	0	0	0
No_affectversion	0	0	0	0	1	0
No_fixversion	2	2	2	0	1	0
No_issuelink	0	0	0	0	1	0
No_blocking	0	0	0	0	0	0
No_blockedby	0	0	0	0	0	0
No_fixversion_change	0	0	0	0	0	0
No_priority_change	0	0	0	0	0	0
No_des_change	0	0	0	0	0	0
Gunning_fog	easy	easy	easy	easy	medium	easy

Tablo 4.11 JIRA Veri Seti 50-80 fazı

storypoint	1	1	1	1	3	3
Type	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task	Sub-task
Priority	Minor	Minor	Minor	Minor	Minor	Minor
No_comment	0	0	0	0	0	0
No_affectversion	0	0	0	0	1	0
No_fixversion	2	2	2	0	1	0
No_issuelink	0	0	0	0	1	0
No_blocking	0	0	0	0	0	0
No_blockedby	0	0	0	0	0	0
No_fixversion_change	0	0	0	0	0	0
No_priority_change	0	0	0	0	0	0
No_des_change	0	0	0	0	0	0
Gunning_fog	easy	easy	easy	easy	medium	easy

Tablo 4.8, Tablo 4.9, Tablo 4.10 ve Tablo 4.11 de Scrum metodolojisi kullanılarak Scrum ortamında geliştirilen projelere ait örnekler verilmiştir.

Bu veri setini üzerinde bir dizi işlemler gerçekleştirdik. Verinin ham hali 99 tane özellikten oluşmaktadır. Fakat veri setini incelediğimizde bu özelliklerin çoğunun değerinin olmadığını gördük. Bu nedenle değeri olmaya özellikleri veri setimizden çıkardık. Ayrıca bazı özelliklerin diğer özellikler tarafından kapsandığını tespit ettik. Bu nedenle bu özellikleri de veri setimizden çıkardık. Örnegin min_no_issuelink (minimum number of issues link), max_no_issuelink (maximum number of issues link) ve avg_no_issulink (average number of issues link) özelliklerine baktığımızda avg_no_issulink özelliğinin diğer özellikler tarafından kapsandığını açıkça görebiliyoruz. Veri seti üzerinde yaptığımız son işlem ise özellik azaltmada kullanılan PCA (Principle Component Analysis) kullanmaktı. Burada da hiç veri kaybına sebep olmayacak özellikleri tespit edip veri setinden çıkardık. Böylece veri setinin yeni versiyonu oluşturuldu.

Öncelikle oluşturacağımız model faz tabanlı olduğu için veri seti için fazlar belirledik. Veri setinin yeni formu üzerinde 4 tane faz belirledik. Veri tabanı ve alt modülerinin geliştirilmesi (database and low module development) ilk fazı oluşturur. Servis ve kullanıcı modüllerinin geliştirilmesi (service and user module development) ikinci fazı oluşturur. Modüllerin testi ve beta test (module test and beta test) üçüncü fazı oluşturur. Bakım ve destek(maintenance) ise dördüncü fazı oluşturur. Veri setindeki verileri ilgili iterasyonlara ve daha sonra ilgili fazlara attık. Böylece üçüncü modelin üzerinde koşturulacağı veri seti hazırlanmış oldu.

Bu bölümde yazılım maliyet tahmininde kullandığımız farklı veri setlerini, veri setlerinin özelliklerini ve veri seti üzerinde yapılan işlemleri açıkladık.

5
Geliştirilen Modeller ve Sonuçlar

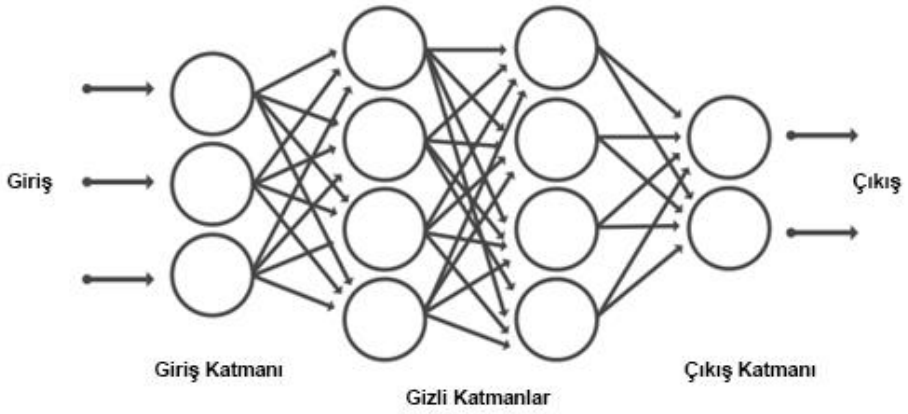
5.1 Yapay Sinir Ağları Tabanlı Model

Yapay Sinir Ağları (Artificial Neural Networks) veya kısaca YSA insan beyninden esinlenerek geliştirilmiş, ağırlıklı bağlantılar aracılığı ile birbirine bağlanan işlem elemanlarından oluşan paralel ve dağıtılmış bilgi işleme yapılarıdır [35]. En önemli özelliği, deneyimlerden (tecrübe) yararlanarak öğrenebilmesidir. Yapay sinir ağları, insan beyninin özelliklerinden olan öğrenme yolu ile yeni bilgiler türetebilme, yeni bilgiler oluşturabilme ve keşfedebilme gibi yetenekleri herhangi bir yardım almadan otomatik olarak gerçekleştirmek amacı ile geliştirilmişlerdir. Yapay sinir ağları, öğrenmenin yanı sıra bilgiler arasında ilişkiler oluşturma yeteneğine de sahiptir [35].

Bir yapay sinir ağında, birbirleriyle bağlantılı sinir hücrelerinin yer aldığı girdi katmanı (input layer), çıktı katmanı (output layer) ve gizli katman (hidden layer) olmak üzere temelde üç katman bulunmaktadır. Girdi katmanı ilk katmandır ve dışarıdan gelen verilerin yapay sinir ağına alınmasını sağlar. Bu veriler istatistikte bağımsız değişkenlere karşılık gelmektedir. Girdi katmanı probleme etki eden parametrelerden oluşmaktadır ve girdi katmanındaki nöron sayısı parametre sayısına göre şekillenmektedir. Son katman çıktı katmanı olarak adlandırılır ve bilgilerin dışarıya iletilmesi işlevini görür. Çıktı değişkenleri, istatistikte bağımlı değişkenlere karşılık gelir [36]. Modeldeki diğer katman ise girdi katmanı ile çıktı katmanı arasında yer alır ve gizli katman olarak adlandırılır. Gizli katmanda bulunan nöronların dış ortamla bağlantıları yoktur. Yalnızca girdi katmanından gelen sinyalleri alırlar ve çıktı katmanına sinyal gönderirler. Gizli katman ve gizli

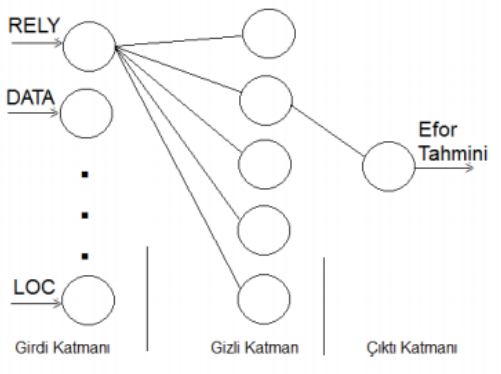
katmanlarda yer alacak nöronların sayısının seçimi, kurulan ağın performansı açısından önemlidir [37].

Yöntem, insan beyninin 'nöron' adı verilen birimlerin ağı şeklinde çalıştığı ile ilgili olan biyolojik bulgulardan esinlenmiştir. Tipik bir yapay sinir ağları modelinde girdi, gizli ve çıktı katmanları vardır. Şekil 5.1'de basit bir yapay sinir ağı modeli gösterilmiştir. Çıktı katmanı, girdi katmanından, aradaki gizli katmanlardan da geçerek, girdi ve sinyalleri alır. Gizli katman sayısı, uygulama alanına bağlı olarak değişiklik gösterir. Yapay sinir ağları, verimlilik tahmini, ses tanıma veya görüntü tanıma gibi verilen bir uygulamaya uygun olarak tasarlanırlar. Konu hakkında daha detaylı bilgi bu alanda yayınlanmış olan çok sayıda kitap ve çalışmadan elde edilebilir [35] [36].



Şekil 5.1 Örnek Yapay Sinir Ağları Modeli

Şekil 5.1'de temel bir yapay sinir ağları modeli görülmektedir. Oluşturulan yapay sinir ağında dört tane katmanın olduğunu görebilmekteyiz. İlk katman giriş değerlerini temsil ederken son katman ise çıktı değerlerini ifade etmektedir. Ara katmanlar ise gizli katmanları ifade eder. Gizli katman sayısının çokluğu oluşturulacak olan modelin karmaşıklığını gösterir. Ayrıca eğer veri çok fazla ise buna bağlı olarak oluşturulacak olan modelde gizli katman sayısı da fazla olabilir. Ama bu da yapılacak olan işlem sayısını arttırır.



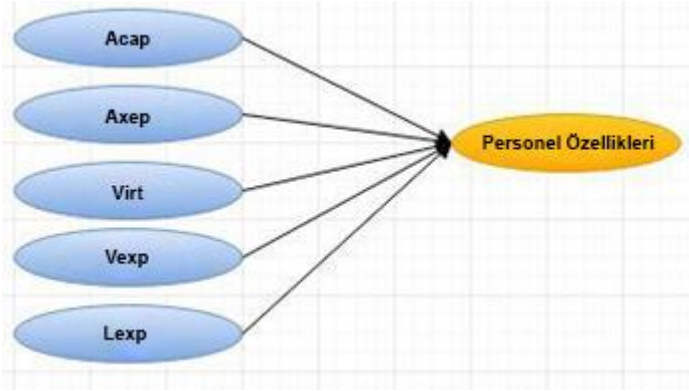
Şekil 5.2 Yazılım Efor Tahmini için Temel Yapay Sinir Ağları Modeli

Şekil 5.2'te Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri göz önünde bulundurularak oluşturulan yapay sinir ağı modeli gösterilmiştir. Burada veri setinde 16 tane giriş değeri bulunmaktadır. Çıktı katmanında yazılım efor değeri bulunmaktadır. Bu veri setlerinde örnek sayıları azdır. Bu nedenle bu şekilde oluşturulan yapay sinir ağı modeli istenen başarıyı ortaya koyamayacaktır. Bu modeli gerçekleştirmemizin ana nedeni yeni oluşturacağımız modelin bu modele göre başarısını ortaya koymaktır. Oluşturacağımız yeni model için veri setindeki giriş değerleri arasındaki ilişkiye bakarak giriş değerlerini (feature) gruplandırdık. Gruplandırma neticesinde veri seti özellik sayısı 5 adete indirgenmiştir. Kod satırı sayısı bu gruplandırmanın dışında tutulmuştur. Gruplandırma sonucunda elde edilen 4 tane giriş değeri ve kod satırı sayısı ile beraber 5 tane giriş değeri elde edilmiştir. Bu beş tane giriş değeri kullanılarak yapay sinir ağı modeli oluşturulmuştur.

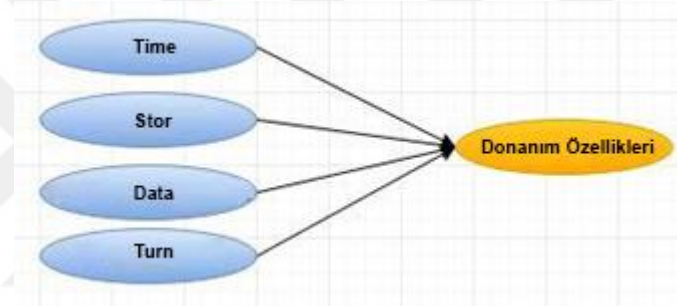
Bu tez çalışmasında veri setinin yeni versiyonu üzerinde YSA modeli geliştirildi. Bu model Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerini kullanarak yazılım efor tahmini gerçekleştirmemize yardımcı olmaktadır. Kullandığımız veri setleri temel olarak Cocomo modeli tarafından kullanılmıştır. Yazılım efor tahmininde 16 tane giriş değerimiz olmasına rağmen, bu 16 tane giriş değeri dört tane temel başlık altında sınıflandırılmıştır. Bu dört ana başlık ürün faktörleri, bilgisayar faktörleri, personel faktörleri ve proje faktörleri olarak ifade edilmiştir.

Şekil 5.4'te ve Şekil 5.5'te geliştirdiğimiz gruplandırma modeli görülmektedir. Oluşturulan gruplandırma modelleri temel 16 tane giriş değerini alarak kendi

içerisinde ayrı ayrı gruplandırır. Bu gruplandırma işlemi veri seti tarafından sunulan özelliklere bağlı olarak ortaya konur.



Şekil 5.3 Kişisel Özellikler için Gruplandırma

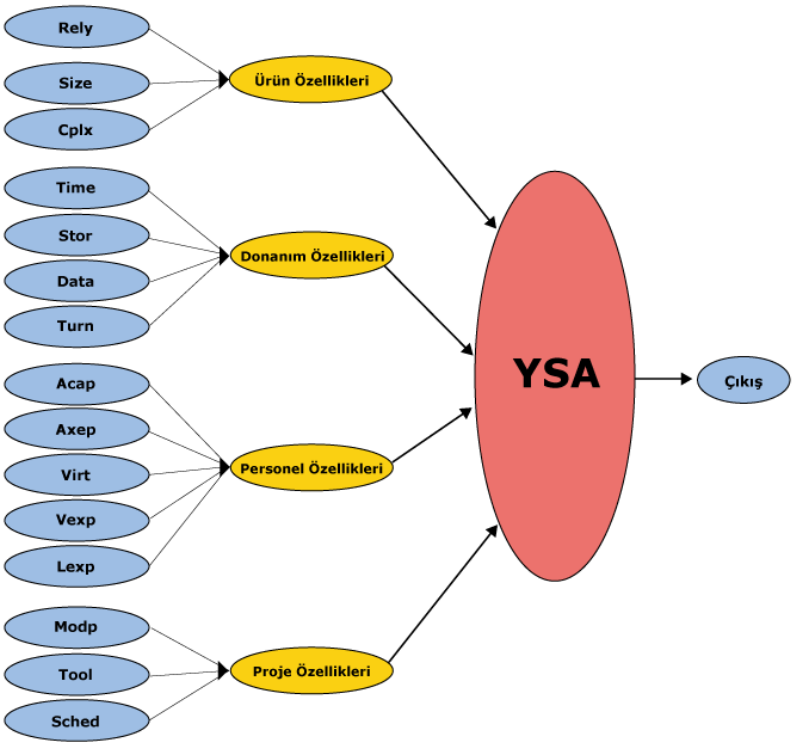


Şekil 5.4 Donanımsal Özellikler için Gruplandırma

Şekil 5.3'te ve Şekil 5.4'te ifade edilen gruplandırma işlemleri aynısı Ürün özellikleri ve Proje özellikleri için de ayrı ayrı oluşturulur. Şekil 5.5'te bu gruplandırma Ürün özellikleri ve Proje özellikleri için yapılan gruplandırma da görülmektedir. Bu nedenle ayrıca gösterilmemiştir.

Bu çalışmada oluşturulan yapay sinir ağları modelinde gözetimli öğrenme modeli (supervised learning) kullanılmıştır. Burada ortaya çıkan sonuç ile gerçek değer arasındaki fark hata olarak tanımlanmaktadır. Bu Model'de ana amacımız ortaya çıkan hata miktarını minimuma çekebilmektir. Bu nedenle bir tane gözetmene (supervisor-supervised learning) ihtiyacımız vardır. Bu gözetmen sayesinde ortaya çıkan hata miktarı sürekli gözlemlenir ve iyileştirilir. Bu iyileştirme işlemi iterasyonlar boyunca hata miktarının en az seviyeye kadar çekilip ve hatanın

değişmemesine kadar devam eder. Bu çalışmada geliştirilen yapay sinir ağları modeli regresyon tabanlı bir modeldir.



Şekil 5.5 Geliştirilen Modelin Ana Yapısı

Şekil 5.5'te oluşturulan modelin ayrıntılı yapısı gösterilmiştir. Geliştirilen yapay sinir ağları modelinde Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri kullanılmaktadır. Bu veri setleri rely, data, cplx, time, stor, virt, turn, acap, aexp, pcap, vexp, lexp, modp, tool, sced, equivphyskloc, act_effort gibi sütünlarin olduğu görülmektedir. Rely, data, cplx, time, stor, virt, turn, acap, aexp, pcap, vexp, lexp, modp, tool, sced, equivphyskloc giriş değerlerini oluşturur. Bu giriş değerleri veri seti kısmında ayrıntılı bir şekilde açıklanmıştır. Şekil 5.3, Şekil 5.4 ve Şekil 5.5'te ifade edilen giriş değerlerinin nasıl gruplandığı gösterilmiştir. Bu gruplandırma sonucunda grup çıkış değerlerini oluşturan değerler oluşan yapay sinir ağları modeli için giriş değerlerini oluşturmuştur. Bu giriş değerlerine bağlı olarak ortaya konan yapay sinir ağları modeli yazılım maliyet tahminini ortaya koymaktadır. Yukarıda ifade edilen özelliklerin gruplandırdıktan sonra veri setinin yeni halini oluşturmamız gerekti. Bu bağlamda Cocomo81 veri setini özelleştirdik. Bu gruplandırma sonucunda elde ettiğimiz özelliklerin değerlerini maliyete olan

etkisine bağlı olarak ilgili gruba attık. Böylece veri setinin yeni formunu elde etmiş olduk. Tablo 5.1'de model oluşturmak için kullanacağımız veri seti gösterilmektedir. Bu veri seti oluşturulurken daha öncede ifade ettiğimiz gibi özellikler gruplandırıldı. Her bir grubun değerini bulmak için o grupta bulunan özelliklerin maliyet ile olan ilişkisine bakıldı. Buna göre o gruptaki özelliklerin değerleri çarpılarak grubun değeri bulundu.

Tablo 5.1 Yapay Sinir Ağı için Oluşturulan Yeni Veri Seti

Ürün Özellikleri	Donanımsal Özellikler	Proje Özellikleri	Personel Özellikleri	Kod Satiri Sayısı	Gerçek Efor
1,24	1,081	0,75	0,86	25,9	117,6
1,24	1,081	0,75	0,86	24,6	117,6
1,24	1,081	0,75	0,86	7,7	31,2
1,24	1,081	0,75	0,86	8,2	36
1,24	1,081	0,75	0,86	9,7	25,2
1,24	1,081	0,75	0,86	2,2	8,4
1,24	1,081	0,75	0,86	3,5	10,8
1,24	1,081	0,75	0,86	66,6	352,8
1,24	1,51	1,45	0,58	7,5	72
1,08	1	0,75	0,46	20	72
1,08	1	0,75	0,57	6	24
1,08	1	0,75	0,46	100	360
1,08	1	0,75	0,75	11,3	36
1,08	1	0,75	0,84	100	215
1,08	1	0,75	0,57	20	48
1,08	1	0,75	0,98	100	360
1,08	1	1,18	0,46	150	324

Veri setini oluşturduktan sonra. Bu veri seti üzerinde koşacak olan modeli oluşturduk. Bu modeli oluştururken öncelikle verinin %90'nını eğitim için ve %10'nunu ise test için ayırdık. İkinci adım olarak ise hata fonksiyonun tanımladık. Bundan sonra ise öğrenme (train) metodumuzu belirledik. Stochastic Gradient Descent (SGD) öğrenme metodu olarak belirlendi. Yapılan çalışmalarda regresyon

tabanlı YSA çözümlerinde SGD'nin tercih edildiği görülmektedir [61]. Bu aşamadan sonra iterasyon sayımızı belirledik. Burada ne kadar fazla iterasyon kullanırsak o kadar doğru sonuçlar elde ederiz. Burada amaç oluşturulan ağın eğitim kaybını ve doğrulama kaybını en düşük seviyeye çekmektir. İterayon sayısının çok fazla olması durumunda hesaplama kaynaklarını boşa tüketebilir ve ani azalan sonuçlar döndürülebilir. Bundan sonraki aşamada ise öğrenme oranını (Learning Rate) ayarlamamız gerekiyor. Öğrenme oranı ile ağırlıkların boyutlularının değişimlerini ölçeklendiriyoruz (scale). Eğer bu değer çok büyük olursa ağ patlayabilir ve veriye uymayabilir. Genel de iyi başlangıç değeri olarak 0.1 seçilir. Eğer data çok uyumlu bir data değilse bu bağlamda bu oranı ile oynanabilir. Biz oluşturduğumuz model de öncelikle öğrenme oranı olarak 0.1 kullandık. Fakat daha sonra yaptığımız deneyler sonucunda en iyi sonucu 0.3 değeri ile ettik. Bu nedenle öğrenme oranı değerini 0.3'e ayarladık. Bundan sonraki adımda ise modelimizde kullanacağımız gizli katman sayısını (number of hidden layer) ayarladık. Tüm ağırlıkların optimize edildiği bir ortamda gizli katman sayısı ne kadar fazla olursa, modelin üreteceği tahminler de o kadar doğru olur. Tamamen optimize edilmiş bir model sıfır ağırlığa sahip olur. Fakat düğüm sayısı ne kadar fazla olursa ağırlıkları optimize etmek te o kadar zor olur. Ayrıca düğüm sayısının çok fazla olması durumunda "overfittinge" neden olur. Bu durumda model öğrenme yerine deseni ezberler. Bu da istenmeye sonuçlar verir. Oluşturduğumuz model 'de düğüm sayısını ilk olarak 15 belirledik. Fakat yapığımız deneyler sonucunda 10'da en iyi sonucu aldık. Veri setimizdeki örnek sayısı az oluğundan dolayı düğüm sayısını daha fazla arttıramazdık. Zaten yüksek bir değer seçtiğimizde model patlamaktadır. Yukarıdaki açıklamalar ışığında modelimiz en iyi sonucu iterasyon sayısı 10000, öğrenme oranı 0.3 ve düğüm sayısı 10 olduğunda vermektedir. Veri setindeki örneklerin az olmasından dolayı oluşturulan yapay sinir ağları modelleri tek gizli katmana sahiptir. Aşağıdaki Tablo 5.2'de sonuçlar listelenmiştir.

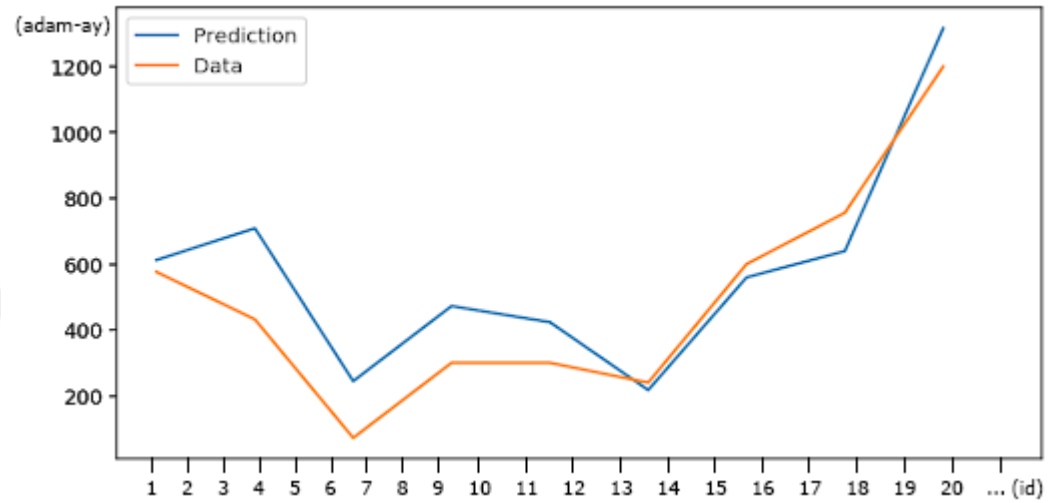
Tablo 5.2 Yapay Sinir Ağları ile Efor Tahmini

Gerçek Değer	Normal Veri Seti Üzerinde Oluşturulan YSA'nın Ürettiği Değerler	Normal Veri Seti Üzerinde Oluşan MARE Değeri	Yeni Veri Seti Üzerinde Oluşturulan YSA'nın Ürettiği Değerler	Gruplandırılmış Veri Seti Üzerinde Oluşan MARE Değeri
576	521,33	0,09	612,47	0,06
432	784,35	0,82	708,85	0,64
72	198,36	1,76	244,42	2,39
300	401,35	0,34	472,42	0,57
300	514,35	0,71	423,66	0,41
240	136,35	0,43	217,25	0,09
600	504,18	0,16	559,42	0,07
756	690,33	0,09	639,60	0,15
1200	1402,07	0,17	1316,26	0,10

Tablo 5.2'de geliştirilen yapay sinir ağları modellerinin sonuçları ortaya konmuştur. Normal veri seti üzerinde oluşturulan modelde kullanılan parametreler Tablo 4.1'de gösterilmiştir. Yeni veri seti üzerinde oluşturulan modelde kullanılan parametreler Tablo 5.1'de gösterilmiştir. Yeni veri seti üzerinde oluşturulan modelin başarısı sonuç kısmında ayrıntılı bir şekilde yorumlanacaktır. Tablo 5.2'de sonuçların bir kısmına ait sonuçlar ortaya konmuştur. Ayrıca oluşan MARE değerleri listelenmiştir. Geliştirilen modelin ortaya koyduğu sonuçlar ile gerçek değerler arasında yakın bir ilişki vardır. Normal veri seti üzerinde oluşan modelin MARE değeri 0.14'dur. Gruplandırılmış veri seti üzerinde geliştirilen YSA'nın MARE değeri ise 0.09781'dir. MARE değeri, gerçek değer ile tahmin edilen değerin farkının mutlak değerinin alınması ve çıkan sonucun gerçek değere bölünmesi elde edilmiştir. Buna "Magnitude of Reltaive Error- MARE" denir. Bu kısımda yapay sinir ağları yöntemi ile geliştirilen yazılım efor tahminin ürettiği sonuçlar listelenmiş ve oluşturulan yapay sinir ağı modeli açıklanmıştır. Yukarıda açıklandığı üzere verilerin gruplandırılması ile daha iyi MARE değeri elde edilmiştir. Buda gruplandırma çözümünün daha iyi sonuçlar ortaya koyduğunu göstermektedir.

5.1.1 Yapay Sinir Ağları Modelin Sonuçları

Geliştirilen yapay sinir ağları tabanlı modelde, Cocomonasa2 adlı veri kümesi kullanılmıştır. Bu veri kümesinde 69 tane projeye ait örnek bulunmaktadır. Aşağıdaki Şekil 5.6'da gerçekleştirdiğimiz modelin ürettiği tahmin değeri ile gerçek değer arasındaki ilişki gösterilmiştir.



Şekil 5.6 Yapay Sinir Ağları Tabanlı Modelin Başarı Grafiği

Nasa veri setiyle hazırlanan küçük ve orta ölçekli proje verileri ile gerçekleştirilen bu çalışma sonucunda, modelin ortalama %9.781 oranında sapma ile gerçeğe yakın tahmin yapabildiği görülmüştür. Bu sapma oran yapılan benzer çalışmalara göre oldukça düşük bir orandır. Örneğin Z. Zia, A. Rashid ve K. uz Zaman'ın 2009 yılında geliştirdiği modelle Vb.Net, Visual Basic ve Visual C# ile geliştirilen 19 proje için %11,61 MARE (Magnitude Absolute Relative Error) değeri elde edilebilmiştir [120]. Van Koten modeli bu projeler için uygulandığında ise %17,81 MARE değeri elde edilebilmiştir [121]. Modelin çekirdek veri kümesini oluşturan 15 projeden dış kaynak kullanılarak gerçeklesen 13 tanesinin farklı firmaların farklı ekipleri tarafından gerçekleşmiş olması, modelin geçerliliğine olumlu katkıda bulunmuştur. Bu modelin farklı ortamlarda ve farklı organizasyonlarda başarılı sonuçlar sunabilmesi için her ortam için ayrı kalibrasyonun yapılmasına ihtiyaç olduğu görülmüştür. Elde edilen sonuçların başarısının seçilen nesne sayılarının ve bunlara ait karmaşıklık derecelerinin doğru bir şekilde seçilmesi ile doğru orantılı olarak arttığı görülmüştür.

Bu tezde geliştirilen modellerden ilki yapay sinir ağları tabanlı modeldir. Sonuçlardan yapılabilecek çıkarımlara göre önerilen modelin dengeli olduğu, amaca hizmet edebileceği, karar destek aşamalarında birden çok çıktı ile yorum yapmanın daha sağlıklı olabileceği gösterilmiştir. Model, Nasa veri seti üzerine bina edilmiş ve YSA algoritması ile kullanılmıştır. Model sonuçlarının hem Cocomo hem de daha önce geliştirilen YSA' dan daha başarılı olduğu gözlemlenmiştir.

Tablo 5.3 Önerdiğimiz Yapay Sinir Ağları Tabanlı Model ile Yapay Sinir Ağları Tabanlı Modellerin Sonuçları

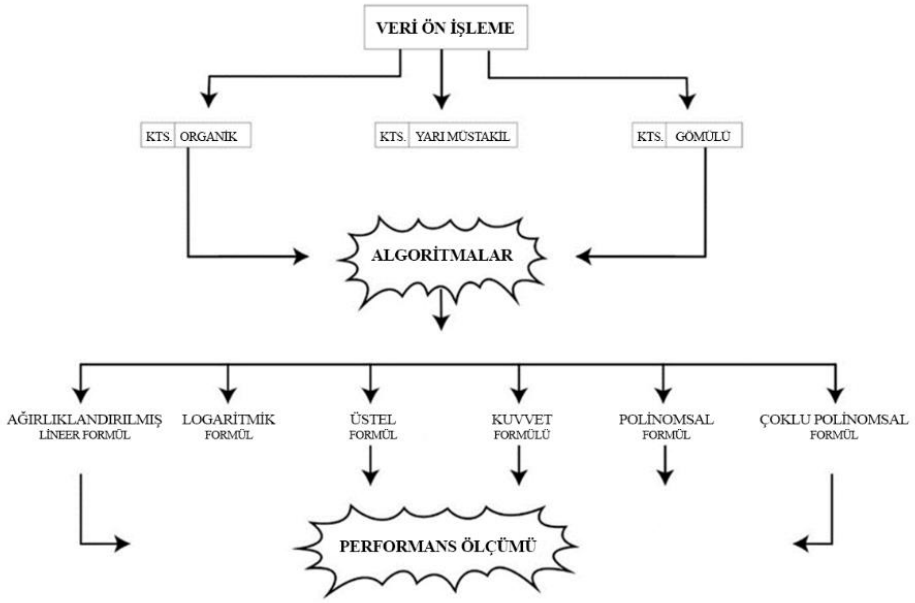
Yazarlar	Metot	Sonuç	Veri Seti
Muaz G. et al	YSA	MARE = %9,781	69 projeye ait- Semi - Detached- veri seti
Z. Zia et al	Multi-Layer Neural Network	MARE =%11,61	19 Projeye ait veri seti
Van Koten	Multi-Layer Neural Network	MARE =%17,81	15 Endüstri Projesi
Serluca [122]	Back-Propagation	MARE = %24	28 Proje- Mermaid-2
Srinivasan [123]	Back-Propagation	MARE = %30	Cocomo 81
Hughes [124]	Back-Propagation	MARE = %45	Hughes veri seti

Yukarıda Tablo 5.3'de önerdiğimiz modelin ve bu alanda geliştirilen modellerin başarıları listelenmiştir. Önerdiğimiz YSA modeli Tablo 5.3'de listelenen çalışmalardan daha başarılı sonuçlar ortaya koyduğu görülmektedir. Ayrıca verilerin gruplandırılması ile daha iyi sonuçlar ortaya çıkmıştır. Bu da özelliklerin doğru bir şekilde gruplandırıldığını göstermektedir. Önerdiğimiz model gruplandırma modelini kullanması ile bugüne kadar geliştirilen modellerden

ayrılmakta ve bugüne kadar gerçekleştirilen YSA tabanlı modellere göre daha iyi bir başarı ortaya koymaktadır.

5.2 Regresyon Tabanlı Yazılım Tahmin Model

Bu çalışmada en küçük kareler yöntemi tabanlı regresyon modeli geliştirilmiştir. Yazılım maliyet tahmininde kullanılan modellerin başında Cocomo modeli gelmektedir [25]. Cocomo modeli regresyon tabanlı bir modeldir. Cocomo modeli ile ilgili ayrıntılı bilgi giriş Kısımında anlatılmıştır. En küçük kareler yöntemi tabanlı geliştirdiğimiz regresyon modellerinin başarısını hata fonksiyonları ile ortaya koyduk ve Cocomo modeline göre başarısını gösterdik. Bu modeli gerçekleştirirken üç farklı veri setini kullandık. "Promise Repository" tarafından sunulan Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerini kullandık [25]. Bu veri setleri ile ilgili ayrıntılı bilgi veri seti bölümünde anlatılmıştır. Geliştirdiğimiz modelin genel yapısı sekil 5.7'da gösterilmiştir. Geliştirdiğimiz modeli S2ELSQ (Software Effort Estimation by Least Square) olarak ifade ettik.



Şekil 5.7 S2ELSQ Modelinin Genel Yapısı. "KTS." Katsayıyı ifade etmektedir

Bu modelin genel yapısını incelediğimizde üç farklı katmanın olduğunu görmekteyiz. İlk katman veri safhasından oluşmaktadır. Bu safhada veri seti seçimi yapılmaktadır. Ayrıca bu katmanda verilerin normalizasyonu ve iyileştirilmesi işlemleri gerçekleştirilmektedir. İkinci safhada ise en küçük kareler yöntemi tabanlı

regresyon çözümleri belirlenmektedir. Bu yöntemler en küçük kareler yöntemi tabanlı regresyon çözümü, en küçük kareler yöntemi tabanlı ağırlıklandırılmış doğrusal regresyon, en küçük kareler yöntemi tabanlı üstel regresyon, en küçük kareler yöntemi tabanlı exponansiyel regresyon, en küçük kareler yöntemi tabanlı polinomsal regresyon ve en küçük kareler yöntemi tabanlı çoklu polinomsal regresyon çözümleridir. İfade ettiğimiz yöntemler ayrı ayrı iyileştirilmiş olan Cocomo81, Cocomonasa ve Cocomonasa2 veri setleri üzerinde uygulandı. Ayrıca veri setinde üç farklı proje tipi bulunmaktadır. Bu proje tipleri yarı-müstakil, gömülü ve organik olarak ifade edilmiştir. Bu algoritmalar her bir proje türü için ayrı ayrı uygulanmıştır. Böylece daha ayrıntılı bir formülizasyon yapılmıştır. S2ELSQ modelinin üçüncü aşaması ise Cocomo algoritmalarının uygulanması aşamasıdır. Burada her bir proje turu için Cocomo yöntemi uygulanmıştır. Ayrıca geliştirilen modelin başarısı ile Cocomo yönteminin başarısı karşılaştırılmış olup sonuçlar kısmında gösterilmiştir.

Bu bölümde tezimizde kullandığımız en küçük kareler yöntemi kullanarak yapılan doğrusal ve doğrusal olmayan regresyon yöntemlerini ele alacağız. En küçük kareler yönteminin birçok farklı normları mevcuttur. En küçük Kareler yöntemi istatistikte kullanılan temel teoremlerden bir tanesidir. En küçük kareler yöntemi International Software Benchmarking Standard Groups (ISBSG) tarafından yazılım maliyeti tahmininde desen tanımlama (pattern recognition), makine öğrenmesi (machine learnings) [62], [63], sınıflandırma (classification) [64, [65], çoklu regresyon (multi-target regression) [66], [67], derin öğrenme (deep learning) [66], [68] ve özellik çıkartma uygulamalarında (feature selection application) [69], [70], [71] ve birçok çalışmada kullanılmıştır. Bu bölümde, çalışmamızda kullandığımız Ağırlıklandırılmış Doğrusal Regresyon, Logaritmik regresyon, Exponansiyel regresyon, Kuvvet regresyon, Polinomsal regresyon ve Çoklu Polinomsal regresyon metotlarını detaylı bir şekilde ele alacağız.

5.2.1 Ağırlıklandırılmış Doğrusal Regresyon Çözümü

Ağırlıklandırılmış Doğrusal Regresyon (Weighted Lineer Regression) Gauss tarafından lineer sistemleri çözmek için 1821-1823 yılları arasında ortaya konmuştur. Gauss bu yöntemi matrislerden türetilen bit yöntem kullanmıştır [72].

Ağırlıklandırılmış Lineer Regresyon (WLS) lineer ve non-lineer regresyonun aksine lineer ve non-lineer girişleri olan bir fonksiyon ile ilişkili değildir. Ağırlıklandırılmış Lineer Regresyon, hataların varyansının sabit olmadığı durumlarda kullanılır. Böylece hataların varyansı sabit olduğu bütün gözlemler ihmal edilir.

Bu yöntemin genel çözümü 2 numaralı denklemde gösterilmiştir. ε burada normal dağıtımlı ortalama varyans-kovarynas matrisi olarak ifade edilmektedir. X1 ve X2 ise bağımsız değişkenleri oluşturmaktadır. Y yazılım eforu değişkenini temsil ederken, X1 ise kod satırı sayısını temsil etmektedir. X2 ise maliyet çarpanlarını temsil etmektedir.

$$y = \times_1 \ \beta_1 \ + \times_2 \ \beta_2 \ + \ \varepsilon \tag{5.1}$$

Bu yöntem artık kareler toplamını en küçük yapmak için kullanılmıştır. W_i ve x_i birer vektördör.

$$RSS = a_0 + \sum_{n=1}^{n} (y_i - x_1 \,.\, \beta) \text{ x } (y_i - x_1 \,.\, \beta) \tag{5.2}$$

Bu yöntem ağırlıklı kareler yöntemini kullanır.

$$WSS(\beta, w) = a_0 + \sum_{n=1}^{n} w_i \text{ x } (y_i - x_1 \,.\, \beta) \text{ x } (y_i - x_1 \,.\, \beta) \tag{5.3}$$

Ağırlıklar, her bir varyansın (σi^2) tersi ($w_i = 1/\sigma i^2$) alınarak hesaplanır. Küçük hataların büyük varyanslar, büyük hataların ise düşük varyanslar oluşturduğu ve varyans ile ağırlıkların ters orantılı olduğu gözlemlenmiştir. Varyans ve kovaryans matrisi 5.4 numaralı formülde gösterilmiştir.

$$\begin{pmatrix} \sigma_1^2 & 0 & K & 0 \\ 0 & \sigma_2^2 & \Delta & 0 \\ M & M & O & M \\ 0 & 0 & K & \sigma_n^2 \end{pmatrix} \tag{5.4}$$

Ağırlık matrisi 5.5 numaralı formülle gösterilmiştir.

$$w = \begin{pmatrix} w_1 & 0 & K & 0 \\ 0 & w_2 & \Delta & 0 \\ M & M & O & M \\ 0 & 0 & K & w_n \end{pmatrix} \quad (5.5)$$

WLS yöntemi 5.3 numaralı denklemi baz alarak tahmin yapar.

$$\beta_{wls} = (X^T W X)^{-1} X^T W Y \quad (5.6)$$

WLS metodu küçük veri setlerinde iyi sonuçlar veren bir yöntemdir. Bu çalışmada kullanılan Cocomonasa2 veri seti küçük bir veri seti olduğundan dolayı bu yöntem tercih edilmiştir. Bu yöntemin önemli dezavantajlarından bir tanesi ağırlıkları tam olarak biliniyor kabulüne dayanması [73]. Bazı özel durumlarda ağırlıklar daha önce yapılmış çalışmaların önceki versiyonlarında elde edilen değerler baz alınarak ta yapılabilir [73]. Fakat gerçek uygulamalarda bu durum çok kabul edilebilir bir durum değildir. Ağrılıklar küçük değerlerden seçildiği zaman sonuçların iyi olmadığı tespit edilmiştir [73], [74]. Bu çalışmada kullanılan WLS ağırlıkları beta dağılımına dayanmaktadır [75]. Bu çalışmada geliştirilen S2ELSQ (software effort estimation by least square) olarak adlandırılmış olup ağırlıklandırılmış lineer regresyon için ürettiği sonuçlar Tablo 5.4'te gösterilmiştir.

Tablo 5.4 Ağırlıklandırılmış Lineer Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	WLS Sonucu	Hata Miktarı	MARE (Bağıl Hata Oranı)
Semi-detached	411,0	418,66	7,595	1,84
Semi-detached	349,7	390,39	40,68	11,63
Semi-detached	580,5	676,59	96,04	16,54
Semi-detached	622,5	715,45	92,95	14,92
Semi-detached	1.028	814,39	214	20,80
Semi-detached	187,6	213,72	26,08	13,9
Semi-detached	341,4	378,38	36,89	10,8
Embedded	895,2	941,97	46,74	5,22
Embedded	1.001	1.012,33	11,12	1,11
Embedded	691,9	741,71	50,12	7,24
Embedded	372,5	330,37	42,18	11,31
Organic	304,3	304,3	0	0
Organic	281,7	281,77	0	0

Yukarıda Tablo 5.4'te lineer regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir. Bu Tablode gerçek değerler, tahmin edilen değerler, hata miktarı ve bağıl hata oranı gösterilmiştir.

En küçük kareler tabanlı ağırlıklandırılmış regresyon modelinin genel çözümü ayrıntılı bir şekilde açıklanmıştır. Veri setimizdeki proje türlerine bakarak WLS (Weighted Least Square) çözümleri gerçekleştirildi.

Ağırlıklandırılmış regresyon çözümü için yukarıda listelenen 5.2, 5.3, 5.4,5.5 ve 5.6 denklemlerin çözümü ışığında 5.1 denkleminin formatında 5.7, 5.8 ve 5.9 numaralı denklemler elde edilmiştir. Bu denklemler her bir proje türüne özgüdür.5.7 numaralı denklem Embedded, 5.8 numaralı denklem Semidetached ve 5.9 numaralı denklem ise organic proje türü için elde edilmiştir.X1 değişkeni kod satır sayısını (Kilo Line Of Code) ifade etmektedir. X2 ise maliyet çarpanlarını temsil etmektedir. Y değişkeni ise yazılımın eforu ifade etmektedir.

$$y = 42.22328913\,x1 + 5.708242788\,x2 + 76.65483177 \quad (5.7)$$

$$y = -965.6101687\,x1 + 5.096894923\,x2 + 936.0800798 \quad (5.8)$$

$$y = -728.4874478\,x1 + 0.450835546\,x2 + 651.5453659 \quad (5.9)$$

Tablo 5.5 Ağırlıklandırılmış Regresyon Hata Miktarı

Proje Türü	Örnek Adedi	Artık Karelerin Toplamı
Embedded	21	5.644.279,70
Semidetached	69	4.080.253,69
Organic	3	1,38

5.7 numaralı denklem gömülü(embedded) proje türü için efor tahmini yapmaktadır. 5.8 numaralı denklem ise yarı müstakil (semi-detached) proje türü için efor tahmini yapabilmektedir.5.9 numaralı denklem ise organik (organic) proje türü için efor tahmini yapabilmektedir. En küçük kareler tabanlı ağırlıklandırılmış regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

Tablo 5.5'de veri seti üzerinde WLS uygulandığında her bir proje türü için toplam hata miktarı hesaplanmıştır. Tablo 5.4'de kullanılan veri seti Cocomonasa2 veri setidir.

5.2.2 Logaritmik Regresyon Çözümü

Logaritmik regresyon yöntemi (LLSM) denklemi çözmek için bir ağırlık vektörü ortaya koyar [76]. [77] ve [78] 'te logaritmik en küçük kareler yönteminin karakteristiği ortaya konulmuştur. LLMS (Logarithmic least square method) basit ve tek bir çözüm ortaya koyar [79]. Bugüne kadar LLMS üzerine birçok çalışma yapılmıştır. LLSM'deki ağırlık türevi, aksiyomatik yaklaşımla tam olarak ifade edilememektedir. Bu çözüm 5.10 numaralı formülde gösterilmiştir.

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \left[\log a_{ij} - \log \left(\frac{w_i}{w_j} \right) \right]^2 \tag{5.10}$$

Bu yöntem geometrik ortalamayı kullanır. Ağrılıklar ise 5.11 numaralı denkleme göre hesaplanır.

$$W_{\text{i}} = \frac{\sum_{j=1}^{n} a_{ij}^{1/n}}{\sum_{k=1}^{n} \prod_{j=1}^{n} a_{kj}^{1/n}} \tag{5.11}$$

LLSM yöntemi 5.12 numaralı denklemde ifade edilen şekli ile ortaya konmuş 5.13 ve 5.14 numaralı denklemlerde ise denklemin katsayılarını bulan çözümler ortaya konulmuştur.

$$y = a + bln\, x \tag{5.12}$$

$$b = \frac{n \sum_{j=1}^{n} (y_i \log_e x_i) - \sum_{j=1}^{n} y_i \sum_{i=1}^{n} \log_e x_i}{n \sum_{i=1}^{n} \log {x_i}^2 - (\sum_{i=1}^{n} \log x_i))^2} \tag{5.13}$$

$$a = \frac{\sum_{i=1}^{n} y_i - b \sum_{i=1}^{n} (\log x_i)}{n} \tag{5.14}$$

Bu çalışmamızda logaritmik en küçük kareler yöntemini kullandık. Bu yöntemi farklı veri setleri üzerinde test ettik. Bu yöntemdeki asıl amacımız 5.13 ve 5.14 denklemlerde ifade edilen katsayıların optimum değerini bulmaktır. Logaritmik en küçük kareler yöntemi bu çalışmada beklenen sonuçları ortaya koymuştur. Ayrıca Birçok farklı veri seti ile çalışmamızdan dolayı katsayıların optimum değerini bulma açısından daha iyi sonuçlar ortaya koyduğunu ifade edebiliriz.

Tablo 5.6 Logaritmik Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	LLSM Sonucu	Hata Miktarı	MMRE (Bağıl Hata Oranı)
Semi-detached	63,95	70,47	6,52	10,19
Semi-detached	52,17	76,08	23,91	45,83
Semi-detached	1.031,24	1.022,04	9,2	0,89
Semi-detached	1.028,39	1.045,84	17,45	1,69
Semi-detached	806,31	851,49	45,18	5,60
Semi-detached	1.209,5	1.192,06	17,44	1,44
Embedded	784,43	862,97	78,54	10,01
Embedded	372,53	404,19	31,66	8,49
Embedded	1.882,57	1.552,76	329,81	17,51
Organic	304,313	338,87	34,55	11,35
Organic	281,77	262,85	18,92	6,71
Organic	446,46	430,81	15,64	3,50

Yukarıda Tablo 5.6'da Logaritmik regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir. Veri setimizdeki proje türlerine göre LLSM çözümleri gerçekleştirildi. Logaritmik regresyon çözümü için yukarıda listelenen 5.10, 511, 5.13 ve 5.14 denklemlerin çözümü ışığında 5.12 denkleminin formatında 5.15, 5.16 ve 5.17 numaralı denklemler elde edilmiştir. Bu denklemler her bir proje türüne özgüdür.5.15 numaralı denklem Embedded, 5.16 numaralı denklem Semidetached ve 5.17 numaralı denklem ise organic proje türü için elde edilmiştir. X değişkeni kod satır sayısını ifade etmektedir. Y değişkeni ise eforu ifade etmektedir.

$$y = 421.2857206\, ln(x) - 944.7596407 \quad (5.15)$$

$$y = 363.9290443\, ln(x) - 895.3684452 \quad (5.16)$$

$$y = 31.343992\, ln(x) + 165.130592 \quad (5.17)$$

Tablo 5.7 Logaritmik Regresyon Hata Miktarı

Proje Türü	Örnek Adedi	Artık Karelerin Toplamı
Embedded	21	6159565.33
Semidetached	69	18802988.22
Organic	3	5.19

Tablo 5.7'de veri seti üzerinde LLSM uygulandığında her bir proje türü için hata miktarı hesaplanmıştır. Tablo 5.6'da kullanılan veri seti Cocomonasa2 veri setidir.

5.15 numaralı denklem embedded proje türü için efor tahmini yapmaktadır. 5.16 numaralı denklem ise semi-detached proje türü için efor tahmini yapabilmektedir. 5.17 numaralı denklem ise organik proje türü için efor tahmini yapabilmektedir. En küçük kareler tabanlı logaritmik regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

5.2.3 Üstel Regresyon Çözümü

Üstel en küçük kareler yöntemi geliştirilmesinde Guggenheim metodu kullanılmıştır [80]. Guggenheim metodunu uyguladığımızda 5.19 ve 5.20 numaralı formüllerde ifade edilen denklemler ortaya konmaktadır. Bu denklemler tekrardan düzenlendiğinde 5.21 ve 5.22 numaralı denklemler elde edilmektedir. Böylece a1 ve a2 ara değerleri elde edilmiştir. Denklemin diğer bir katsayısı olan b'nin değeri elde edilememiştir. 5.21 ve 5.22 numaralı denklemlerdeki a1 ve a2 değerlerini birbirine yaklaştırarak ve b'ye ilk değer verilerek hesaplanır [81].

Bu yöntemin genel denklemi 5.18 numaralı formülde ortaya konmuştur. Bütün bu denklemleri çözdüğümüzde ise 5.24 ve 5.25 numaralı denklemler ortaya çıkmaktadır. Bu denklemler ise ifade ettiğimiz genel denklemin katsayılarını vermektedir.

$$y = ae^{bx} \tag{5.18}$$

$$\frac{\partial X^2}{\partial a} = 0: \sum y_i e^{bx_i} = a \sum e^{2bx_i} \tag{5.19}$$

$$\frac{\partial X^2}{\partial b} = 0: \sum x_i y_i e^{bx_i} = a \sum x_i e^{2bx_i} \tag{5.20}$$

$$a_1 = \frac{\sum y_i e^{bx_i}}{\sum e^{2bx_i}} \tag{5.21}$$

$$a_2 = \frac{\sum x_i y_i e^{bx_i}}{\sum x_i e^{2bx_i}} \tag{5.22}$$

$$lny = lna + bx \tag{5.23}$$

$$a = \frac{\sum_{i=1}^{n} lny_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i lny_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{5.24}$$

$$b = \frac{n\sum_{i=1}^{n} x_i lny_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{5.25}$$

Bu çalışmada üstel en küçük kareler yöntemini kullanıldı. Bu yöntemi de birçok farklı veri seti üzerinde test ettik. Önemli olan katsayıların optimum değerini bulmaktır. Yukarıda açıkladığımız şekli ile katsayıların optimum değerini bulduk. Ayrıntılı bir şekilde açıkladık. Farklı proje türleri için farklı denklemler bulduk. Bütün denklemlerin katsayılarını bulduk.

Tablo 5.8 Üstel Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	Üstel Regresyon Sonucu	Hata Miktarı	MMRE (Bağıl Hata Oranı)
Semi-detached	1.209,5	1.212,19	2,69	0,22
Semi-detached	570,66	586,96	16,3	2,85
Semi-detached	572,31	563,39	8,92	1,55
Semi-detached	656,02	611,5	44,52	6,78
Semi-detached	806,31	634,71	171,6	21,28
Embedded	2.390,77	1.949,84	440,9	18,44
Embedded	895,23	918,34	23,11	2,58
Embedded	1.001,21	987,93	13,28	1,32
Embedded	691,59	745,96	54,37	7,86
Organic	304,31	311,42	7,107	2,33
Organic	281,77	276,55	5,22	1,85
Organic	446,46	444,70	1,754	0,39

Yukarıda Tablo 5.8'de Üstel regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir. Veri setindeki proje türlerine bakılarak ELS (Exponantial Least Square) çözümleri gerçekleştirildi.

Üstel regresyon çözümü için yukarıda listelenen 5.19, 5.20, 5.21, 5.22, 5.23, 5.24 ve 5.25 denklemlerin çözümü ışığında 5.18 denkleminin formatında 5.26, 5.27 ve 5.28 numaralı denklemler elde edilmiştir. Bu denklemler her bir proje türüne özgüdür. 5.26 numaralı denklem Embedded, 5.27 numaralı denklem Semidetached ve 5.28 numaralı denklem ise organic proje türü için elde edilmiştir. X değişkeni kod satır sayısını ifade etmektedir. Y değişkeni ise eforu ifade etmektedir.

$$y = 407.6178329\, e^{5.736624223*10-3\, x} \quad (5.26)$$

$$y = 383.138099\, e^{2.980546559*10-3\, x} \quad (5.27)$$

$$y = 143.9983436\, e^{1.206794274*10-3\, x} \quad (5.28)$$

Tablo 5.9 Üstel Regresyon Toplam Hata Miktarı

Proje Türü	Örnek Adedi	Artık Karelerin Toplamı
Embedded	21	9.980.572,89
Semidetached	69	17.469.740,29
Organic	3	3,58

Tablo 5.9'da veri seti üzerinde ELS uygulandığında her bir proje türü için hata miktarı hesaplanmıştır. Tablo 5.8'de kullanılan veri seti Cocomonasa2 veri setidir.

5.26 numaralı denklem embedded proje türü için efor tahmini yapmaktadır.5.27 numaralı denklem ise semi-detached proje türü için efor tahmini yapabilmektedir. 5.28 numaralı denklem ise organik proje türü için efor tahmini yapabilmektedir. En küçük kareler tabanlı exponansiyel regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

5.2.4 Kuvvet Regresyon (Power Least Square Regression)

Kuvvet en küçük kareler yöntemi (PLSM-Power Least Square Methodu) doğrusal olmayan bir yöntemdir. PLSM yönteminin genel çözümü 5.29 numaralı denklemde ortaya konmuştur. Bu denklemin her iki tarafının doğal logaritmasını aldığımızda ise 5.30 numaralı denklem ortaya çıkmaktadır. Bu denklemi çözüldüğünde ise 5.31 ve 5.32 numaralı denklemler elde edilmiş olur [82]. Böylece denklemi ortaya koyan katsayılar elde edilmiş oldu.

$$y = ax^B \quad (5.29)$$

$$lny = lna + \beta lnx \quad (5.30)$$

$$a = \frac{\sum_{i=1}^{n}(lny_i) - b\ \sum_{i=1}^{n}(lnx_i)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \quad (5.31)$$

$$b = \frac{b\sum_{i=1}^{n}(lnx_i lny_i) - \sum_{i=1}^{n}(lnx_i)\ \sum_{i=1}^{n}(y_i)}{n\sum_{i=1}^{n}(lnx_i)^2 - \left(\sum_{i=1}^{n} lnx_i\right)^2} \quad (5.32)$$

Bu çalışmada PLSM yöntemini kullandık. S2ELSQ çözümü üç farklı veri seti üzerinde işlem yapmaktadır. Bu yöntemde yine önemli olan denklemi oluşturan katsayıların optimum değerini bulmaktır. Kullandığımız veri setleri ile katsayıların optimum değerini bulmaya çalıştık. Farklı proje türleri için (semi-detached, organic and embedded) farklı denklemler ortaya koyduk.

Tablo 5.10 Kuvvet Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	Kuvvet Regresyon Sonucu	Hata Miktarı	MMRE (Bağıl Hata Oranı)
Semi-detached	133,86	128,56	5,30	3,96
Semi-detached	40,97	43,31	2,34	5,71
Semi-detached	448,78	461,38	12,6	2,80
Semi-detached	136,59	145,4	8,81	6,44
Semi-detached	90,09	76,68	13,41	14,88
Embedded	2.390	2.377,43	12,57	0,52
Embedded	895	942,48	47,48	5,30
Embedded	257	259,91	2,91	1,13
Organic	304,31	331,52	27,20	8,94
Organic	281,77	267,978	13,79	4,89
Organic	446,46	428,84	17,62	3,94

Yukarıda Tablo 5.10'da Kuvvet regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir.

Veri setimizdeki proje türlerine bakarak PLSM çözümleri gerçekleştirildi.

Kuvvet regresyon çözümü için yukarıda listelenen 5.30, 5.31 ve 5.32 denklemlerin çözümü ışığında 5.29 denkleminin formatında 5.33, 5.34 ve 5.35 numaralı denklemler elde edilmiştir. Bu denklemler her bir proje türüne özgüdür. 5.33 numaralı denklem Embedded, 5.34 numaralı denklem Semidetached ve 5.35 numaralı denklem ise organic proje türü için elde edilmiştir. X değişkeni kod satır sayısı olarak ifade edilmektedir. Y değişkeni ise eforu ifade etmektedir.

$$y = 5.385115503\, x^{1,053450845} \tag{5.33}$$

$$y = 5.118981183\, x^{9.558324474*10-1} \tag{5.34}$$

$$y = 193.5410828\, x^{1.013229003*10-1} \tag{5.35}$$

Tablo 5.11 Kuvvet Regresyon Hata Miktarı

Proje Türü	Örnek Adedi	Artık kareler toplamı
Embedded	21	6.707.457,02
Semidetached	69	5.657.623,04
Organic	3	1,58

Tablo 5.11'de veri seti üzerinde PLSM uygulandığında her bir proje türü için hata miktarı hesaplanmıştır. Tablo 5.10'da kullanılan veri seti Cocomonasa2 veri setidir.

5.33 numaralı denklem embedded proje türü için efor tahmini yapmaktadır. 5.34 numaralı denklem ise semi-detached proje türü için efor tahmini yapabilmektedir. 5.35 numaralı denklem ise organik proje türü için efor tahmini yapabilmektedir. En küçük kareler tabanlı üstel regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

5.2.5 Polinomsal ve Çoklu Polinomsal Regresyon (Polynomial and Multiple Polynomial Least Square Regression)

Çoklu polinomsal regresyon değişkenlerin ve derecelendirmenin bir den fazla olduğu durumlarda kullanılan mühendislik çözümüdür. Çoklu polinomsal regresyon metodolojinde en çok kullanılan metot en küçük artık kareler yöntemidir. Yazılım

maliyeti tahmini için bu modeli geliştirirken en küçük artık kareler yöntemi kullanıldı. Bu yöntemin 4 tane ana temel faktörü vardır [83].

a: Polinom Katsayısı

k: Polinom Derecesi

N: Kullanılan veri setinde bulunan örnek sayısı

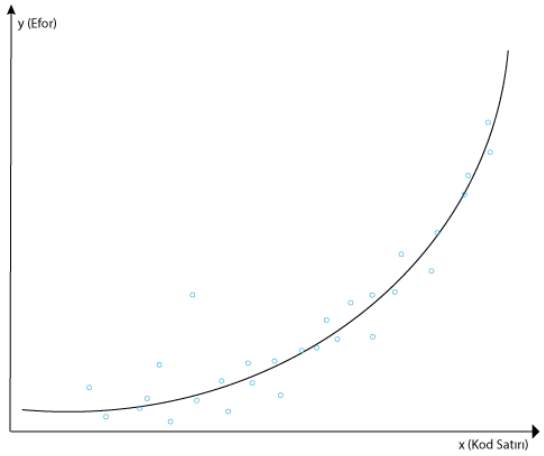
ε : Hata Miktarı

Bu model veri setini bir denklem vasıtası ile temsil etmek için kullanılır. Veri setlerini kth formunda ifade etmek istediğimizde aşağıdaki formda ifade ederiz.

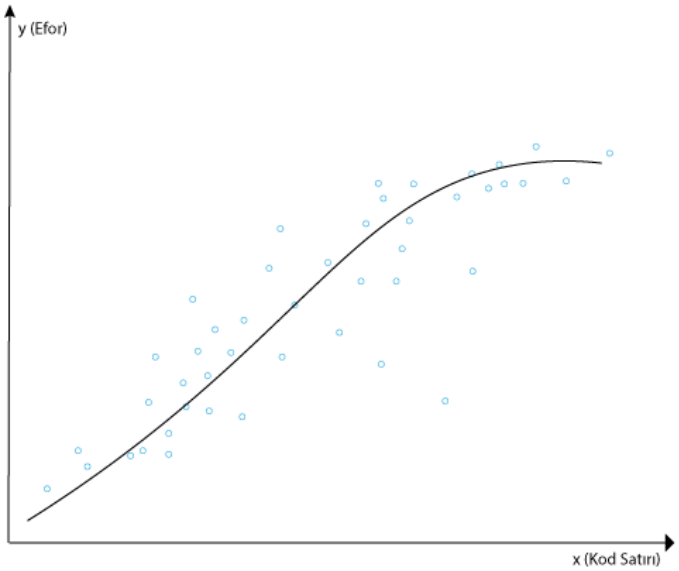
$$y = a_k x^k + a_{(k-1)} x^{(k-1)} + \cdots + a_1 x^1 + \epsilon \qquad (5.36)$$

Yukarıda 5.36 numaralı denklem genel polinom çözümünü göstermektedir. Burada elde edilen y değerine karşı gerçek y değeri arasındaki fark hata miktarın olan ε'yi oluşturur [84].

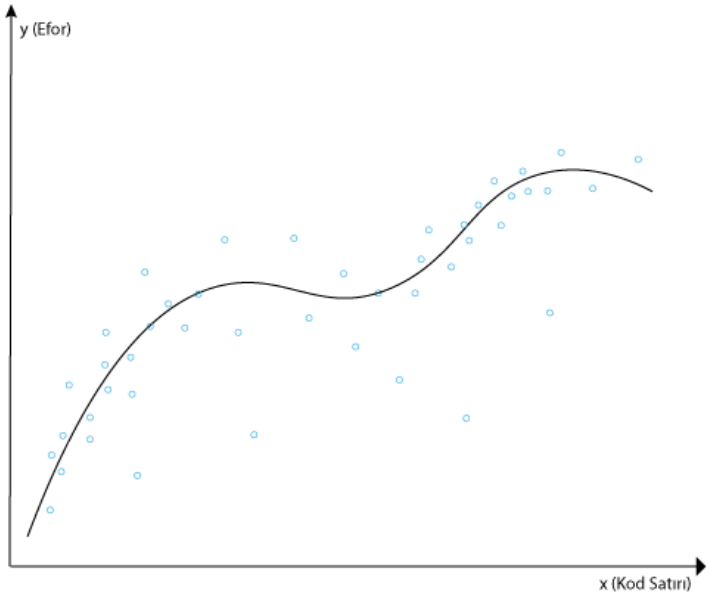
Bir polinom derecesi en çok k = n -1 olabilir. Fakat burada çoklu polinom denklemini çözebilmek için en az n x n adet veriye ihtiyaç vardır. Bu denklemin çözümü çok zor olduğu için genellikle düşük dereceden denklem çözümü yapılır (2 ve 3'uncu derece gibi). Aşağıdaki Şekil 5.7, Şekil 5.8 ve Şekil 5.9'da kullanılan Polinomsal regresyon çözümlerine ait basit 2'inci, 3'üncü ve 4'üncü dereceden grafikler sunulmuştur [85].



Şekil 5.8 İkinci Dereceden Polinom Grafiği



Şekil 5.9 Üçüncü Dereceden Polinom Grafiği



Şekil 5.10 Dördüncü Dereceden Polinom Grafiği

Şekil 5.7, Şekil 5.8 ve Şekil 5.9'da da açık bir şekilde görüldüğü gibi polinom derecesi yükseltildikçe veriyi daha iyi temsil eden bir grafiğin oluştuğunu görmekteyiz. Fakat burada asıl problem ise veriyi temsil eden grafiği bulabilmek ve ortaya çıkan karmaşık matematiksel denklemleri çözebilmektir.

Yukarıda tek değişkenli k'ninci dereceden denklem çözümlerine ait denklemler ve grafikler sunulmuştur. Aşağıda 5.37 numaralı denklemde ise k'ninci dereceden çok değişkenli bir denklemi çözmek için genel çözüm yolu verilmiştir. Burada kullanılan yöntem en küçük artık kareler yöntemidir. Bu çözümde asıl amaç gerçek değer ile tahmin edilen değer arasındaki farkı minimize etmektir.

Çoklu polinomsal regresyonun katsayılarını (ak, ak-1, ak-2 …) bir matris yardımı ile ifade edebiliriz. 5.37 numaralı denklemde katsayıları çözmek için bir matris çözümü sunulmuştur.

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i \cdots & \sum_{i=1}^{N} x_i^k \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} \cdots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=i}^{N} x_i\, y_i \\ \vdots \end{bmatrix} \tag{5.37}$$

Bu matrisi çözmek için farklı yöntemler mevcuttur. Bu yöntemlerin başında "cramer" kuralı ve "LU Decomposition" yöntemleri gelmektedir.

Tablo 5.12 Polinomsal Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	Polinomsal Regresyon Sonucu	Hata Miktarı	MMRE (Bağıl Hata Oranı)
Semi-detached	133,8	123,06	10,801	8,06
Semi-detached	35,51	31,56	3,95	11,1
Semi-detached	401,5	408,5	6,92	1,72
Semi-detached	136,5	150,75	14,16	10,3
Semi-detached	202,9	218,75	15,85	7,81
Embedded	307,4	309,71	2,27	0,73
Embedded	1.882	1.881,46	1,11	0,05
Embedded	1.001	1.061,76	60,55	6,04
Embedded	1.507	1.489,9	17,33	1,14
Organic	304,3	304,31	0	0
Organic	281,7	281,77	0	0
Organic	446,4	446,46	0	0

Yukarıda Tablo 5.12'de polinomsal regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir.

Tablo 5.13 Çoklu Polinomsal Regresyon Uygulama Sonuçları

Proje Türü	Gerçek Değer	Çoklu Polinomsal Regresyon Sonucu	Hata Miktarı	MMRE (Bağıl Hata Oranı)
Semi-detached	133,86	119,16	14,701	10,98
Semi-detached	136,59	127,57	9,02	6,603
Semi-detached	5.501,86	5.502	0,14	0,002
Semi-detached	57,09	58,14	1,05	1,839
Embedded	895,23	945,25	50,02	5,587
Embedded	135,51	134,28	1,23	0,907
Embedded	402,36	484,32	81,96	20,36
Organic	304,31	304,31	0	0
Organic	281,77	281,77	0	0
Organic	446,46	446,46	0	0

Yukarıda Tablo 5.13'de polinomsal regresyonun uygulandığı Cocomonasa2 veri setine ait sonuçlar kısmi olarak listelenmiştir. Veri setimizdeki proje türlerine bakarak POLS (Polynomial Least Square) çözümleri gerçekleştirildi.

5.38, 5.39 ve 5.40 numaralı denklemlerde X değişkeni kod satır sayısını ifade etmektedir. Y değişkeni ise eforu ifade etmektedir.

$$y = 4.15363366 * 10^{-18}x^{10} - 6.606886485 * 10^{-15}x^{9} + 4.399333307 * 10^{-12}\,x^{8} - 1.592175421 \cdot 10^{-9}\,x^{7} + 3.406086345 \cdot 10^{-7}\,x^{6} - 4.397387768 \cdot 10^{-5}x^{5} + 3.36611025 \cdot 10^{-3}\,x^{4} - 0.144125551\,x^{3} + 3.009353783\,x^{2} - 13.64035497\,x + 54.9946003 \quad (5.38)$$

$$y = -2.953658722 * 10^{-21}x^{10} + 4.430735525 * 10^{-18}x^{9} - 2.407727883 * 10^{-16}\,x^{8} - 2.636991482 \cdot 10^{-12}\,x^{7} + 1.920308353 \cdot 10^{-9}\,x^{6} - 6.54553747 \cdot 10^{-7}x^{5} + 1.25 \cdot 10^{-4}\,x^{4} - 1.348234233 \cdot 10^{-2}\,x^{3} + 7.143504597 \cdot 10^{-1}\,x^{2} - 9.236789382\,x + 69.09446239 \quad (5.39)$$

$$y = -(-1.139415047 \cdot 10^{-3}\,x^{2} + 5.989595021 \cdot 10^{-1}\,x + 259.6369007) \quad (5.40)$$

Tablo 5.14 Polinom Regresyon Hata Miktarı

Proje Türü	Örnek Adedi	Artık Kareler Toplamı
Embedded	21	236.466,61
Semi-detached	69	2.360.259,006
Organic	3	0

Tablo 5.14'de veri seti üzerinde POLS uygulandığında her bir proje türü için hata miktarı hesaplanmıştır. Tablo 5.13'de kullanılan veri seti Cocomonasa2 veri setidir. Burada Organik Proje türünde 0 hata ile çözümün gerçekleştiğini görmekteyiz. Hatanın 0 olmasının nedeni örnek sayısının az olmasıdır.

5.38 numaralı denklem embedded proje türü için efor tahmini yapmaktadır. 5.39 numaralı denklem ise semi-detached proje türü için efor tahmini yapabilmektedir. 5.40 numaralı denklem ise organik proje türü için efor tahmini yapabilmektedir. En küçük kareler tabanlı polinomsal regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

Veri setimizdeki proje türlerine bakarak M-POLS (Multiple Polynomial Least Square) çözümleri gerçekleştirildi

5.41, 5.42 numaralı denklemlerde X_1 değişkeni kod satır sayısını, X_2 değişkeni ise maliyet çarpan katsayısını ifade etmektedir. Y değişkeni ise eforu ifade etmektedir.

Bu bölümde yazılım maliyet tahmininde kullandığımız farklı veri setlerini, veri setlerinin özelliklerini ve veri seti üzerinde yapılan işlemleri açıkladık.

$$
\begin{aligned}
& y \\
& = -211.663042\, x_1^5 - 93.68821115\, x_1^4\, x_2 \\
& - 3.371731875\, x_1^3\, x_2^2 - 1.471154132 \cdot 1 - 2\, x_1^2\, x_2^3 + 5.662008064 \cdot 10 \\
& - 5\, x_1\, x_2^4 + 4.626188194 \\
& \cdot 10^{-8}\, x_2^5 + 6356.509984\, x_1^4 + 1167.250834\, x_1^3\, x_2^2 \\
& + 22.50981588\, x_1^2\, x_2^2 + 3.012590306 \cdot 10^{-2}\, x_1\, x_2^3 - 6.548267882 \\
& \cdot 10^{-5}\, x_2^4 - 53449.7202 - 4808.434925\, x_1^2\, x_2 - 42.81586011\, x_1\, x_2^2 \\
& - 2.13733394 \cdot 10^{-2}\, x_2^3 + 184910.2815\, x_1^2 + 7705.898309\, x_1 x_2 \\
& + 25.64215322\, x_2^2 - 273446.5731\, x_1 - 4153.95322\, x_2 + 141427.5932
\end{aligned}
\tag{5.41}
$$

$$
\begin{aligned}
y \;=\; & 23119.06749\, x_1^6 \;+\; 862.5018255\, x_1^5\, x_2 \\
& + 4.41786926\, x14\, x22 \\
& - 2.866504159 \cdot 10^{-2}\, x_1^3\, x_2^3 \\
& + 3.390949033 \cdot 10^{-5}\, x_1^2\, x_2^4 \\
& + 3.897680767 \cdot 10^{-8}\, x_1\, x_2^5 \\
& - 3.972673082 \cdot 10^{-10}\, x_2^6 - \; 156564.3303\, x_1^5 \;-\; 4404.957733\, x_1^4\, x_2 \\
& \qquad - 7.01985427\, x_1^3\, x_2^2 \;+\; 6.564946639 \cdot 10^{-2}\, x_1^2\, x_2^3 \\
& - 1.029826913 \cdot 10^{-4}\, x_1\, x_2^4 \\
& + 2.713032325 \cdot 10^{-7}\, x_2^5 \;+\; 419503.2195\, x_1^4 \\
& + 7998.958729\, x_1 3\, x2 - \; 1.9370608\, x_1^2\, x_2^2 \;-\; 3.126829874 \\
& \qquad \cdot 10^{-2}\, x_1\, x_2^3 \;-\; 1.922486904 \\
& \qquad \cdot 10^{-5}\, x_2^4 \;-\; 561967.344\, x_1^3 \\
& - 6416.400581\, x_1^2\, x_2 \;+\; 4.95385961\, x_1 x_2^2 \\
& + 6.107468715 \cdot 10^{-3}\, x_2^3 \;+\; 392640.5752\, x_1^2 \\
& + 2285.162116\, x_1 x_2 - \; 1.226251993 x_2^2 \;-\; 134841.9876\, x_1 \\
& - 294.0712089\, x_2 \;+\; 17849.57894
\end{aligned}
\tag{5.42}
$$

Tablo 5.15 Çoklu Polinom Regresyon Hata Miktarı

Proje Türü	Örnek Sayısı	Artık Kareler Toplamı
Embedded	21	$7{,}96 \cdot 10^{-3}$
Semidetached	69	718.849,43
Organic	3	0

Tablo 5.15'de veri seti üzerinde M-POLS uygulanıp her bir proje türü için hata miktarı hesaplanmıştır. Tablo 5.13'de kullanılan veri seti Cocomonasa2 veri setidir. Burada Organik Proje türünde 0 hata ile çözümün gerçekleştiğini görmekteyiz. Hatanın 0 olmasının nedeni örnek sayısının az olmasıdır. Diğer Yanda embedded proje türünde hatanın 0 çok yakın olduğunu görmekteyiz. Ayni şekilde semi-detached proje türünde hatanın çok azaldığını görmekteyiz.

5.41 numaralı denklem embedded proje türü için efor tahmini yapmaktadır. 5.42 numaralı denklem ise semi-detached proje türü için efor tahmini yapabilmektedir. Organik proje türü için efor tahmini ise polinomsal regresyon ile aynı fonksiyona sahiptir. En küçük kareler tabanlı çoklu polinomsal regresyon modeli Cocomo81, Cocomonasa ve Cocomonasa2 veri setlerine uygulanmıştır. Her bir proje türü için 3 farklı formül bulunmuştur. Bu formüller içerisinden en iyi çözümü sunan denklem seçilmiştir.

Bu kısımda geliştirdiğimiz en küçük artan kareler tabanlı yöntemi kullanarak geliştirdiğimiz regresyonun modelinin uygulamasını açıkladık. Bu modelde kullandığımız algoritmaları izah ettik. Elde ettiğimiz denklemleri ve hata miktarlarını Tabloler vasıtası ile gösterdik. Bu konuda yapılan çalışmaların başarısı ve önerdiğimiz çözümün başarısı sonuçlar kısmında karşılaştırmalı ve ayrıntılı bir şekilde açıklanmıştır.

5.2.6 Regresyon Tabanlı Yazılım Tahmin Modelinin Sonuçları

Cocomonasa2 veri setinde Cocomo yöntemi ve geliştirdiğimiz S2ELSQ modeli uygulanmış ve tahminler elde edilmiştir. Veri seti 3 bölüme ayrılmıştır ve elde edilen sonuçlar birbiriyle karşılaştırılmıştır. Değerlendirme kriterleri için mutlak hata kullanılır. Tablo 5.16'da elde edilen sonuçları göstermektedir.

Tablo 5.16 Regresyon Tabanlı Modellerin Toplam Hata Miktarları

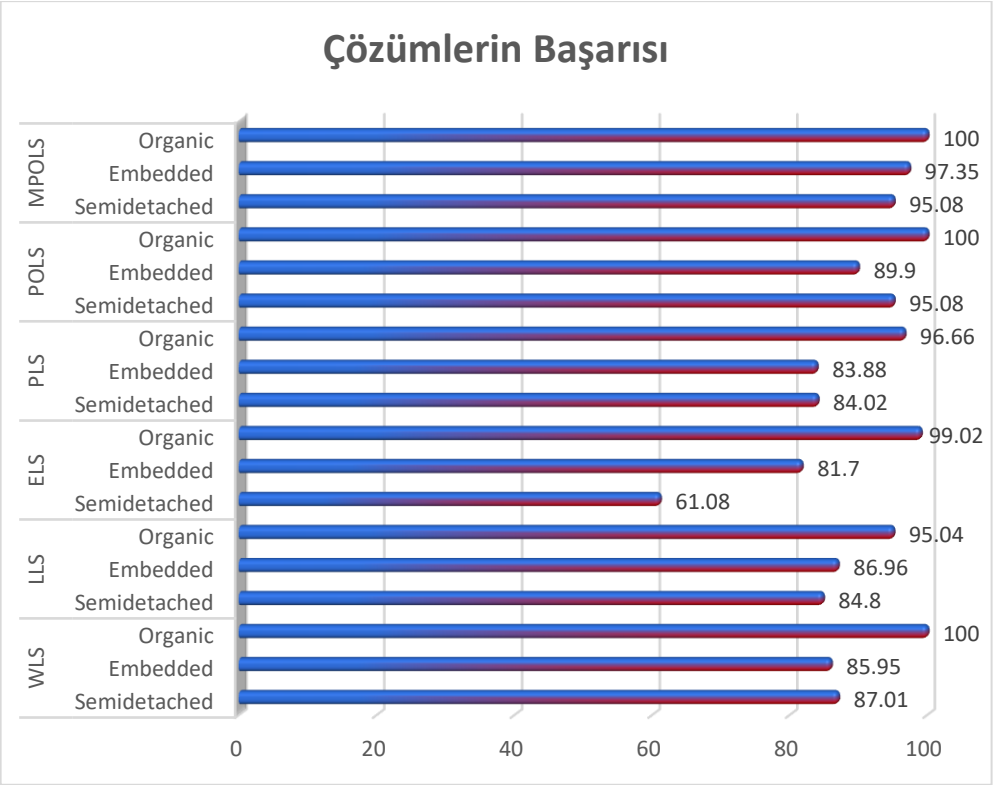
Geliştirme Modu	Cocomo Hata	Logaritmik Model Hata	Exponansiyel Model Hata	Kuvvet Model Hata	Polinomsal Model Hata	Çoklu Polinomsal Model Hata
Semi-Detached	9.300.688,	18.802.988	17.469.740	5.657.623	2.360.259	718.849
Embedded	70.364.49	6.159.565,3	9.980.572,9	6.707.457	236.466	0,007
Organic	332.855,83	5,20	3,59	1,58	0,00	0,00

Tablo 5.16'da bütün proje türleri için Cocomo'nun verdiği sonuca ek olarak geliştirdiğimiz modelin her bir algoritma için verdiği toplam hata miktarı gösterilmiştir.

Tablo 5.17 Geliştirlen Çözümde Algoritmaların Performansları

Çözüm	Proje Türü	Başarı Oranı (MARE)
Ağırlıklandırılmış En Küçük Kareler	Semi-detached	87.01
	Embedded	85.95
	Organic	100.0
Logaritmik En Küçük Kareler	Semi-detached	84.80
	Embedded	86.96
	Organic	95.04
Üstel En Küçük Kareler	Semi-detached	61.08
	Embedded	81.70
	Organic	99.02
Kuvvet En Küçük Kareler	Semi-detached	84.02
	Embedded	83.88
	Organic	96.66
Polinomsal En Küçük Kareler	Semi-detached	95.08
	Embedded	89.90
	Organic	100.0
Çoklu Polinomsal En Küçük Kareler	Semi-detached	95.08
	Embedded	97.35
	Organic	100.0
Cocomo	Semi-detached	80.28
	Embedded	82.16
	Organic	84.57

Şekil 5.11'de kullanılan algoritmaların sonuçları grafik yardımı ile gösterilmiştir. Çoklu polinomsal regresyonun başarısının diğer algoritmalara göre çok iyi olduğu açıkça görülmektedir.



Şekil 5.11 Önerilen Regresyon Tabanlı Modelin Başarı Grafiği

Geliştirilen yazılım çaba tahmini modelinin performansını ölçmede Cocomonasa2 veri kümesi kullanıldı. Mutlak bağıl hata, değerlendirme kriteri olarak kullanıldı. Sonuçlar, en küçük kareler yöntemlerinin performansının "Cocomonasa2" veri kümesi için Cocomo'dan daha iyi performans gösterdiğini göstermektedir. Ağırlıklandırılmış regresyon modeli, Logaritmik regresyon modeli, Üstel regresyon modeli, Kuvvet regresyon modeli, Polinom regresyon modeli ve Çoklu polinom regresyon modeli, Cocomo modelinden sırasıyla yüzde olarak daha iyi sonuçlar vermektedir. Bu başarıyı yukarıda şekiller ve grafikler yardımı ile gösterdik. Geliştirme modunun organik olduğu durumlarda Polinom modeli ve Çoklu Polinom modeli hata vermemektedir. S2ELSQ, çoklu polinom regresyon teknikleri kullanıldığında daha iyi sonuç verdiği açıkça görülmektedir. Çoklu polinom regresyon tekniği ve ağırlıklandırılmış regresyon tekniği birden fazla girdiye sahiptir. İlk giriş değeri kod satır sayısı ve ikinci giriş ise maliyet çarpan katsayısıdır. Bu katsayı, programcıların yeteneği, uygulama deneyimi, modern programlama uygulamaları, yazılım araçlarının kullanımı, sanal makine deneyimi, dil deneyimi, program kısıtlaması, ana bellek kısıtlaması, veri tabanı boyutu, CPU için zaman

kısıtlaması, geri dönüş süresi, sanal makine deneyimi, süreç karmaşıklığı ve gerekli yazılım güvenilirlik özellikleridir. Bu çalışmada, girdi arttıkça daha doğru tahminler yapabileceğimizi (Çaba Tahminini) gördük. İleri çalışmalarda her özelliği bir katsayı yerine girdi olarak alacağız. Ancak bu yöntemi geliştirmek için daha fazla veriye ihtiyacımız var. Önümüzdeki yıllarda istenilen verileri elde edebilirsek bu modeli daha da geliştirebiliriz. S2ELSQ şu an regresyon tekniklerine dayanan Cocomo ve regresyon tabanlı önerilen modellere göre daha iyi sonuç vermektedir. Tahminlerdeki hatayı azaltmak için daha fazla araştırma yapılması gerekmektedir.

Tablo 5.18 Önerdiğimiz Regresyon Tabanlı Model ile Regresyon Tabanlı Modellerin Sonuçlarının Karşılaştırılması

Yazarlar	Geliştirilen Model	Temel Alınan Algoritma	Başarı Oranı	Kullanılan Veri Seti
Liu [125]		İstatksel Regresyon Analizi(R)	61%	ISBSG
Stensrud [126]		İstatksel Regresyon Analizi(R)	34%	Laturi database ,48 completed
Hu [127].	MSCM	İstatksel Regresyon Analizi(R))	89%	Kamerer
V. Anandhi, R. Manicka Chezian [128]		Liner Regresyon	80%	Cocomonasa2 Veri Seti
Muaz G. et. al	S2ELSQ-Embedded	Polinomsal	89,9	Cocomonasa2 Veri Seti
Muaz G. et. al	S2ELSQ-Embedded	Çoklu Polinomsal	95,08	Cocomonasa2 Veri Seti
Muaz G. et. al	S2ELSQ-Semi Detached	Polinomsal	95,08	Cocomonasa2 Veri Seti
Muaz G. et. al	S2ELSQ-Semi Detached	Çoklu Polinomsal	97,35	Cocomonasa2 Veri Seti

Yukarıdaki Tablo 5.18'de ortaya koyduğumuz modelin diğer modellere karşı ürettiği değerler listelenmektedir. Geliştirdiğimiz çoklu polinomsal regresyon çözümü bugüne kadar yapılan çalışmalardan daha başarılı sonuçlar ortaya koymuştur.

5.3 Scrum Metodolojisi ile Geliştirilen Yazılımlar için Maliyet Tahmini

Yazılımların istenen kalitede, istenen özellikte, istenen zamanda ortaya konması ve belli bir plan dahilinde geliştirilmesi için Yazılım metodolojileri geliştirilmiştir. Yazılım geliştirme metodolojileri Yazılım maliyet tahminin önemli bir parçasıdır. Günümüzde geliştirilen yazılım tahmin modellerinin birçoğu Yazılım geliştirme surecinden bağımsız olarak çalışmaktadır. Bu çalışmada ise Scrum metodolojisi kullanılarak geliştirilen yazılımlar için Yazılım proje maliyet tahmini yapmaya çalıştık. Geliştirdiğimiz bu model regresyon tabanlı makine öğrenme algoritmalarını kullanmaktadır. Bu yeni yaklaşım ile Scrum metodolojisi ile yazılım geliştiren karar vericiler için güçlü bir destek maliyet tahmini ortaya koyduk. Özellikle son yıllarda Çevik metodolojisi kullanılarak yazılım geliştirilmektedir. Ayrıca bu model canlı ve çevik bir model olduğundan dolayı yazılım efor tahmini süreçle beraber tahmin edilmesi çok daha doğru sonuçlar ortaya koyacaktır. Bu modelde Scrum ile geliştirilen projelere ait veriler istenen şekilde düzenlenmiştir. Projeler 4 farklı aşamaya ayrılmıştır. Her bir asama için regresyon tabanlı makine öğrenme algoritmaları kullanılarak ayrı ayrı maliyet tahmini gerçekleştirildi. Çıkan sonuçlar birleştirilerek toplam maliyet tahmini ortaya kondu.

Şimdiye kadar yazılım karar vericileri için birçok yazılım efor tahmin modeli önerildi. Bu modeller, türlerine göre kategorize edilir. Örneğin, analojiye [86], [87], [88], [89] ve [90] dayanan birçok yazılım efor tahminleri çözümleri vardır. Bu tür tahminlerde proje düzeyinde yapılır. Ancak, iterasyon seviyesinde analojiye dayalı yazılım tahmini için başka bir yaklaşım vardır [91], [92] ve [93]. Bu çalışmaların zayıf noktası, maliyet tahmininin, yazılımın geliştirildiği ortamdan bağımsız olarak ele almasıdır. Cocomo ve Putnam modelleri [94], [95] gibi hesaplama yaklaşımlarına dayanan birçok yazılım efor tahmini vardır. Bu modeller, proje seviyesinde tahmin yapmaya hizmet eder ve kod satırına dayanır. Bu yöntemlerin zayıflıkları, geliştirildikleri ortamdan bağımsız olmaları ve tahmin edilmesi zor olan kod satırı tabanlı olmalarıdır.

Çevik yazılım geliştirmede kullanılabilecek efor tahmin tekniklerinden bazıları; Uzman Görüşü, Analoji, Ayrıştırma ve Planlama Poker'i çözümlerini kullanmaktadır [96]. Greening'in [97] tanıttığı Poker Planlaması, "Expert Opinion in Agile (Çevik yöntemlerde uzman fikri)", "Analogy (Analoji)" ve "Disaggregation (Disagregasyon)" öğelerini birleştiren bir çaba tahmini tekniğidir.

Evita Coelho ve arkadaşları [98]. Çevik yazılım geliştirme modelinde, istenen özellik, tahmini boyut, tahmini zaman Tablosi adımlarından oluşan bir yazılım efor tahmini modeli önermişlerdir.

Saurabh Bilgaiyan ve arkadaşları tarafından çevik tabanlı yazılım projelerinin maliyet ve eforunun tahmini için Genetik Algoritma (Genetic Algorithm -GA), Parçacık Sürü Optimizasyonu (Particle Swarm Optimization-PSO), Yapay Sinir Ağı, Bulanık Çıkarım Sistemleri (Fuzzy Inference System- FIS) başarıyla uygulanmaktadır [99].

Alaa ve arkadaşları, Düzeltilmiş Öykü Puanlarını (Adjusted Story Point-ASP) hesaplamak için Teknik Faktörleri (Technical Factor- TF) ve Çevresel Faktörler (Enviormental Facor -EF)' i göz önünde bulundurarak bir algoritmik model önermiş ve modeli çok katmanlı nöral ağ kullanarak uygulamışlardır [100].

Aditi Panda ve arkadaşları, yazılımın eforunu tahmin etmek için Genel Regresyon Sinir Ağı (General Regression Neural Network- GRNN), Olasılıksal Sinir Ağı (PNN) ve Kaskad Korelasyon Sinir Ağı (Cascaded Corelated Neural Network -CCNN) gibi farklı tipte sinir ağlarını kullanmışlardır. MSE (Means Square Error), R2-Squared (Kare Hata), MMRE (Mean Magnitude of Relative Error) ve PRED (Prediciton) tekniklerini kullanmışlardır. Sinir ağlarının performansını değerlendirmek için 21 projeden oluşan veri setini kullanmışlardır [101].

Suman ve arkadaşları, yazılım bakım eforunun hesaplanması için bir sezgisel yöntem önermişlerdir. Geliştirdikleri Modeli, Yazılım Bakım Çaba Tahmin Modeli (Software Maintenance Effort Estimation Model-SMEEM) olarak adlandırmışlardır. Bu teknik parametre olarak hikâye noktalarını alır ve bakım hacmini hesaplar [102].

Choetkiertikul ve arkadaşları, yazılım projelerini planlamak ve izlemek için derin öğrenme modeli geliştirmişlerdir. Hikâye noktası Tahmini derin öğrenme modeli

(Deep Learning Based Software Effort, Deep-SE), Uzun süreli kısa süreli bellek (Long Term and Short Term Memory- LSTM) ve tekrarlayan otoyol ağı (Recurrent Highway Networks- RHN) kombinasyonundan oluşmaktadır. LSTM, bir konunun metinsel açıklamasında uzun vadeli bağlamı modellememize izin verirken, RHN bize bu modelin derin bir temsilini sunmaktadır. Hikâye noktası efor tahmini için özelleştirilmiş yeni bir veri seti kullanılmıştır. Bu veri seti, 23.313 kullanıcı öyküsü ve 16 büyük çeşitli yazılım projesinden oluşmaktadır. Ortalama Mutlak Hata (Mean Absolute Error -MAE), efor tahmini modellerinin performansını ölçmek için kullandıkları yöntemlerden biridir. Önerilen yaklaşım, MAE değerini 16 projede ortalama %34.06 olarak saptamıştır. Bu önemli gelişme, bir sorunun metinsel açıklamasını modellemek için derin öğrenme LSTM mimarisinin kullanılmasından kaynaklanmaktadır [103].

Porru ve arkadaşları, SVR yöntemini kullanarak Story point tabanlı çevik yazılım geliştirme metodolojisi ile geliştirilen yazılımlar için efort tahmini yapmaya çalıştılar. Destek Vektör Makinesi (SVM), Naïve Bayes (NB), K-En Yakın Komşu (KNN) ve Karar Ağaçlarının (DT) performansını ölçtüler. SVM'nin diğerlerinden daha iyi performans gösterdiğini ifade ettiler [104].

Rashmi ve arkadaşları, hikâye noktalarına dayanan çevik ortamlarda algoritmik bir model önermişlerdir. Direnç, proje ve insan faktörlerini geliştirdikleri modellerinde dikkate almışlardır. Story Pointleri, Hız Faktörünü (Velocity Factor -VF) ve Karmaşıklık Faktörlerinin Ayarlanmamış Değerini Toplam Tahmini Efor (Totoal Effort Estimation-TEE) ve Toplam Tahmin Maliyetini (Total Effort Cost-TEC) hesaplamak için kullanmışlardır [105].

Christophe ve arkadaşları, efor hesaplamasında öykü noktaları ve COSMIC Fonksiyon Noktalarını (Function Point-) kullanmışlardır. Bu yöntemler daha iyi MMRE & R2 tahminleri sağlamıştır [106].

Satapathy ve arkadaşları, 21 projeden oluşan veri setini kullanarak çaba tahmini için SVR'nin farklı Kernel yöntemleriyle MMRE & PRED'i değerlendirmiş ve kıyaslamıştır. SVR tabanlı geliştirilen modelde, radyal tabanlı çekirdek (Radial Base Kernel- RBF) kullanıldığında diğer modellerden daha iyi performans gösterdiği tespit edilmiştir [107].

Hind ve arkadaşları, scrum metodoloji projelerinde hikâye noktalarını ayarlamak için bir algoritma tanıtmışlardır. Yazılımın toplam tamamlanma süresini hesaplamışlardır. Öncelik Faktörü (Priority Factor-PF), Hikâye Boyutu Faktörü (Story Point Factor -SF), Karmaşıklık Faktörü (Complexity Factor -CF) gibi hızlandırma faktörleri kullanılmıştır. Hız, Sürtünme Faktörleri (Friction Factor-FR) ve Dinamik Faktörler (Dynamic Factors-DF) kullanılarak model geliştirilmiştir [108].

Tanveer ve arkadaşları, çevik için efor tahmininin ve yapılan tahminlerin doğruluğunun uygulayıcıların uzmanlığına duyarlı olduğunu ve önyargıya eğilimli olduğunu vurgulamışlardır. Agile geliştirme ekipleri açısından tahmin sürecini araştırmak amacıyla, Alman çokuluslu bir yazılım şirketi olan SAP'de üç çevik geliştirme ekibi ile ve iki gözlemci ile bir görüşme gerçekleştirilmiş olup ayrıca bir araştırma yapılmıştır. Araştırmanın sonuçlarına bakıldığında geliştiricinin bilgisi ve deneyimi ile alt sistemlerdeki değişikliklerin karmaşıklığı yapılacak olan tahminleri doğrudan etkilediği görülmüştür [109].

Yves ve arkadaşları, gereksinimlerin üstesinden gelmek ve mantık şemasını oluşturmak için bütünleşik bir model önermişlerdir. Bu modelin bir diğer amacı da çevik süreci içindeki Kullanıcı Hikâyeleri'ni temsil etmektir [110].

Bu çalışmada Scrum'da geliştirilen yazılım için destek vektör regresyon, çok katmanlı perseptron, rastgele orman, gradyan artırma regresyon yöntemleri kullanılmıştır. Yukarıda Geçmiş çalışmalara ait bilgiler sunuldu. Bu çalışmalar dikkatle incelenildiğinde çalışmaların zayıf yönleri, maliyet tahminini parça parça ele almak yerine, bir bütün olarak maliyet tahminini yapmaya çalıştıkları gözlemlenmektedir.

Yazılım eforunu tahmin etmede farklı bir yaklaşım da yazılımın geliştirildiği metodolojiyi dikkate almaktır. Bu metodolojilerden biri Scrum metodolojisidir. Scrum metodolojisi ile geliştirilen yazılımların yazılım maliyetini belirlemek için Story Point değerleri kullanılır [111]. Scrum, sprintlerden oluşmaktadır ve sprintler yapılacak işleri içermektedir. Sprint'teki her bir iş Scrum takımı tarafından belirlenir ve takım yapılacak her iş parçasını Story Point kullanarak puanlar. Bu puanlama sistemi için belirli bir model yoktur. Bazı takımlar S, M, L gibi puanlama

kullanırlar, diğerleri ise 1,2,4,8 gibi bir aritmetik sistemi kullanır. Bu çalışmada aritmetik ve Fibonacci serileri kullanılmaktadır (0.5, 1, 2, 3, 5, 8, 13 ...). Story Point değerleri görev akışına göre düzeltilebilir. Bu bağlamda, Story Point yapılan işin büyüklüğünü ve harcanan çabayı ifade eder. Bu sebepten ötürü, Story Point değerinin ürün iş listesi hazırlandıktan sonra belirlenmesi projenin başarısı için şarttır. Bu bağlamda, bu çalışmada Story Point değerini tahmin etmek için farklı makine öğrenme algoritmaları kullanılmıştır.

Bu çalışmada kullanılan ilk yöntem SVR (Destek Vektör Regresyon) 'dir. Önerdiğimiz model, projeyi 4 farklı aşamaya ayırmıştır ve her adımda "özelleştirilmiş" veri kümesindeki maliyeti tahmin etmiştir. Bu alanda geliştirdiğimiz SVR çözümü, çekirdek fonksiyonu olarak “Radial Basis” fonksiyonunu kullanmaktadır.

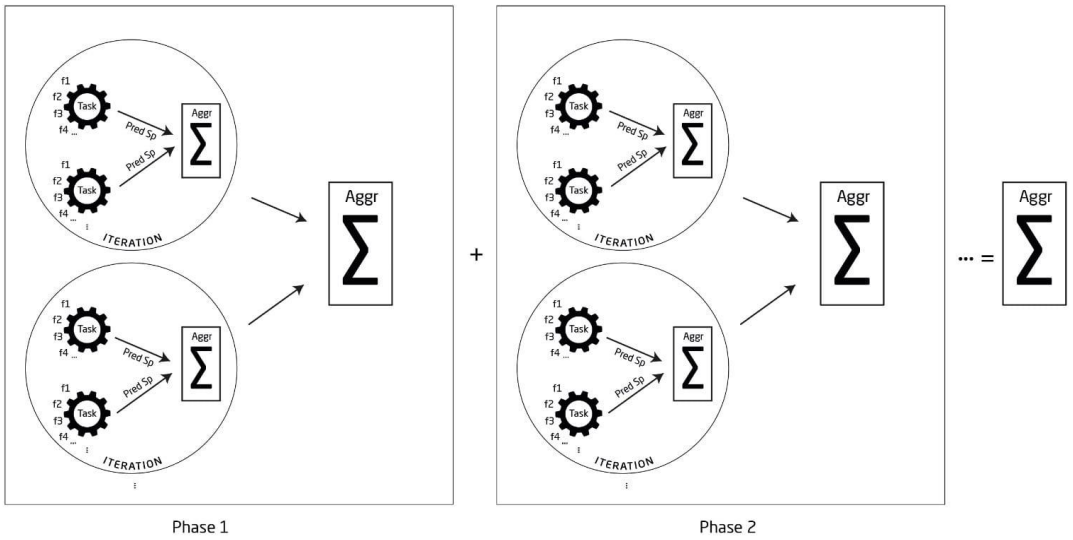
Bu çalışmada kullanılan ikinci yöntem ise Ratgele Orman Regresyon yöntemidir [112]. Scrum ile geliştirilen yazılımlarda Story Point değerlerini bulmak için ise Ratgele Orman Regresyon yöntemini kullandık.

Üçüncü olarak, Boosting Gradient Regression yöntemi (GBM) uygulanmıştır. GBM, farklı kayıp fonksiyonlarıyla öğrenme gibi, uygulamanın belirli ihtiyaçlarına göre özelleştirilebilir. GBM ağaçları üretir ve bunları topluluğa sırayla ekler. İlk ağaç, Rasgele Ormanlarda yapıldığı şekliyle üretildi. Temel fark, ilk ağacın ürettiği tahmin hatalarını en aza indirmeyi amaçlayan ikinci ağacın üretimidir. İlk ve ikinci ağaçlar topluluğa eklenir, ancak her birine farklı ağırlıklar verilir. Bu işlem birçok kez tekrarlanır. Her adımda, yeni bir ağaç öğrenilen toplumun hatalarına göre eğitilir ve daha sonra topluluğa eklenir. Topluluk, yeni girdilerin sonuçlarını tahmin etmek için bir model olarak kullanılmaktadır. Bizim çözümümüzde GBM algoritmaları mutlak kayıp fonksiyonunu kullanır.

Son yöntem Çok Katmanlı Perseptron'dur ve regresyon amacıyla kullanılır. Çok katmanlı perseptron yaklaşımımızda lojistik regresyon kullanılmıştır. RMSE (root mean square) kayıp fonksiyonu olarak kullanılmıştır [113] [114].

Scrum ile geliştirilen projelerde yazılım maliyeti geleneksel proje maliyet tahminlerinden farklıdır. Bu çalışmada öncelikli olarak veri seti tasarımı

gerçekleştirilmiştir. Veri seti dört farklı aşamadan oluşmaktadır. Veri setinden yapılacak bütün işler tek tek belirlenmiştir. Bu işler belirlendikten sonra bu işlere harcanacak efor tahmin edilecektir. Şekil 5.10'da önerilen çerçevenin(framework) genel yapısı gösterilmiştir.



Şekil 5.12 Story Point Tabanlı Maliyet Tahmini Modeli

Scrum metodolojisini kullanılarak geliştirilen yazılımlar için yazılım maliyet tahmini için önerilen model Şekil 5.12'de gösterilmiştir. Scrum yazılım geliştirme metodolojisi canlı ve müşterinin sürekli olarak işin içinde olduğu bir modeldir. Bu nedenle de değişim taleplerine açık bir modeldir. Bu bağlamda yazılım maliyetini baştan bir bütün olarak ele alabilmek için yapılacak olan işlerin çok iyi belirlenmesi gerekmektedir. Yapılacak olan işlerin iyi belirlenebilmesi için yazılım geliştirme sürecinde yapılacak olan işler 4 ayrı safhaya ayrılmıştır. Bir bütün olarak yazılım maliyeti tahmini yerine her bir safha için ayrı ayrı maliyet tahmini gerçekleştirilmiştir. Böylece daha doğru bir şekilde maliyet tahmini yapılması hedeflenmiştir. Bu model safhalardan oluşur. Safhalar da iterasyonlardan oluşur. İterasyonlar da iş parçalarından oluşur. Bir iterasyonda bulunan iş parçaları için Story Point tahmini yapılır. Bütün iterasyonlardaki bütün iş parçaları için Story Point tahmini yapılır. Bir iş parçası birden fazla iterayonda devam ediyor olabilir. Bu nedenle oluşturduğumuz özelleştirilmiş toplam fonksiyonu kullanıldı. Aynı şekilde her bir faz da birden fazla iterasyondan oluşmaktadır. Aynı şekilde oluşturulan

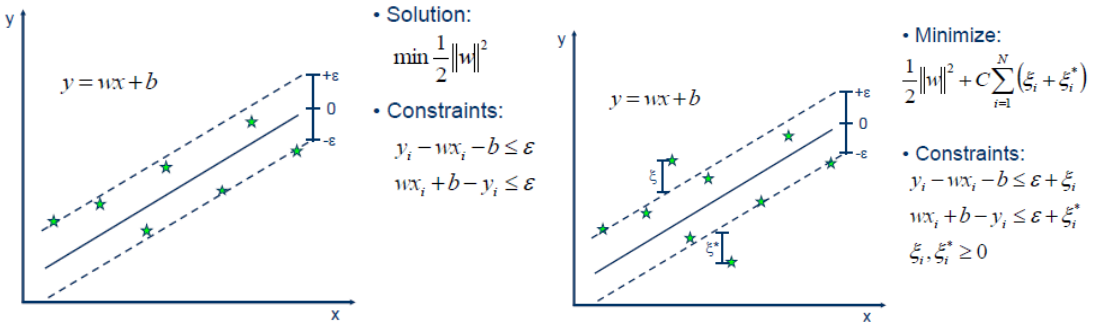
toplam fonksiyonu fazların değerinin tahmininde de kullanıldı. Daha sonra fazların değerleri toplanarak toplam maliyet elde edildi.

Scrum metodolojisinde yazılım efor tahmininde en önemli faktör Story Point tahminidir. Eforu belirleyen asıl faktör Story Pointlerdir. Bu nedenle eğer yapılacak olan her bir iş için Story Point değeri doğru bir şekilde tahmin edilebilirse yazılım maliyeti de doğru bir şekilde tahmin edilebilir. Bu bağlamda önemli olan faktörün Story Pointler olduğu açıktır. Burada önemli olan sınıf tabanlı öğrenme algoritmaları yerine regresyon tabanlı öğrenme algoritmalarını kullanabilmek. Bu nedenle Story Pointlerin değerini tahmin edebilmesi için regresyon tabanlı öğrenme algoritmaları kullanıldı. Yukarıda önerilen model öncelikle yapılacak olan işleri belirler. İkinci adımda ise yapılacak olan işler ilgili iterasyonlara atılır. Her bir iş parçası için makine öğrenme algoritmaları kullanılarak her bir işin Story Point değeri hesaplanır. Böylece her bir işin toplam Story Point değeri belirlenmiş olur. Bundan sonraki aşamada ise oluşturulan özelleştirilmiş toplam fonksiyonu ile iterasyonun toplam değeri bulunur ve aynı şekilde fazların değeri de hesaplanır. Son aşama olarak da bütün fazların Story Point değeri, oluşturulan toplam fonksiyonu ile birleştirilir. Böylece toplam Story Point değerine bağlı olarak efor değeri elde edilmiş olur. Bu model vasıtası ile Story Point tahmininde öznellik minimum seviyeye çekilip modelin ürettiği sonuçlar ile daha objektif tahminler yapılabilecektir.

5.3.1 Destek Vektör Regresyon (Support Vector Regression) Tabanlı Maliyet Tahmini

Destek vektör makinesi yöntemi, makine öğrenimi uygulamalarında yaygın olarak kullanılmaktadır. Destek vektör regresyonu kullanılarak yazılım maliyet tahmini gerçekleştirmeye çalışan çalışmalarda Destek vektör regresyonunun (SVR) farklı formlarını kullanılmıştır [105] [106]. Destek Vektör Regresyon küçük bir fark ile SVM (Support Vector Method) ile aynı özelliklere sahiptir. İki tip destek vektör regresyonu vardır, ilki lineer destek vektör regresyonudur. İkinci tip doğrusal olmayan regresyon. Regresyon söz konusu olduğunda, problemi önceden sormuş olan SVM'nin yakınlaştırılmasında bir tolerans sınırı (epsilon) kurulmuştur. Regresyonun ana fikri, gerçek değer ile ortaya konan değer arasındaki farkı en aza

indirmektir. Destek vektör regresyon aynı kuralı korur. Vektör regresyon desteği, hatayı en aza indirmeye çalışır. Ancak, ana fikir her zaman aynıdır: hatanın en üst düzeye çıkarıldığı hiper düzlemi bireyselleştirerek hatayı en aza indirgemek, hatanın bir kısmının tolere edildiğini göz önünde bulundurmak.



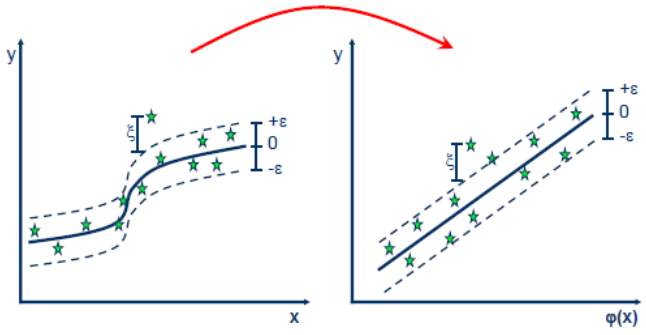
Şekil 5.13 Genel Lineer SVR

Destek vektör regresyon doğrusal yöntem Şekil 5.13'de gösterilmiştir.

$$y = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \cdot \langle x_i, x\rangle + b \tag{5.43}$$

Doğrusal Olmayan Destek vektör regresyonunda iki çekirdek fonksiyonu vardır. Çekirdek fonksiyonları sayesinde, verileri doğrusal ayırmanın gerçekleştirilmesini mümkün kılmak için verileri daha yüksek boyutlu bir özellik alanına dönüştürür.

$$y = \sum_{i=1}^{N}(\alpha\_i - \alpha\_i\hat{} *) \cdot \langle \varphi(x\_i), \varphi(x)\rangle + b \tag{5.44}$$



Şekil 5.14 Lineer Olmayan SVR Dönüşümü

İki çekirdek işlevi vardır. Birincisi polinom fonksiyonudur ve ikincisi gauss radyal temel fonksiyonudur.

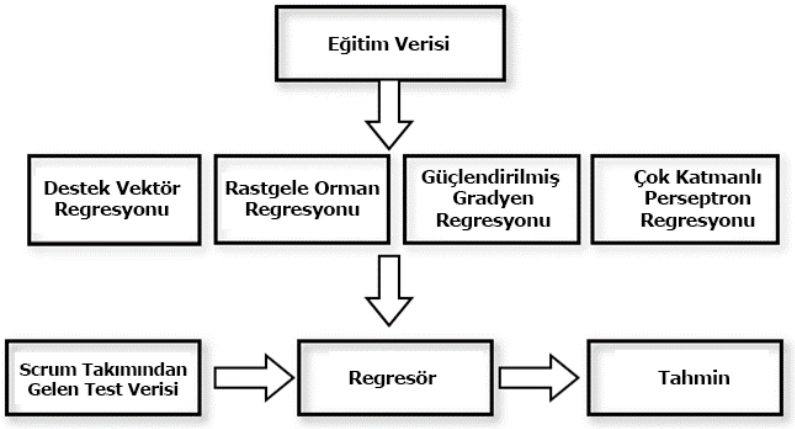
Polinom çekirdeği 5.45 numaralı formülde gösterilmiştir.

$$k(x_i, x_j) = (x_i \cdot x_j)^d \tag{5.45}$$

Gauss radyal temel fonksiyonu 5.46 numaralı formülde gösterilmiştir.

$$k(x_i, x_j) = exp\left[-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right] \tag{5.46}$$

Destek vektör regresyon yöntemi kullanılarak her bir fazda bulunan yapılacak olan her bir iş için Story Point tahmini yapıldı. Daha sonra spesifik toplam fonksiyonu kullanılarak her bir iterasyon için Story Point tahmini gerçekleştirildikten sonra toplam fonksiyonu kullanılarak toplam efor tahmin edilmiştir. Beş farklı veri seti üzerinde çalışılmıştır. Bu veri setleri JIRA, MongoDB, SPRING, JBOSS ve APACHE veri setleridir. Bu veri setlerinin her bir fazı için SVR (Support Vector Regression) kullanılarak ayrı ayrı tahmin gerçekleştirilmiştir.



Şekil 5.15 Destek Vektör Regresyon Tabanlı Model

Sekil 5.15'te SVR tabanlı önerilen model gösterilmiştir. Bu model de öncelikle yapılacak olan işler ilgili fazlara atılır. Daha sonra her bir iş için SVR kullanılarak efor tahmini gerçekleştirilmektedir. Bu her bir faz için tek tek yapılmaktadır. Oluşturulan toplam fonksiyonu sayesinde fazların değerleri birleştirilerek toplam efor elde edilmiş olur.

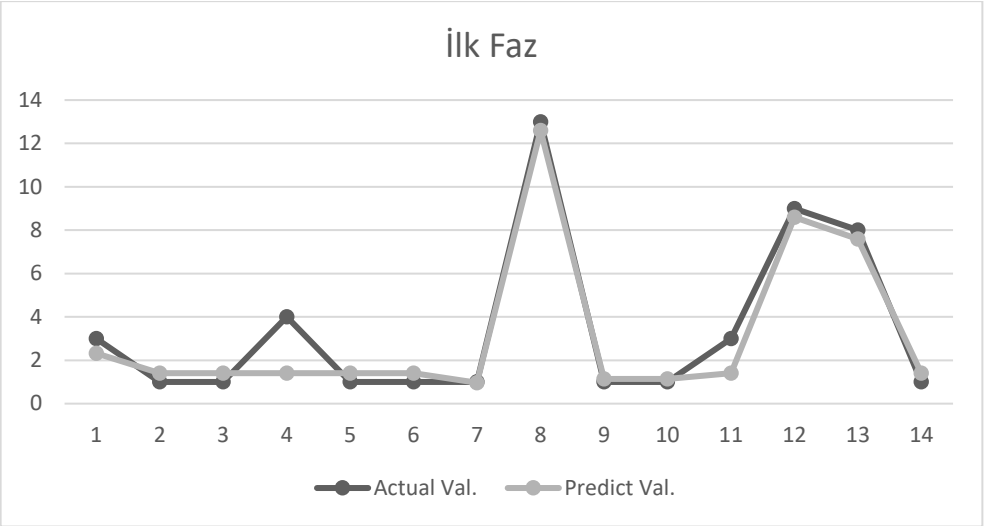
SVR metodu 3278 örnek üzerinde uygulandı. Tablo 5.15'te MongoDB üzerinde uygulanması sonucu ortaya çıkan sonuçlardan sadece 15 tanesine ait örnekler

listelenmiştir. Aynı şekilde SVR metodu 1724 örnek içeren SPRING veri seti üzerinde test edilmiştir. Ayni şekilde SVR metodu 3233 örnek içeren JIRA veri seti üzerinde uygulanmıştır. Aynı şekilde SVR metodu 1805 örnek içeren SPRING veri seti üzerinde uygulanmıştır. Aynı şekilde SVR metodu 1806 örnek içeren APACHE veri seti üzerinde uygulanmıştır.

Her bir fazda Story Point tahmini yapıldı. Bazı yapılacak olan işler birden fazla fazda tamamlanmadığında diğer fazda halen devam edebilmektedir. Bu nedenle yapılacak olan işin bittiği fazda Story Point tahmini yapılır. Bu nedenle ayni iş için birden fazla Story Point tahmini gerçekleştirilmiş oldu. Bu nedenle tekrar edilen Story Point'lerden düşük olanın toplam değerden düşürülmesi gerekmektedir. Tablo 5.19'da SVR'nin ortaya koyduğu sonuçlardan bir kısmına ait örnekler listelenmiştir.

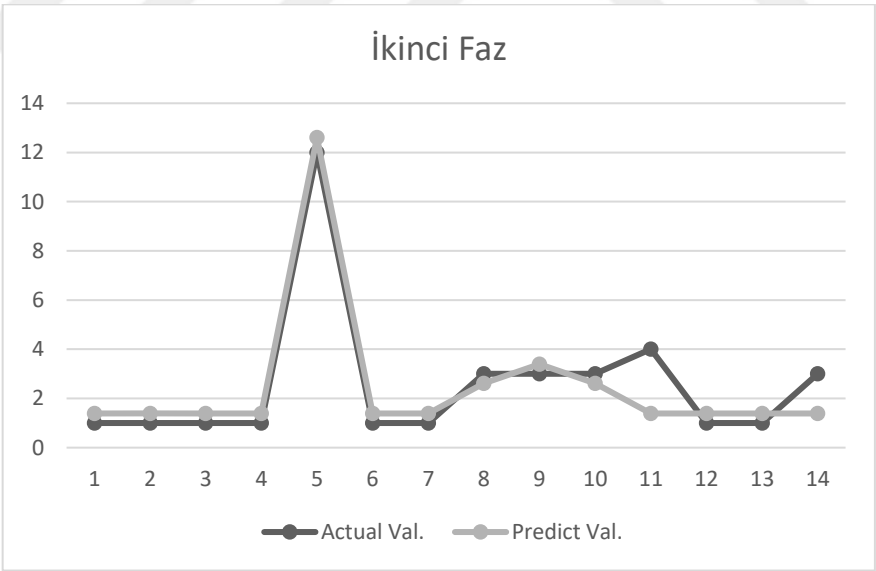
Tablo 5.19 SVR Çözümün Ürettiği Sonuçlar

İlk faz		İkinci faz		Üçüncü faz		Dördüncü	
Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin
3	2,32	1	1,39	1	1,20	1	1,90
1	1,40	1	1,39	2	1,90	1	1,02
1	1,40	1	1,39	1	0,97	8	3,84
4	1,40	1	1,39	1	1,28	1	1,19
1	1,40	12	12,60	1	1,20	1	2,28
1	1,40	1	1,39	8	3,96	3	2,80
1	0,96	1	1,39	4	2,30	5	3,08
13	12,59	3	2,60	3	2,83	1	1,42
1	1,13	3	3,39	5	3,26	2	2,19
1	1,13	3	2,60	3	2,79	3	3,01
3	1,40	4	1,39	3	3,19	1	0,84
9	8,59	1	1,39	1	2,20	1	2,28
8	7,59	1	1,39	5	3,26	2	2,22
1	1,41	3	1,39	3	2,79	1	1,19

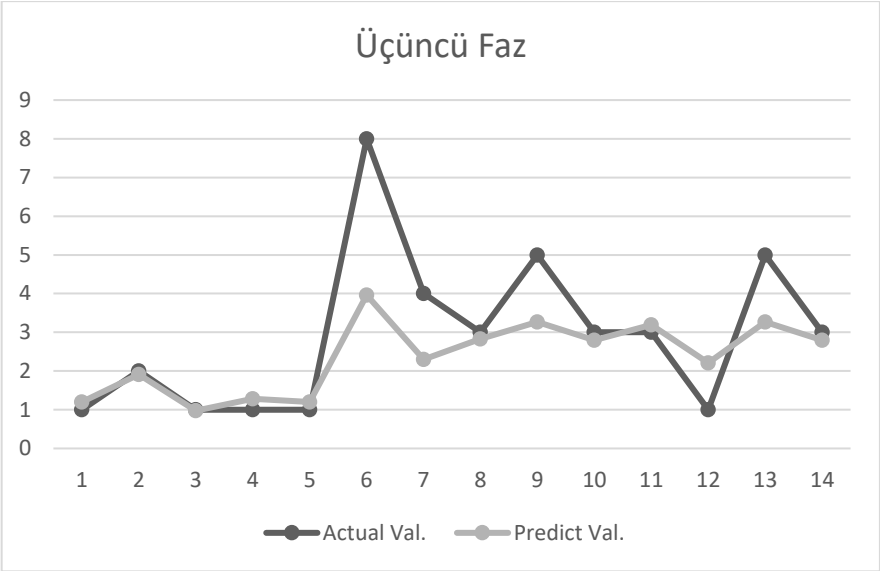


Şekil 5.16 SVR ile Story Point Tahmini- Birinci Faz

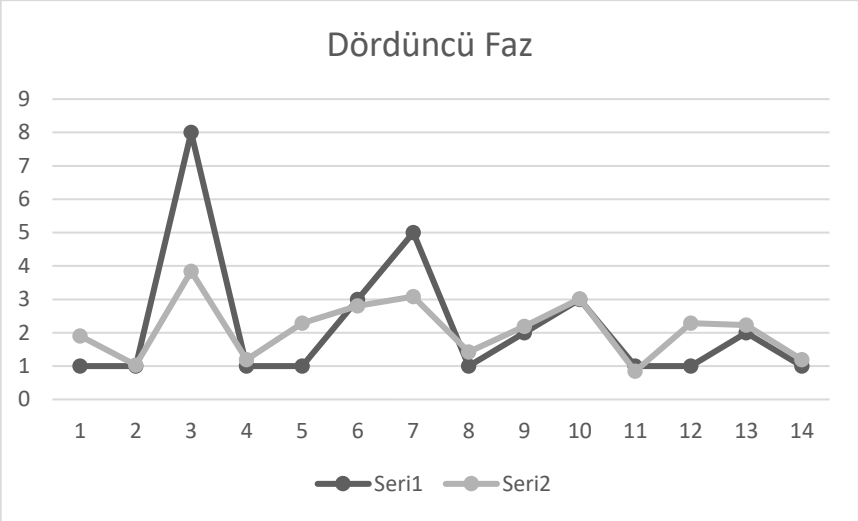
Şekilde 5.16'da oluşturulan SVR modelinin ürettiği sonuçlar grafik yardımı ile gösterilmiştir. Bu grafite X ekseninde her bir örneğin numarası vardır. Y ekseninde ise Story Point değerleri mevcuttur. Buna göre gerçek değer ile tahmin edilen değer yığılmış çizgi grafiği yardımı ile gösterilmiştir. Bu gösterim şekli, bu bölümde fazlara ait sonuçları gösteren bütün grafiklerde kullanılmıştır.



Şekil 5.17 SVR ile Story Point Tahmini- İkinci Faz



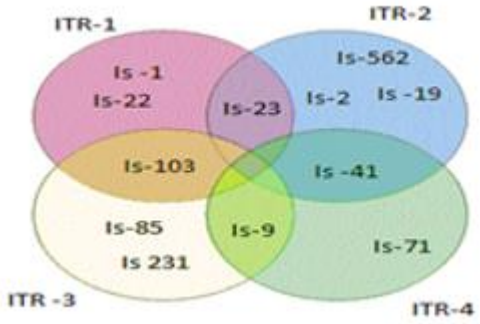
Şekil 5.18 SVR ile Story Point Tahmini- Üçüncü Faz



Şekil 5.19 SVR ile Story Point Tahmini- Dördüncü Faz

Her bir iterasyonda Story Point tahmini Yapılmıştır. Bazı iş parçaları birden fazla iterasyonda tamamlanamamıştır, ancak diğer iterasyonda da devam edebilmektedir. Bu sebeple, Story Point tahmini iş parçalarının bittiği aşamada yapılır. Bu nedenle, aynı işler için birden fazla Story Point tahmini gerçekleştirilmiştir. Bu nedenle, tekrarlanan Story Point sayısını toplam değerden çıkarmak gerekir. Bu nedenle özelleştirilmiş toplam fonksiyonu oluşturuldu. Topla

fonksiyonu Venn diyagramı mantığını kullanmaktadır. Kullanılan Venn diyagramı mantığı, Şekil 5.20'de gösterilmiştir.



Şekil 5.20 Her İterasyon için Oluşturulan Venn Diyagramı

Her Küme bir iterasyonu temsil eder ve birçok iş parçacığını içerir. ITR iterasyonlar için ve Is ise iş parçaları için kullanılmaktadır. Küme sayısı, ilgili fazdaki iterasyon sayısına bağlıdır. Bir fazda 100 iterasyon varsa, 100 küme oluşturulacaktır. Venn mantığını kullanan toplama fonksiyonu kümelerin birleşme kuralı mantığıyla çalışır. Böylece Formül 5.48 ile net Story Point değeri tespit edilebilir.

$$f(PH\ point) = \sum_{k=0}^{m} IT(k) \quad \sum_{is=0}^{n} SP(is) \tag{5.47}$$

IT iterasyon için, SP Story Point için ve PH ise faz eforu için kullanılır. Formül 5.47 ile bir iterasyonda toplam Story Point değeri hesaplanır. Ancak, bir iterasyona ait olan bazı iş parçaları, diğer iterasyonlarda da devam edebilir. Bir iş parçası 10 veya daha fazla iterasyonda aktif olabilir. Bu tekrarlanma kadar bir değer artışına neden olacağı anlamına gelir. Ancak, bunun sadece bir defa maliyete eklenmelidir. Bu nedenle tekrarlanan sorunlar iterasyonların toplamından çıkarılmalıdır. Bundan dolayı iki farklı çözüm kullanılabilir. Birincisi, Şekil 5. 18' de gösterilen Venn diyagramının mantığını kullanarak net toplam Story Point değerini hesaplayan Formül 5.48'de gösterilmiştir. ITR, iterasyonları ifade eder. Kullanılan iterasyonların sayısı arttıkça denklem daha karmaşıklaşır.

$$\boldsymbol{f(Phase)} = ITR\_1 + ITR\_2 + ITR\_3 \ldots. - \{(ITR\_1 \cap ITR\_2) + (ITR\_2 \cap ITR\_3) + \ldots.\} - \{ITR\_1 \cap ITR\_2 \cap ITR\_3 + \ldots\} + (ITR\_1 \cap ITR\_2 \cap ITR\_3 \cap ITR\_4 \ldots. \tag{5.48}$$

Bir iterasyonda toplam Story Point değerini bulmanın ikinci yolu "boardID'yi" incelemektir. Tüm ilgili iterasyonlarda tekrarların sayısı önce bulunur ve daha sonra toplam değerden çıkarılır ve bu değer, toplam değere yalnızca bir kez eklenir. Aşağıdaki adımlara göre çalışan bir script dosyası geliştirilerek toplam Story Point değeri hesaplanır:

1. Bir fazdaki tüm iş parçalarının değerlerini toplayın
2. Birinci iş parçasını seçin
3. İş parçasını boardaId'sine bakın ve iş parçasını toplam değerini bulun
4. Üçüncü adımda seçilmiş olan iş parçasını en büyük değerini seçin
5. Üçüncü adımda bulunan değeri toplam değerden çıkarın
6. Dördüncü adımda bulunan iş parçasını değerini toplam değere ekleyin
7. Tekrar eden konular bitene kadar bu işleme devam edin

$$f(PT) = \sum_{I=0}^{4} PH(\iota) \tag{5.49}$$

Son olarak, formül 5.49'deki toplam fonksiyonu ile bütün fazlardaki toplam maliyeti toplayarak projenin maliyetini kestirmiş oluruz. Bu özelleştirilmiş toplam fonksiyonu kullanılan bütün algoritmalarda çalıştırıldı.

5.3.2 Rastgele Orman (Random Forest Regression) Tekniği İle Maliyet Tahmini

Rastgele orman, karar ağacının bir versiyonudur ve Breiman tarafından önerilmiştir [112]. Rastgele Orman Regresyon denetimli öğrenme algoritmasıdır. Sınıflandırma ve regresyon için kullanılır. Bu çalışmada Rastgele Orman Regresyon yöntemi kullanılmıştır. Onu ormandan rastgele yaratan isminden gözlemleyebiliriz. Ormandaki çıkış ve ağaç arasında bağlantı vardır. Ormandaki ağaç sayısı daha yüksekse daha doğru sonuç verebilir. Orman istenen ağaca sahipse, regresyon eğrisi verilere uyar. Rastgele orman çözümü, eksik değeri kaldırabilir. Rastgele orman kategorik veriler üzerinde çalışabilir. Rastgele Orman Regresyon yöntemi kullanılarak yazılım maliyet tahmini yapmaya çalışan çalışmalar vardır [115] [116].

Rastgele orman algoritmasının iki aşaması vardır. İlk aşamada rasgele orman yaratılır. İkinci aşamada rasgele orman regresyonu üzerinden tahmin yapılır.

Rastgele orman modeli ormanının ilk aşaması model yaratmadır. Modelin ikinci aşaması test aşamasıdır.

Rastgele orman regresyon yöntemi kullanılarak her bir fazda bulunan yapılacak olan her bir iş için Story Point tahmini yapıldı. Böylece her bir faz için Story Point tahmini gerçekleştirildikten sonra toplam fonksiyonu kullanılarak toplam efor tahmin edilmiştir. Beş farklı veri seti üzerinde çalışılmıştır. Bu veri setleri JIRA, MongoDB, SPRING, JBOSS ve APACHE veri setleridir. Bu veri setlerinin her bir fazı için Rastgele orman regresyon kullanılarak ayrı ayrı tahmin gerçekleştirilmiştir.

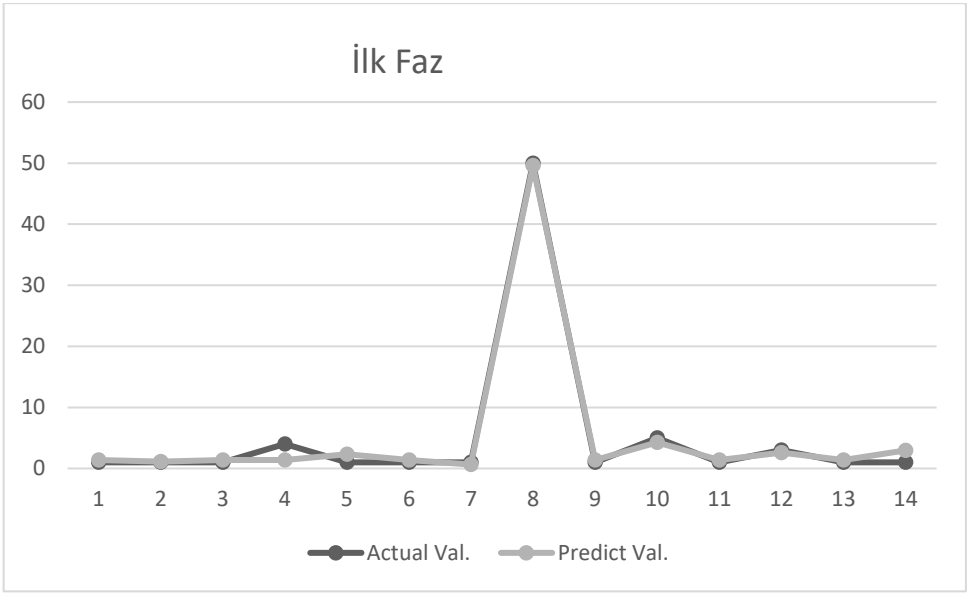
Şekil 5.21 Rastgele Orman Regresyon Tabanlı Model

Yukarıda 5.21 numaralı Şekilde gösterildiği gibi rastgele orman tabanlı bir regresyon modeli geliştirilmiştir. Bu model 5 farklı veri seti üzerinde test edilmiştir. Bu veri setleri APACHE, SPRING, JIRA, MongoDB ve JBOSS veri setleridir. Geliştirdiğimiz Model'de kullanıcı istediği oranda veriyi seçip model oluşturabilir. Ayni şekilde kullanıcı istediği oranda veriyi seçip test için de kullanabilir.

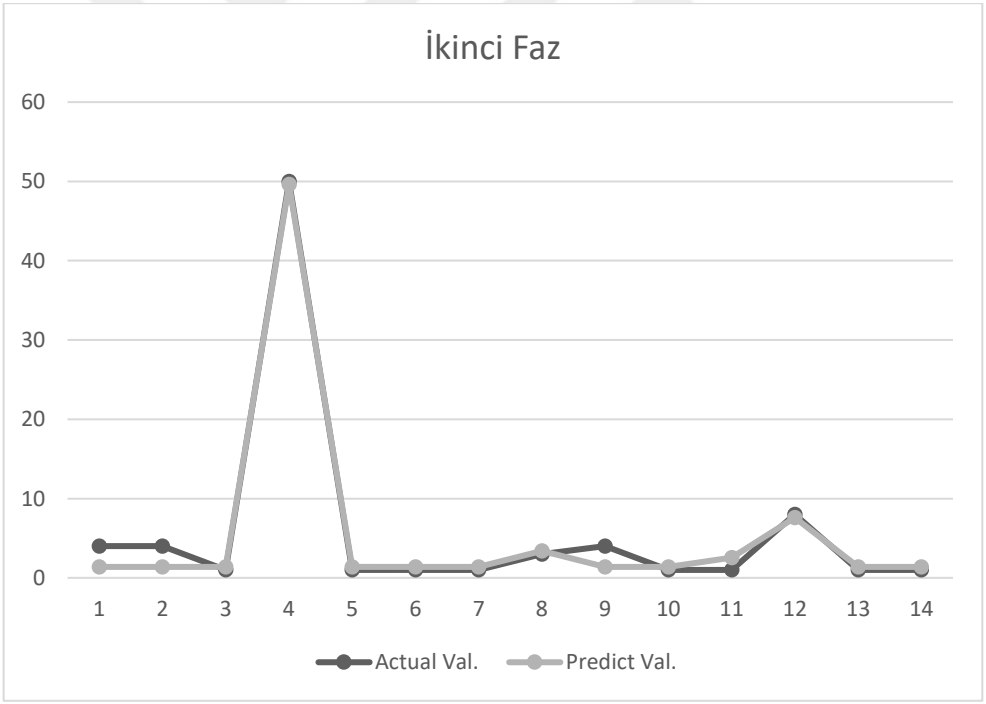
Geliştirildiğimiz modelde verinin %70'i eğitim, %30'u ise test için kullanılmıştır. Modelden anlaşılacağı üzere öncelikle yapılacak işler belirlendi daha sonra ilgili fazlara atıldı. Sonra veri üzerinde önişlem (pre-proccessing) işlemleri yapıldı. Daha sonra verinin son hali üzerinden rastgele orman regresyon modeli geliştirildi. Geliştirilen rastgele orman regresyon modeli veri üzerinde test edildi. Sonra her bir faz için Story Point tahmini gerçekleştirildi. Daha sonra çıkan sonuçlar oluşturulan toplam (aggregrate) fonksiyonu ile birleştirildi.

Tablo 5.20 Rastgele Orman Regresyon Tabanlı Model Sonuçları

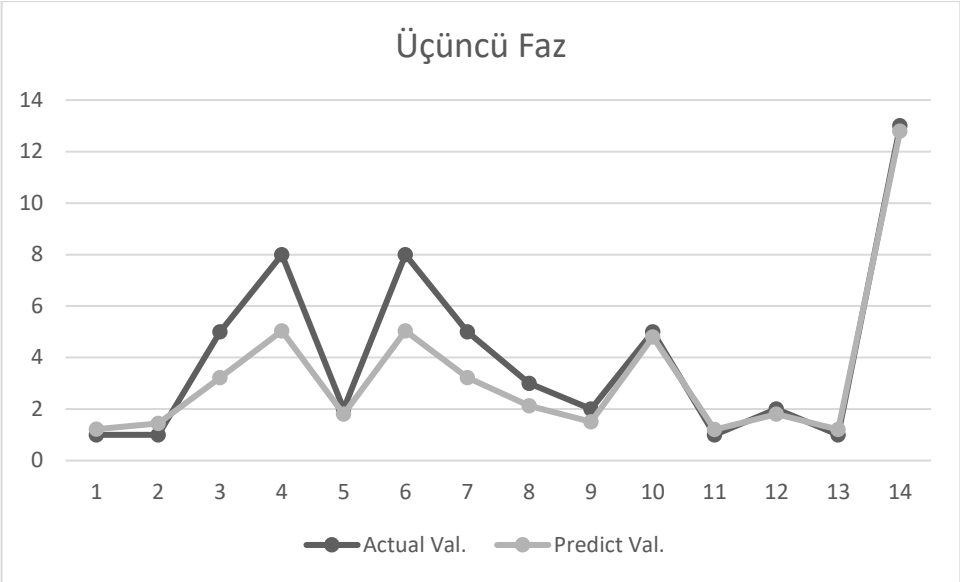
İlk faz	İkinci faz	Üçüncü faz	Dördüncü	İlk faz	İkinci faz	Üçüncü faz	
Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin
1	1,40	4	1,39	1	1,21	1	0,84
1	1,13	4	1,39	1	1,44	13	10,79
1	1,40	1	1,39	5	3,22	5	2,80
4	1,40	50	49,60	8	5,03	8	4,97
1	2,34	1	1,38	2	1,79	3	2,80
1	1,40	1	1,39	8	5,03	3	1,80
1	0,63	1	1,39	5	3,22	1	1,19
50	49,59	3	3,39	3	2,12	5	4,80
1	1,40	4	1,39	2	1,49	13	10,79
5	4,26	1	1,39	5	4,79	13	10,79
1	1,40	1	2,55	1	1,20	5	4,83
3	2,58	8	7,60	2	1,79	3	2,00
1	1,40	1	1,39	1	1,20	13	10,79
1	2,96	1	1,39	13	12,79	8	4,97



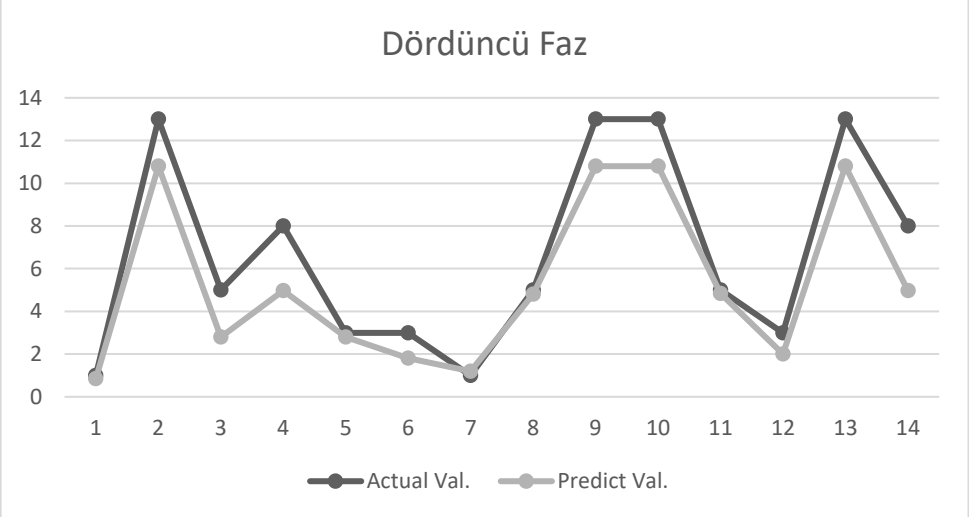
Şekil 5.22 Rastgele Orman Regresyon ile Story Point Tahmini- Birinci Faz



Şekil 5.23 Rastgele Orman Regresyon ile Story Point Tahmini- İkinci Faz



Şekil 5.24 Rastgele Orman Regresyon ile Story Point Tahmini- Üçüncü Faz



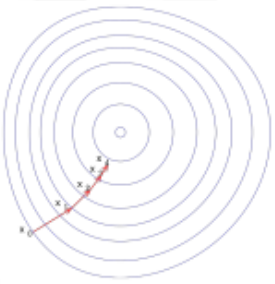
Şekil 5.25 Rastgele Orman Regresyon ile Story Point Tahmini- Dördüncü Faz

RFR (Random Forest Regression) metodu 3278 örnek üzerinde uygulandı. Tablo 5.20'te MongoDB üzerinde uygulanması sonucu ortaya çıkan sonuçlardan sadece 15 tanesine ait örnek listelenmiştir. Aynı şekilde RFR metodu 1724 örnek içeren SPRING veri seti üzerinde test edilmiştir. Aynı şekilde RFR metodu 3233 örnek içeren JIRA veri seti üzerinde uygulanmıştır. Aynı şekilde RFR metodu 1805 örnek içeren SPRING veri seti üzerinde uygulanmıştır. Aynı şekilde RFR metodu 1806 örnek içeren APACHE veri seti üzerinde uygulanmıştır.

5.3.3 Gradyan Artırma Regresyon (Gradient Boosting Regression) Tekniği İle Maliyet Tahmini

Gradyen artırma, güçlü bir makine öğrenme algoritmasıdır. Bu algoritma sınıflandırma, sıralama ve regresyon için kullanılır. Gradyen artırma algoritması, gradyan iniş ve artırmadan oluşur. İstenen ilk yükseltme algoritması, 1996 yılında Freund tarafından formüle edilmiştir. 1998'de Breiman tarafından özel kayıp fonksiyonu ile desteklemiştir. Freidman'ın 2001'de farklı kayıp fonksiyonlarını ele almak için genel hızlandırma algoritması geliştirilmiştir [117].

Şekil 5.26'da gradyanın ters yönünde hareket ederek bir fonksiyonu en aza indirgemesi gösterilmektedir.



Şekil 5.26 Gradyan İndirgenmesi

Kayıp fonksiyonbu 5.50 numaralı formülde gösterilmiştir.

$$L = \big(y, F(x)\big) = \big(y - F(x)\big) \,/\, 2 \tag{5.50}$$

Formülünü aşağıdaki diziye göre ayarlayarak minimize etmeliyiz:

$$j = \sum_{i=1}^{N} L\big(y_i\,, F(x_i)\big) \tag{5.51}$$

F (x1), F (x2), ..., F (xn) 'nin sadece birkaç değer olduğuna dikkat edin. F (xi) parametrelerini iyileştirebiliriz ve türevlerini alabiliriz.

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L\big(y_i,\ F(x_i)\big)}{\partial F(x_i)} = \frac{\partial L\big(y_i,\ F(x_i)\big)}{\partial F(x_i)} = F(x_i) - y_i \tag{5.52}$$

Kısmı türevini aldıktan sonra 5.52 numaralı formülü elde ederiz.

$$y_i - F(x_i) = \frac{\partial J}{\partial F(x_i)} \tag{5.53}$$

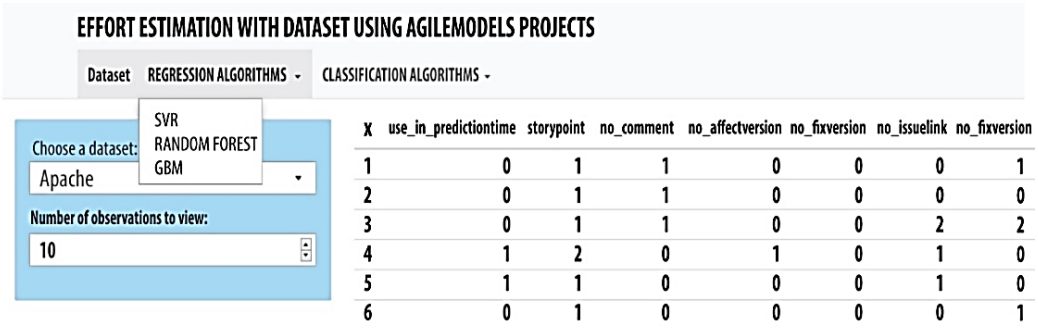
Gradyan artırma yaklaşımı farklı kayıp fonksiyonlarını kullanır. Mutlak kayıp fonksiyonu ve Huber kayıp fonksiyonu gibi.

Gradyan artırma, tahmin modelleri oluşturmak için en güçlü tekniklerden biridir. Gradyan artırma regresyon ağaçları üretir ve bunları sıralı bir şekilde topluluğa ekler. İlk ağaç, Rasgele Ormanlarda yapıldığı gibi üretilir. Buradaki temel fark, ilk ağacın ürettiği tahmin hatalarını en aza indirmeyi amaçlayan ikinci ağacın üretilmesidir [117]. Hem birinci hem de ikinci ağaçlar topluluğa eklenir ancak her birine farklı ağırlıklar verilir. Bu süreç birden çok kez tekrarlanır. Her adımda, öğrenilen tüm topluluğun hatalarına göre yeni bir ağaç eğitilir ve daha sonra topluluğa eklenir. Son topluluk, yeni girdilerin sonucunu kestirmek için bir model olarak kullanılır.

Rasgele Ormanların aksine, GBM'lerde düşük bir öğrenici sadece regresyon ağaçları değil, sinir ağları veya doğrusal regresyon gibi diğer regresyon öğrenme algoritmaları da olabilir [118]. Uygulamada regresyon ağaçları düşük öğreniciler olarak kullanılmış ve 100 ağaç üretilmiştir.

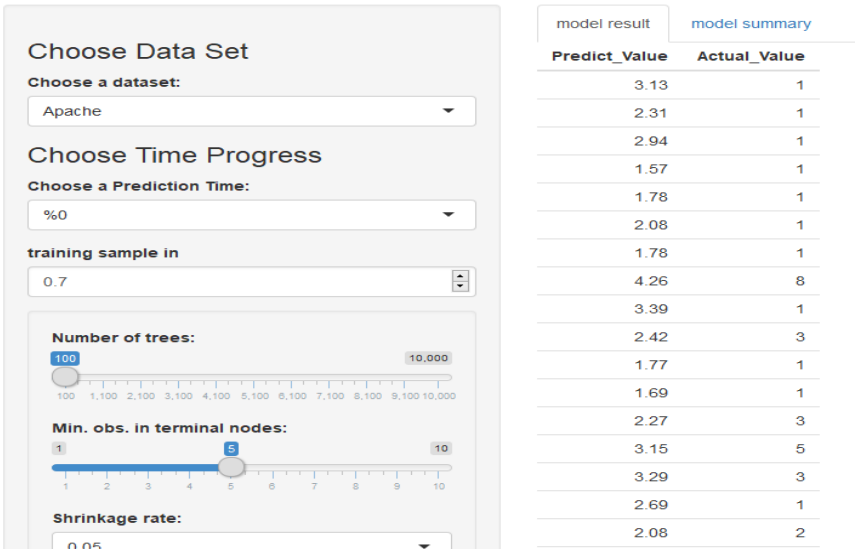
Şekil 5.27 Gradyan Artırma Regresyon Tabanlı Model

Yukarıda 5.27 numaralı Şekilde gösterildiği gibi GBM tabanlı bir regresyon modeli geliştirilmiştir. Bu model 5 farklı veri seti üzerinde test edilmiştir. Bu veri setleri APACHE, SPRING, JIRA, MongoDB ve JBOSS veri setleridir. Geliştirdiğimiz Model'de kullanıcı istediği oranda veriyi seçip model oluşturabilir. Ayni şekilde kullanıcı istediği oranda veriyi seçip test için de kullanabilir. Geliştirildiğimiz modelde verinin %70'i eğitim, %30'u ise test için kullanılmıştır. Modelden anlaşılacağı üzere öncelikle yapılacak isler belirlendi daha sonra ilgili fazlara atıldı. Sonra veri üzerinde önişlem (pre-proccessing) işlemleri yapıldı. Daha sonra verinin son haline üzerinden gradyan artırma regresyon modeli geliştirildi. Geliştirilen gradyan artırma regresyon modeli veri üzerinde test edildi. Sonra her bir faz için Story Point tahmini gerçekleştirildi. Daha sonra çıkan sonuçlar oluşturulan toplam (aggregrate) fonksiyonu ile birleştirildi.

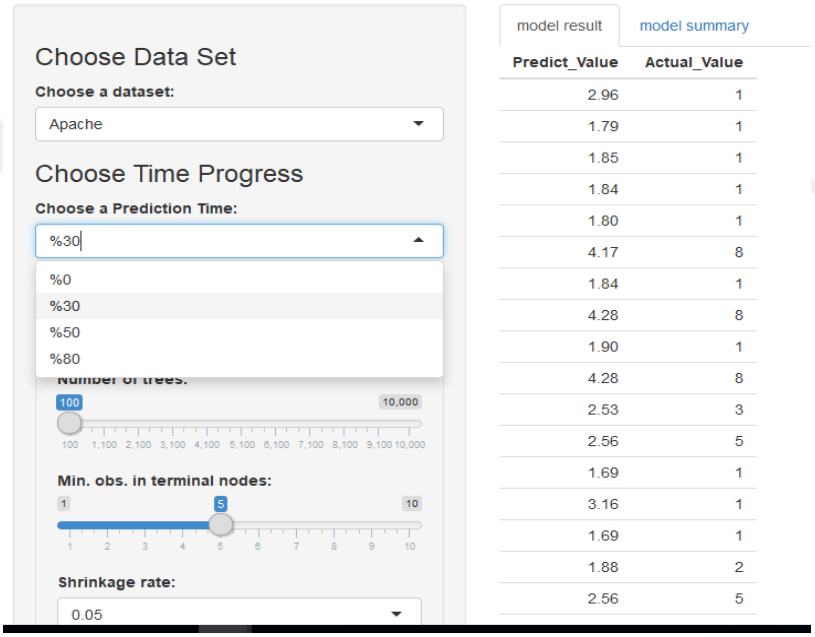


Şekil 5.28 Uygulama Giriş Ekranı

Uygulamanın ilk açılan sayfası Şekil 5.28'de gösterilmiştir. Burada veri setinin içeriği görülmektedir. Daha sonra regresyon algoritma bölümünden GBM algoritması seçilir. GBM algoritmasının parametreleri kullanıcı tarafından girilecek şekilde ayarlanmıştır. Seçilen veri setinin 4 ayrı fazı için ayrı ayrı veri setleri üzerine algoritma çalıştırılmıştır. Veri setinin ne kadarlık kısmı eğitim ne kadarlık kısmı test için kullanılacağı da kullanıcı tarafından belirlenecektir. En ideal olarak %70' i eğitim, %30' u test olarak seçilmiştir. Ağaç sayısı olarak 100 ağaç türetilmiştir. Her bir düğümdeki yaprak sayısı 5 olarak, öğrenme oranı 0.05, ağaç derinliği de 10 olarak seçilmiştir. Bu değerlere göre algoritmanın ürettiği Story Point değerleri Şekil 5.29'da gösterilmiştir.



Şekil 5.29 İlk Aşama için Gradyan Artırma Regresyon Tabanlı Hikâye Noktası Tahmini (R'da geliştirilmiştir.)



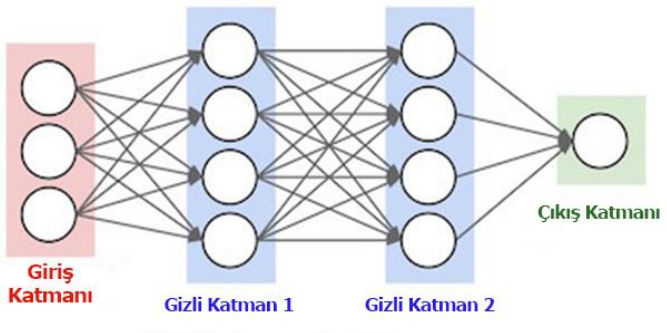
Şekil 5.30 İkinci Aşama için Gradyan Artırma Regresyon Tabanlı Hikâye Noktası Tahmini

Şekil 5.29 ve 5.30'da ilk iki faza ait Story Point tahminleri gösterilmiştir. Burada yaptığım uygulamanın giriş kısmı görülmektedir. Diğer fazlara ait çıktılar buraya eklenmemiştir. Uygulamanın ekranından kolayca görebileceğiniz gibi istediğiniz veri setini seçebiliyorsunuz. İstediğiniz veri setini seçtikten sonra o veri seti için

ilgili fazı seçebiliyorsunuz. İlgili fazı seçtikten sonra gerekli ayarlamaları yaptıktan sonra uygulamayı çalıştırıp algoritmanın başarısını görebiliyorsunuz. Burada sadece iki faza ait örnekler sunulmuştur.

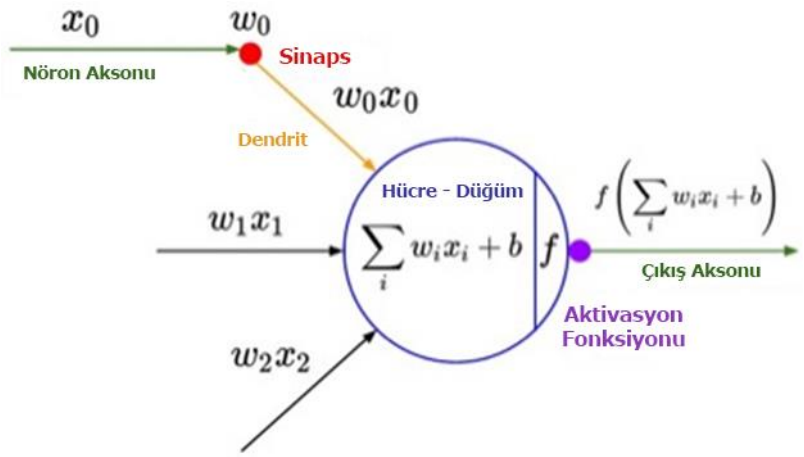
5.3.4 Çok katmanlı Perseptron (Multilayer Perceptron Regression) Tekniği İle Maliyet Tahmini

Çok katmanlı Perseptron (MLP) yöntemi, ileri beslemeli yapay sinir ağının bir çözümüdür. Çok katmanlı Perseptron yöntemi üç katmandan oluşur ve her bir Katman düğümlerden oluşur. Her düğüm, aktivasyon fonksiyonunu kullanan bir nöronu temsil eder. Lineer olarak ayrılamayan veriler için Çok Katmanlı Perseptron yöntemi kullanılmıştır. Çok Katmanlı Perseptron, giriş katmanı, gizli katmanlar ve çıkış katmanı olan üç ana katmana sahiptir. Yazılım maliyetinde kullanılan bir modeldir [119].



Şekil 5.31 Üç Katmanlı Perseptron Çözümü

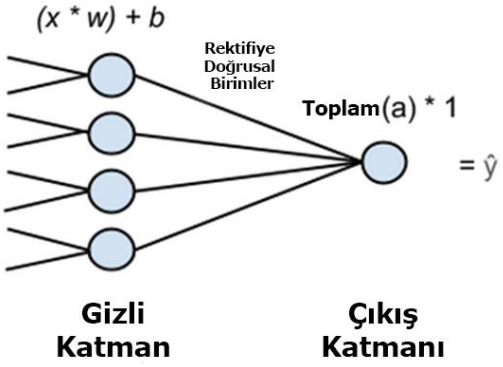
Şekil 5.31'de üç katmanlı yapay sinir ağı grafiği gösterilmiştir. Bu grafikte, verilerin doğrusal olarak ayırt edilemediğinin ağ üzerinden temsil edilebileceğini açıkça görebiliyoruz. Gizli katmanın sayısı artıyorsa, ağın karmaşıklığı da artıyor. Eğer veriler doğrusal olarak ayrıştırılırsa çıktı tabakası doğrudan girdi katmanına bağlanır.



Şekil 5.32 Örnek Bir Düğüm bileşeni

Şekil 5.32'de bir düğümün bileşenleri temsil edilmektedir. Bileşenlerden biri ağırlıktır. Ağırlık, gradyan inişini kullanılarak hesaplanır.

Genel olarak, çok katmanlı perseptron, denetimsiz öğrenme, denetimli öğrenme yoluyla sınıflandırma veya regresyon yoluyla kümelenme amacıyla kullanılır. Bu sayede çok katmanlı perseptron etiketlenmemiş veri grubunu destekliyor, etiketlenmiş verileri kategorize ediyor veya sürekli değerleri öngörüyor. Çok katmanlı perseptron yaklaşımında lojistik regresyon, ağ oluşturmanın veriyi yanlış formda sürdürdüğü sınıflandırma için kullanılmıştır. Fakat gerçek regresyon formunda regresyon, bir dizi girdiyi başka bir çıktı grubuna aktarır. Çok katmanlı perseptron formlarından biri regresyondur. Ancak bu form geleneksel yaklaşımlardan farklıdır. Bu yaklaşımın başka bir zorluğu, farklı sayıda gizli katmanı temsil eden verilerdir. Gizli katmanın sayısı arttırıldığında, ağın karmaşıklığı da artmaktadır. Aynı şekilde çözüm de zorlaşmaktadır.



Şekil 5.33 Regresyon ile Sinir Ağı Kullanımı

Yukarıda gösterilen Şekil 5.33'de, x ağın girişini, nöral ağın önceki katmanından ileriye taşınan özellikleri temsil eder. Son gizli katmanın her bir düğümünde birçok x'ler beslenecek ve her x, karşılık gelen bir ağırlık, w ile çarpılacaktır. Böylece aktivasyon fonksiyonu beslenir. Bu durumda aktivasyon fonksiyonu, sigmoid aktivasyon fonksiyonlarının yaptığı gibi sığ gradyanlarda doymadığı için yaygın olarak kullanılan ve oldukça kullanışlı olan bir düzeltilmiş doğrusal birimdir (ReLU). Her bir gizli düğüm için, ReLU bir aktivasyon çıkarır ve aktivasyonların toplamı içinden geçen aktivasyonlar çıkış düğümünde toplanır [35].

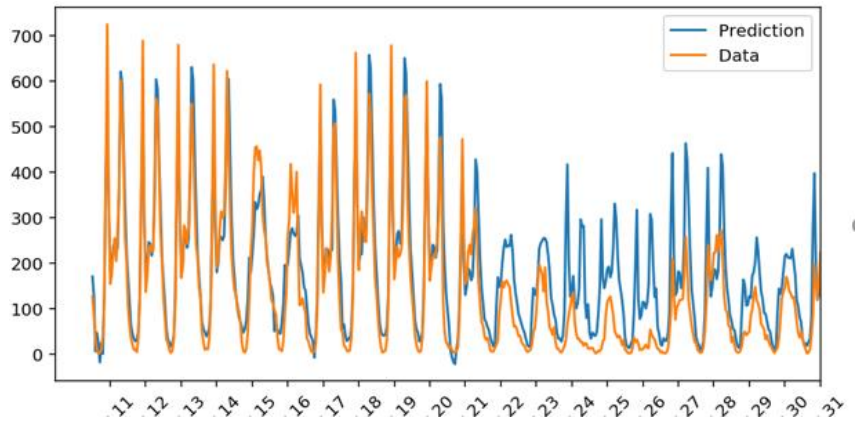
Bu sinir ağı birçok girdi ve bir girişe sahip olan regresyon çözümüne hizmet eder. Her düğüm değeri, önceki katman aktivasyon fonksiyonları ile çarpılır. Y, fonksiyonun çıktısıdır ve bağımsız değişken x değişkenlerine bağlıdır. Önerilen sinir ağı, y değerini y'nin gerçek-gerçek değeriyle karşılaştırır ve buna göre, hata en aza indirilene kadar ağın ağırlıkları üzerinde bir ayarlama yapar. RMSE (Root Mean Square Error) kayıp işlevi olarak kullanılır. Çok katmanlı perseptron regresyon yöntemi "kara kutu" olarak çalışır ve çıktıların nasıl elde edildiğine dair herhangi bir bilgi veya muhakeme sağlamazlar. Yazılım Maliyet verileri iyi bir davranış göstermediği durumlarda, parametreler arasındaki ilişkinin ve bu ilişkilerin oluşturduğu ağırlığın yeterli olup olmaması modelin başarısını belirleyecektir.

Şekil 5.34 Çok Katmanlı Perseptron Tabanlı Model

Şekil 5.34'te görüldüğü gibi çok katmanlı perseptron tabanlı bir model geliştirilmiştir. Bu Model'de öncelikle yapılacak olan isler belirlenir. Yapılacak olan isler belirlendikten sonra ilgili faza atılır. Yapılacak olan is ilgili faza atıldıktan sonra veriler üzerinde on işlemler gerçekleştirilir. Bundan sonra oluşturulan veri üzerinden çok katmanlı perseptron algoritması uygulanır. Daha sonra oluşturulan model test edilir. Model test edildikten sonra toplam fonksiyonu çalıştırılır.

Ağın eğitimi sırasında çıkış ile istenilen sonuç arasındaki farkın azaltılması için ağırlıklar eğitim algoritmasına bağlı olarak değişir. Ortalama karesel hata eğrisi, eğitim iterasyonları boyunca ağın çıkışı ile istenilen sonuç arasındaki farkın karesini gösterir. Ağın fazla eğitilmesi durumunda ağın genelleştirme yapması zorlaşır. Ağın eğitimini durdurmak için geçerlilik kriteri kullanılır. İyi bir genelleştirme elde edilebilmesi için geçerlilikteki hatanın artması durumunda ağın eğitimi durdurulmalıdır.

Aşağıdaki şekil 5.35'te çok katmanlı perseptron regresyon çözümünün tahmin ettiği değerler ile veriler arasındaki korelasyon görülmektedir.

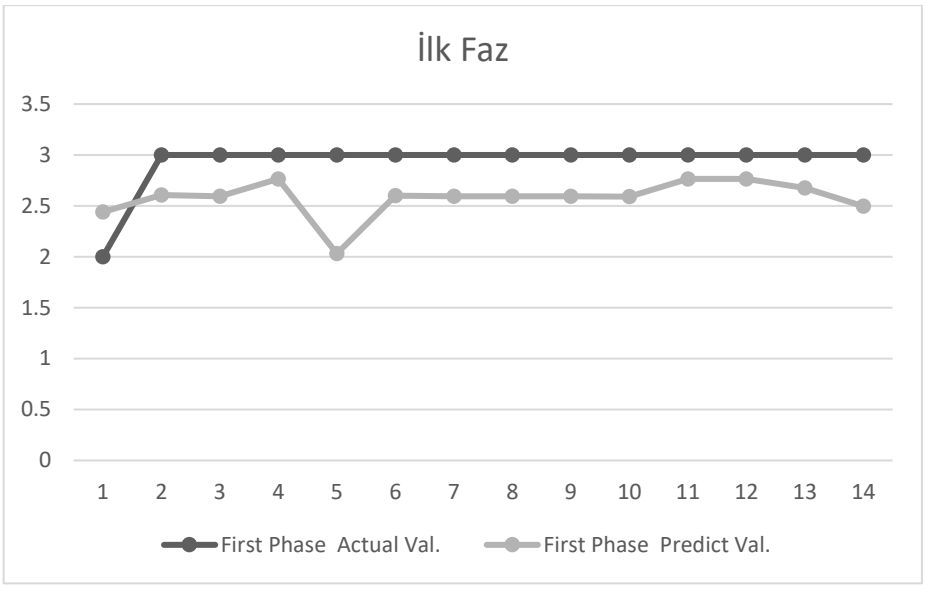


Şekil 5.35 Çok Katmanlı Perseptron Tabanlı Model Sonuç Grafiği

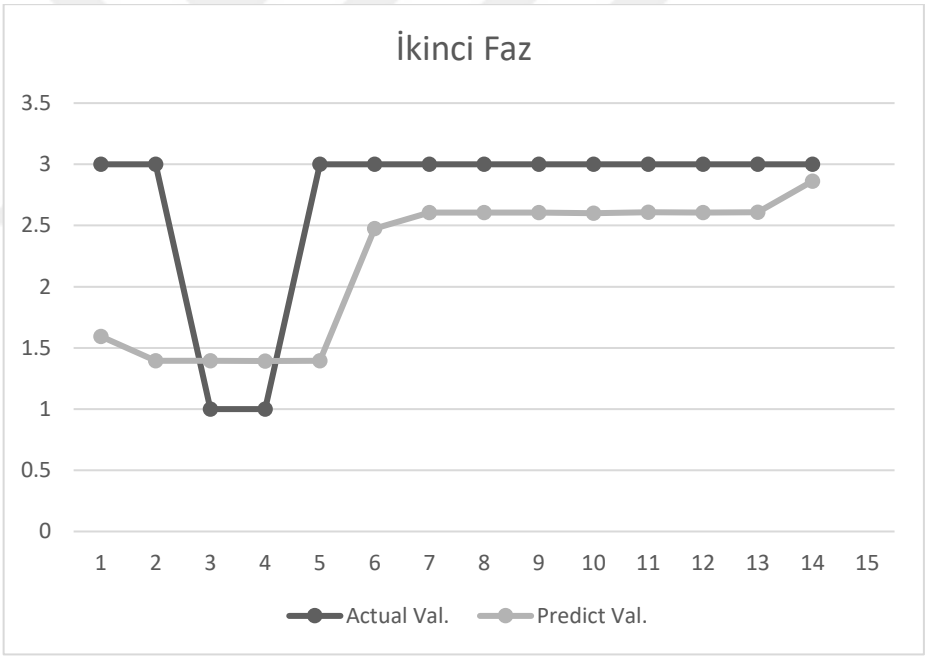
Aşağıdaki Tablo 5.21'de çok katmanlı perseptron algoritmasının her bir fazda uygulanması sonucu ürettiği değerler görülmektedir. Burada kullanıcı istediği ağı oluşturmak için istediği veri oranı seçebilme imkânına sahiptir. Ayni şekilde kullanıcı istediği veri oranını test için seçebilme imkânına sahiptir. Bu çalışmada verinin %70'i eğitim için kullanılmıştır. Ayni şekilde verinin %30'u da test için kullanılmıştır.

Tablo 5.21 Çok Katmanlı Perseptron Tabanlı Hikâye Noktası Tahmini

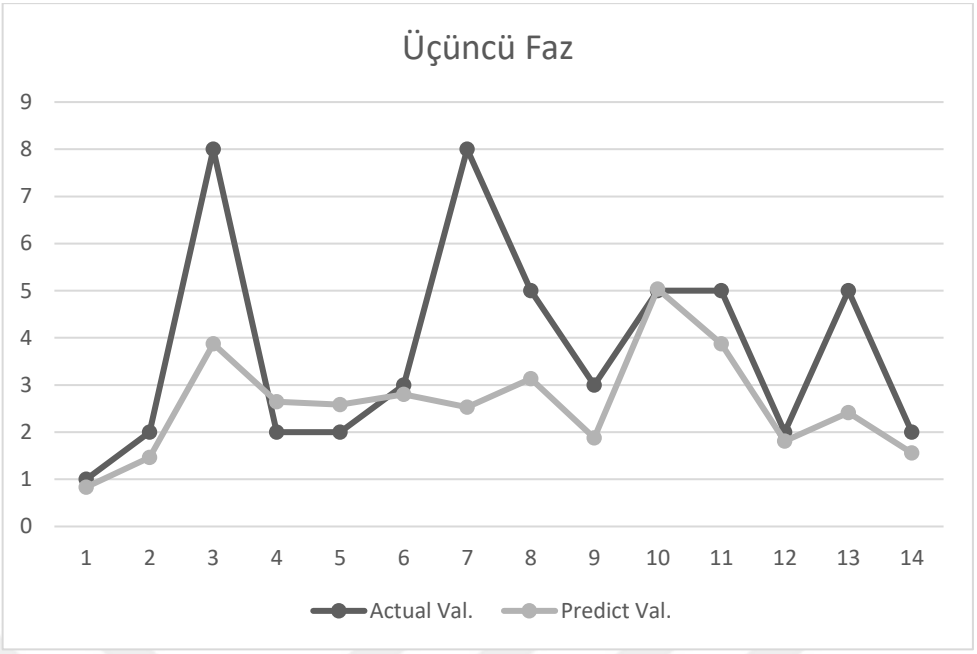
İlk faz		İkinci faz		Üçüncü faz		Dördüncü	
Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin	Gerçek	Tahmin
2	2,4	3	1,59	1	0,83	3	2,44
3	2,60	3	1,39	2	1,46	1	1,19
3	2,59	1	1,39	8	3,87	3	3,40
3	2,76	1	1,39	2	2,64	1	1,12
3	2,03	3	1,39	2	2,58	5	4,97
3	2,59	3	2,47	3	2,79	1	1,19
3	2,59	3	2,60	8	2,53	13	10,7
3	2,59	3	2,60	5	3,13	5	4,97
3	2,59	3	2,60	3	1,87	5	4,81
3	2,59	3	2,60	5	5,03	1	1,36
3	2,76	3	2,60	5	3,87	5	3,44
3	2,76	3	2,60	2	1,81	5	4,81
3	2,67	3	2,60	5	2,41	2	1,87



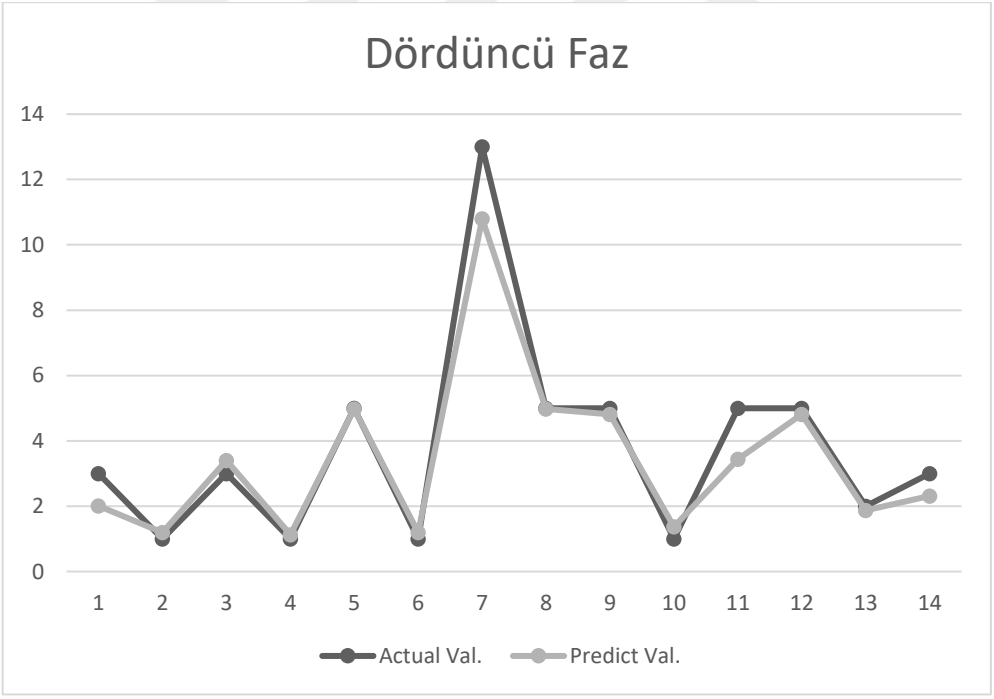
Şekil 5.36 Çok Katmanlı Perseptron ile Story Point Tahmini- Birinci Faz



Şekil 5.37 Çok Katmanlı Perseptron ile Story Point Tahmini- İkinci Faz



Şekil 5.38 Çok Katmanlı Perseptron ile Story Point Tahmini- Üçüncü Faz



Şekil 5.39 Çok Katmanlı Perseptron ile Story Point Tahmini- Dördüncü Faz

Scrum ile geliştirilen yazılımların maliyet tahmini gerçekleştirmek için regresyon tabanlı 4 farklı algoritma kullanılarak elde edilen sonuçlar paylaşılmıştır.

5.3.5 Scrum ile Geliştirilen Yazılımlar için Maliyet Tahmin Modelinin Sonuçları

Scrum metodolojisi kullanılarak geliştirilen yazılımların efor tahmini için Story Pointleri baz alarak 4 farklı regresyon tabanlı makine öğrenme algoritması kullandık. Daha önce de belirtildiği gibi, özellikleri çok düşük kayıplar ile azalttı. Ayrıca, kategorik verilerimizi regresyon tabanlı algoritmaları uygulamak için düzenledik. Story Point tahminine uygulanan yöntemleri farklı ortamlarda test ettik ve sonuçları kaydettik.

Önerdiğimiz modelin başarısını ölçmek için bağıl ortalama hata metodunu kullandık. Kullandığımız başarı oranını ifade eden denklem aşağıdaki 5.54 numaralı formülde ifade edilen şekilde çalışmaktadır.

$$SP_{Absolute} = | SP_{Actual} - SP_{Predicted} | \quad (5.54)$$
$$SP_{Relative} = | 1 - (SP_{Predicted} / SP_{Actual}) |$$
Percent error is then:
$$SP_{Percentage} = | (SP_{Actual} - SP_{Predicted}) / SP_{Actual} | \times 100\%$$

Önerilen modelin başarısı 5.54 numaralı formülle hesaplanmaktadır. SPAbsolute bağıl story point hata değerini vermektedir. SPActual ise gerçek story point değeri için kullanılmaktadır. SPPredicted tahmin edilen story point degeri için kullanılmaktadır. SPPercentage ise bağıl hata oranını vermektedir.

Tablo 5.22'de JIRA veri seti üzerinde uyguladığımız algoritmaların başarısı listelenmiştir.

Tablo 5.22 JIRA Veri Seti Üzerindeki Uygulanan Algoritmaların Başarısı

	Veri Tabanı ve Alt Seviye Modül Geliştirme- Database-low module development *(Ilk faz) (%)*	Servis ve Kullanıcı Modüllerini Geliştirme- Service- user module development *(Ikinci faz) (%)*	Test ve Beta Test - Test- beta test *(Üçüncü faz) (%)*	Bakım ve Destek- Maintenance *(Dördüncü faz) (%)*
SVR	75	82	84	84
RF	70	78	86	85
GBR	82	86	86	87
MLP	76	77	82	84

JIRA veri seti üzerinde en iyi başarıyı gradyan artırma algoritması dördüncü fazda göstermiştir. GBM algoritması dördüncü fazda %88'lik bir başarı ortaya koymuştur. Diğer bütün veri setleri için de aynı şekilde sonuçlar ortaya konulmuştur.

Tablo 5.23 Algoritmaların Farklı Veri Seti Üzerindeki Başarısı

Çözüm	APACHE (%)	JIRA (%)	SPRING (%)	JBOS (%)	MONGODB (%)
SVR	76	80	72	70	97
RF	78	79	74	72	97
GBR	83	85	79	77	99
MLP	74	79	70	73	92

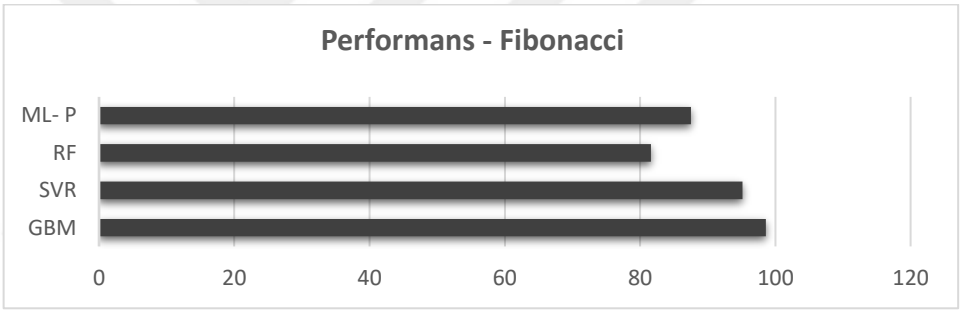
Tablo 5.23'te her bir algoritmanın bütün veri setleri üzerinde genel (overall) başarısı ortaya konmuştur. Tablo 5.23'ten de açıkça görüleceği üzere ortaya konan bütün çözümler en yüksek başarısını MongoDB veri seti üzerinde ortaya koymuşlardır. MongoDB veri setindeki yapılacak olan her bir işin Story Point değerleri birbirine çok yakındır. Bu nedenle ortaya konan çözümler çok iyi sonuçlar ortaya koymuşlardır. Scrum teknolojisi ile geliştirilen bir yazılımın maliyetini, geliştirildiğimiz yöntem sayesinde yüksek bir başarı oranı ile projeye başlamadan

önce kestirilebilmektedir. Burada önemli olan yapılacak olan işlerin detaylı bir şekilde belirlenmesi ve ilgili fazlara atılmasıdır.

Story Point değerleri Fibonanci serisinin özelleştirilmiş bir şeklinden oluşmaktadır. Bu çalışmada kullanılan algoritmaların ortaya koyduğu değerlerin ilgili özelleştirilmiş Fibonanci değerine atanması ve böylece çok daha yüksek oranda yapılacak olan islerin Story Point değerinin saptanması işlemini de gerçekleştirdik.

Tablo 5.24 Algoritmaların JIRA Veri Seti Üzerindeki Fibonacci ile Sunum Başarısı

Çözüm	SVR (%)	RFR (%)	GBR (%)	MLP (%)
	95	81	98	87



Şekil 5.40 Algoritmaların JIRA Veri Seti Üzerindeki Fibonacci Gösterim Başarısı

Sonuç olarak öncelikle yapılacak olan iş parçalarını belirledik. Bu iş parçalarını ilgili iterasyonlara attık. Her bir iterasyonun değeri belirlendikten sonra her bir faza atılacak olan iterasyonlar belirlendi. Her bir fazın değeri belirlenirken geliştirdiğimiz toplam fonksiyonu kullanıldı. Her bir fazın değeri belirlendikten sonra oluşturulan ikinci toplam fonksiyonu ile fazlar birleştirilerek toplam efor tahmini gerçekleştirilmiş oldu. Bu bağlamda bizim çalışmamıza birebir uyan bir çalışma bugüne kadar ortaya konmamıştır. Bu nedenle tam olarak bir karşılaştırma yapmak zordur. Fakat yine de kullandığımız metotları kullanarak ya da farklı veri setleri üzerinde çalışılarak yapılan çalışmaların sonuçları ile geliştirdiğimiz modelin sonuçlarını karşılaştırmalı olarak Tablo 5.25'de listeledik. Önerdiğimiz modeller

GBR yöntemi bugüne kadar yapılan çalışmalardan daha iyi sonuçlar ortaya koymuştur.

Tablo 5.25 Story Point Tabanlı Yapılan Maliyet Çalışmalarının ve Önerdiğimiz Modelin Başarı Sonuçları

Yazarlar	Metotlar	Sonuçlar	Veri Setleri
Porru et al	SVM, NB, KNN and DT	MMRE = {0,16- 0,61}	8 açık kaynak kodlu proje
Aditi Panda et al	GRNN, PNN, GMDA & CCNN	MSE=0,0059, R2 =0,93 MMRE=0,14 ve PRED=94,76 (%)	21 endüstri projesi
Alaa et al M	Multi Layer Neural Network	MAE=12% & Boyut Tahmini = 5,9%	4 endüstri projesi
Satapathy et al	SVR- Polynomial, RBF, Sigmoid	MMRE=0,07 PRED= 95,90 (%)	21 Proje veri seti- NASA
Önerilen MLP	MLP	PRED = 87,5 (%)	5 proje – 3233 iş parçası
Önerilen SVR-RBF	SVR- RBF	PRED = 95,12 (%)	5 proje – 3233 iş parçası
Önerilen GBR	GBR	PRED = 98,61 (%)	5 proje – 3233 iş parçası
Önerilen RFR	RFR	PRED = 81,6 (%)	5 proje – 3233 iş parçası

Önerdiğimiz model Scrum ile geliştirilen yazılımların maliyet tahmini için önemli bir çözüm sunmaktadır. Yazılım proje yöneticilerinin projeye başlamadan önce projede yapılacak olan işleri ayrıntılı bir şekilde gerçekleştirmelerini zorunlu kılmaktadır.

Tablo 5.26 Önerilen modellerin fazlı ve fazsız sonuçlarının karşılaştırılması

	SVR	Random Forest	Gradient Boosting	Multi-Layer Perceptron
Toplam Story Point Değeri	15.264	15.264	15.264	15.264
Toplam Story Point Değeri Fazlı	14.917,5	14.928,19	15.233,47	14.122,25
Toplam Story Point Değeri Fazsız	14.119,2	13.907,33	14.349,68	13.751,33

Gerçek story point değeri, modellerin fazlı halleri ile ürettiği toplam story point değeri ve modellerin fazsız halleri ile ürettiği story point değeri Tablo 5.26 da gösterilmiştir. Modellerin fazlı hallerinin ürettiği değerler gerçek değerlere daha yakındır. Bu nedenle oluşturulan modeller istenilen sonucu ortaya koymuştur.

Ayrıca projeyi fazlara ayırarak yazılım yöneticileri için esnek bir yapı oluşturmaktadır. Yazılım yöneticileri için hata payını tolere edebilecekleri bir yapı ortaya koymaktadır. Geliştirdiğimiz modelin en önemli getirisi ise öznelliği ortadan kaldırıp tamamen objektif bir tahmin gerçekleştirebilmesidir. Bu bağlamda önerdiğimiz model kendi alanında ciddi bir açığı kapatacaktır. Önerdiğimiz modelin önerilen diğer modellere üstünlüğüne bakıldığında ise yazılım geliştirme sürecini parçadan bütüne gidecek bir şekilde ele alarak yazılım yöneticileri için daha doğru bir tahmin gerçekleştirmelerini sağlayabilmesidir. Bunu da oluşturduğu iterasyon ve faz yöntemi ile gerçekleştirmektedir.

6
Sonuç ve Öneriler

Bu tez çalışmasında üç farklı model geliştirildi. Bu modeller yazılım metodolojilerine bağlı olarak ve yazılım metodolojisinden bağımsız olarak yazılım maliyet tahmini gerçekleştirebilmektedir. Bu bağlamda geliştirilen her bir model farklı veri setleri üzerinde uygulanmıştır. Geliştirilen her bir modelin sonuçları uygulama kısmında sunulmuştur. Bu bölümde ise bu modellerin ortaya koydukları sonuçlar yorumlanmıştır. Ayrıca önerdiğimiz modellerin eksik ve pozitif yönleri üzerine bir tartışma gerçekleştirilmiştir.

Yazılım maliyet tahmininde ilk geliştirdiğimiz model yapay sinir ağları tabanlı modeldir. Bu model veri seti özelliklerinin gruplandırılması itibari ile bugüne kadar geliştirilen modellerden ayrılmaktadır. Bu model vesilesi ile benzer özelliklerin gruplandırılması ile oluşturulan yapay sinir ağı modelinin daha iyi sonuçlar verdiğini gördük. Geliştirdiğimiz model 9%'luk bir sapma ile maliyet tahmini geçekleştirmektedir. Bu sapma oranı benzer çalışmalara göre daha iyidir. Geliştirilen bu model vesilesi ile Amaç kısmında belirtilen sorulara aşağıdaki cevapları verdik.

Soru 1: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı modelin bugüne kadar geliştirilen yapay sinir ağları tabanlı yazılım tahmini modellerine göre farkı nedir?

Cevap1: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı model, bugüne kadar gerçekleştirilen yapay sinir ağları tabanlı modellerden benzer özelliklerin gruplandırılması ile ayırılmaktadır. Veri seti örneğinin az olmasında dolayı daha önce yapılan çalışmalarda özellikler gruplandırılmamıştır. Fakat bu çalışmada özellikler gruplandırılarak yapay sinir ağı oluşturuldu.

Soru 2: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı modelde kullanılan veri setinin özellikleri gruplandırılmıştır. Neden bu gruplandırma işlemi yapıldı?

Cevap 2: Yazılım maliyet tahmini için geliştirilen yapay sinir ağları tabanlı modelde özellikler gruplandırıldı. Bugüne kadar Cocomonasa veri seti üzerinde birçok yapay sinir ağı modeli ortaya konulmuştur. Örnek sayısı az olması ve özellik sayısının fazla olmasından dolayı oluşturulacak olan YSA'nın iyi sonuç ortaya koyamayacağı düşünüldüğünden. Örnek sayısı aynı sabit tutup özellik sayısının azalması ile daha iyi sonuçlar ortaya konulabileceği düşünüldüğünden veri setinin özellikleri gruplandırılmıştır. Dört farklı grup oluşturulmuştur. Bu gruplar ürün özellikleri, personel özellikleri, donanım özellikleri ve proje özellikleridir.

Soru 3: Bu gruplandırma işleminin sonuncuda elde edilen sonuçların gruplandırma yapılmadan geliştirilen yapay sinir ağları modeline göre daha ortaya koyduğu düşünüldüğünde burada rol alan gruplandırma şemantiği genelleştirilebilir mi?

Cevap 3: Veri setinin özelliklerini gruplandırmadan ile oluşturulan model %14'lük bir hata payına sahipken, Veri setinin özelliklerini gruplandırılarak oluşturulan veri seti %9'luk bir hata payına sahiptir. Bu durum da veri setinin özelliklerinin gruplandırılması ile oluşturulan YSA'nın daha başarılı olduğunu göstermektedir. Özelliklerin gruplandırılmasında göz önünde bulundurulması gereken ana etken benzer özelliklerin doğru bir şekilde gruplandırılmasıdır.

"Yapay sinir ağları tabanlı model, Geliştirilen yapay sinir ağları tabanlı yazılım maliyet tahmini modelinde daha güçlü veri setlerinin oluşturulması durumunda ve bunun üzerinde koşturulacak farklı yapay sinir ağları çözümleri ile daha iyi sonuçların üretilmesi durumunda önerdiğimiz model güncelliğini kaybedecektir."

Oluşturduğumuz modelin eksik yani sadece çevresel faktörleri ve kod satısı sayısını giriş değeri olarak almasıdır. Kod satırı sayısının tespiti gelişen programlama dilleri ile beraber göreceli hale gelebilmektedir. Bu nedenle kod satırı sayısını da objektif bir şekilde tespit edecek bir sistemin geliştirilmesi ile oluşturduğumuz model son yazılımsal gelişmeler ile beraber çok rahat kullanılabilir. Böylece yazılım geliştiriciler ve karar vericiler için önemli bir araç olarak kullanılabilecektir. Ayrıca

güncel veri setleri üzerinde yapılacak tespitler (kod satırı uyarlaması, Çevresel faktörlerin uyarlanması) ile oluşturulan modelin başarısı güncel projelerde de tespit edilebilir. Bunu da ileriki çalışmalarda yapmayı planlıyoruz.

Geliştirdiğimiz İkinci model regresyon tabanlı modeldir. Bu model kod satırı sayısını ve çevresel faktörleri hesaba katarak maliyet kestirimi yapmaktadır. Bu modelde 6 farklı regresyon tabanlı model kullanılmıştır. Bu modelin ürettiği sonuçları ve bugüne kadar yapılmış bazı regresyon tabanlı maliyet kestirim modellerine karşı başarısı sonuçlar kısmında gösterilmiştir. Özellikle çoklu polinomsal regresyon modelinin ürettiği sonuçların bugüne kadar yapılan çalışmalara kıyasla daha iyi olduğu şekiller ve Tabloler yardımı ile sonuç kısmında ayrıntılı bir şekilde sunulmuştur. Özellikle çoklu polinomsal regresyon ile yapılan çalışma sayısının kısıtlı olduğu görülmektedir. Buda oluşturduğumuz modelin yenilikçi yönünü ortaya koymaktadır. Özellikle çoklu polinomsal regresyon modelinde fonksiyonun derecesi ne kadar artarsa oluşturulan denklemlerin çözümü de o kadar zorlaşacaktır. Bugüne kadar yapılan kısıtlı çalışmalarda fonksiyon derecesi olarak 2 ya da 3 değerleri seçilmiştir. Biz ise bu modelde 6. dereceye kadar olan değerleri test ettik. Fonksiyonun derecesinin en iyi olduğu değerin 6 olduğunu tespit ettik. Geliştirilen bu model vesilesi ile Amaç kısmında belirtilen sorulara aşağıdaki cevapları verdik.

Soru 4: Regresyon tabanlı yazılım maliyet tahmininde farklı regresyon algoritmaları kullanıldı. Neden regresyon tabanlı algoritmalar tercih edildi?

Cevap 4: Geliştirilecek olan bir yazılımın maliyeti yapılan işe büyüklüğüne göre farklı değerler almaktadır. Bu nedenle geliştirilecek olan çözümlerin regresyon tabanlı olması gerekmektedir.

Soru 5: Regresyon tabanlı yazılım maliyet tahmini modelinde kullanılan algoritmalardan en başarılı sonucu ortaya koyan algoritma hangisidir? Bu algoritmanın başarılı olmasında etkili olan faktörler hangileridir?

Cevap 5: Regresyon tabanlı yazılım maliyet tahmini modelinde kullanılan algoritmalardan en başarılı sonucu ortaya koyan algoritma Çoklu Polinomsal Regresyon algoritmasıdır. Bu algoritmanın başarılı olmasında birden fazla giriş

değerine sahip olması ve elde edilen fonksiyonun derecesinin yüksek olması belirleyici olmuştur.

Soru 6: Geliştirilen Regresyon tabanlı yazılım maliyet tahmini modeli bugüne kadar geliştirilen regresyon tabanlı yazılım maliyet tahmini modellerine göre üstünlüğü ve getirdiği yenilik nedir?

Cevap 6: Geliştirdiğimiz çoklu polinomsal regresyon modelinde elde edilen fonksiyonun derecesi 6'dır. Bugüne kadar yapılan çalışmalar kısıtlı olmakla beraber elde edilen fonksiyonların derecesi en fazla 3'tür. Bu nedenle elde ettiğimiz fonksiyon daha başarılı sonuçlar vermektedir.

Soru 7: Geliştirilen Regresyon tabanlı yazılım maliyet tahmini modeli farklı veri setleri üzerinde test edildiğinde aynı başarıyı ortaya koyabilecek midir?

Cevap 7: Geliştirilen regresyon tabanlı model üç farklı veri seti olan cocomo81, cocomonasa ve cocomonasa2 veri setleri üzerinde test edilmiştir. Aynı başarıyı ortaya koymuştur.

"Geliştirilen Regresyon tabanlı yazılım maliyet tahmini modelinde örnek sayısının fazla olduğu veri setlerinin oluşturulması durumunda bu veri setleri üzerinde koşturulacak algoritmalar ile daha doğru sonuçlar üreten regresyon fonksiyonları elde edilebilir. Bu durumda önerdiğimiz model güncelliğini kaybeder."

Bundan sonraki çalışmalarda regresyon tabanlı oluşturduğumuz model için kod satırı sayısını tespit eden bir model oluşturup regresyon modellerine giriş değeri olarak vermeyi planlıyoruz. Böylece karar vericiler ve proje yöneticileri gerçekleştirecekleri projede yapacakları modülü seçtikten sonra sistem ilgili fonksiyonu çalıştırarak maliyet tahmini yapabileceklerdir. Oluşturduğumuz modelin eksik yani sadece çevresel faktörleri ve kod satısı sayısını giriş değeri olarak almasıdır. Kod satırı sayısının tespiti gelişen programlama dilleri ile beraber göreceli hale gelebilmektedir. Bu nedenle kod satırı sayısını da objektif bir şekilde tespit edecek bir sistemin geliştirilmesi ile oluşturduğumuz model son yazılımsal gelişmeler ile beraber çok rahat kullanılabilir. Böylece yazılım geliştiriciler ve karar vericiler için önemli bir araç olarak kullanılabilecektir. Ayrıca güncel veri setleri üzerinde yapılacak tespitler (kod satırı uyarlaması. Çevresel faktörlerin

uyarlanması) ile oluşturulan modelin başarısı güncel projelerde de tespit edilebilir. Bunu da ileriki çalışmalarda yapmayı planlıyoruz.

Geliştirdiğimiz üçüncü model ise Scrum ortamında geliştirilen yazılımlar içim maliyet tahmini yapan modeldir. Bu model yazılımın geliştirildiği ortamı dikkate alarak maliyet tahmini yapmaya çalışmaktadır. Bu model ürettiği sonuçlar ile bugüne kadar yapılan çalışmaların ortaya koyduğu sonuçlar karşılaştırmalı olarak sonuçlar kısmında ayrıntılı bir şekilde gösterilmiştir. Oluşturduğumuz modelin en güçlü yanı yazılım maliyetini en küçük parçaya kadar indirgemesi, küçük iş parçalarını belirlenen fazlara atması ve kullandığı özelleştirilmiş toplam fonksiyonudur. Scrum ile geliştiren yazılımlar için maliyet tahmini yapan diğer çalışmalarda daha çok yazılımı bir bütün olarak ele alıp maliyeti bu şekilde hesaplamalarıdır. Bizim projemizde ise maliyet faz faz ele alınıp hem daha doğru tahminler elde edildi hem de karar vericiler için bir fazda oluşabilecek sapmaları diğer fazlarda telafi edebilme imkânı sağlandı. Bu bağlamda bu çalışama bu özelliği ile bugüne kadar yapılan çalışmalardan ayrılmaktadır. Geliştirilen bu model vesilesi ile Amaç kısmında belirtilen sorulara aşağıdaki cevapları verdik.

Soru 8: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde neden regresyon tabanlı makine öğrenmesi algoritmaları kullanıldı? Sınıflandırma tabanlı algoritmalar kullanılmaz mıydı?

Cevap 8: Geliştirilen yazılım büyüklüğü yapılan işe göre değişmektedir. Bu nedenle giriş değerlerine bağlı olarak elde edilecek olan çıkış değerleri farklı olacaktır. Bu nedenle burada sınıflandırma tabanlı algoritmalar kullanılamaz.

Soru 9: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde fazlara ayrılarak çalışan bir mimari geliştirilmiştir. Fazlara ayrılma işlemi geliştirilen modelin başarısını nasıl etkilemiştir?

Cevap 9: Fazlara ayrıştırma işlemi geliştirilen modelin başarısını pozitif yönde etkilemiştir. Bunuı basit bir örnek ile açıklayacak olursak. MongoDb üzerinde bulunan iş parçalarının toplam Story Point değeri 15264'tür. Fazlama işlemi yapılmadan SVR algoritması koşturulduğunda 14119 Story Point değerini üretmekte, fazlama işlemi yapıldığında ise 14917 story point değerini üretmektedir.

Fazlama işlemi yapılarak elde edilen değer gerçek değere daha yakındır. Fazlama işlemi yapılmadan rastegele orman algoritması koşturulduğunda 13907 story point değerini üretmekte, fazlama işlemi yapıldığında ise 14928 story point değerini üretmektedir. Fazlama işlemi yapılmadan gradyan artırma algoritması koşturulduğunda 14349 story point değerini üretmekte, fazlama işlemi yapıldığında ise 15233 story point değerini üretmektedir. Fazlama işlemi yapılmadan çok katmanlı perseptron algoritması koşturulduğunda 13751 story soint değerini üretmekte, fazlama işlemi yapıldığında ise 14122 story point değerini üretmektedir. Bu durum ile de açıkça anlaşılacağı üzere fazlama işlemi işle toplam gerçek story point değerine daha yakın değerler elde edilmiştir.

Soru 10: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde Fazlara ayrılma işlemi hangi kriterlere göre gerçekleştirilmiştir?

Cevap 10: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde Fazlara ayrılma işlemi yazılım geliştirme metodolojilerinin klasik yapısı göz önünde bulundurularak yapılmıştır. Klasik yazılım geliştirme süreçleri 5 tane temel aşamadan oluşur. Bu adımlar; Analiz, Tasarım, Geliştirme, Test ve Bakımdır. Bu adımlar göz önünde bulundurularak 4 tane faz oluşturulmuştur. İlk faz veri tabanı ve alt modül geliştirme fazıdır. İkinci faz servis ve kullanıcı modüllerinin geliştirilmesi fazıdır. Üçüncü faz test be beta test fazıdır. Dördüncü faz ise Bakım fazıdır.

Soru 11: Scrum ile geliştirilen yazılımlar için yazılım maliyet tahmini yapan modelde temel de aritmetik tabanlı Story Point kullanılmıştır. Aritmetik gösterim yerine farklı bir gösterim kullanılması durumunda maliyet hesaplanmasında nasıl bir başarı ortaya çıkabilir?

Cevap 11: Aritmetik tabanlı gösterim yerine farklı gösterimlerin kullanılması da yapılacak olan tahminin başarısında önemli rol oynamaktadır. Artimetik gösterim ile SVR jira veri seti üzerinde 80%'lik bir başarı ortaya koymuştur. Fibonacci gösterimi ile 95%'lik bir başarı ortaya koymuştur. Artimetik gösterim ile rastgele orman algoritması jira veri seti üzerinde 79%'lik bir başarı ortaya koymuştur. Fibonacci gösterimi ile 82%'lik bir başarı ortaya koymuştur. Artimetik gösterim ile gradyan artırma algoritması jira veri seti üzerinde 86%'lik bir başarı ortaya

koymuştur. Fibonacci gösterimi ile 98%'lik bir başarı ortaya koymuştur. Artimetik gösterim ile çok katmanlı perseptron çözümü jira veri seti üzerinde 80%'lik bir başarı ortaya koymuştur. Fibonacci gösterimi ile 87%'lik bir başarı ortaya koymuştur. Fibonacci gösterimi ile çok daha iyi sonuçların ortaya çıktığı gözlemlemiştir. Bu durumda gösteriyor ki farklı story point gösterimi de başarının oluşmasında önemli bir etkiye sahiptir.

"Geliştirilen makine öğrenmesi tabanlı yazılım maliyet tahmini modelinde birçok makine öğrenmesi algoritması kullanıldı. Fakat kullandığımız algoritmaların dışında bir algoritmanın kullanılması ve kullandığımız algoritmalardan başarılı bir sonuç üretmesi durumunda önerdiğimiz model güncelliğini kaybeder."

"Geliştirilen makine öğrenmesi tabanlı yazılım maliyet tahmini modelinde 4 farklı faz geliştirilerek model geliştirilmiştir. Bu fazlama işlemi yazılım geliştirilme metodolojileri göz önünde bulundurularak gerçekleştirilmiştir. Burada daha ayrıntılı bir fazlama yapılarak ve bütün iş parçalarının doğru iterasyonlara atılması durumunda başarı oranı yükselecektir. Bu durumda geliştirdiğimiz model güncelliğini kaybedecektir."

"Geliştirilen makine öğrenmesi tabanlı yazılım maliyet tahmini modelinde efor tahmininde story pointler kullanılmıştır. Story pointlerin farklı gösterimleri bulunmaktadır. Biz bu çalışmada en çok kullanılan gösterimlerden olan aritmetik ve fibonacci gösterimlerini tercih ettik. Yeni bir gösterim modelinin geliştirilmesi, aritmetik ve fibonacci gösterimlerinden daha başarılı sonuçlar ortaya koyması durumunda önerdiğimiz model güncelliğini kaybedecektir."

Bunun yansıra ileriki çalışmalarda sektörün önde gelen kuruluşları için JIRA üzerinde kullanabilecekleri entegre bir modül geliştirilebilir. Bu modülde kullanılacak özellikler bu alanda uzman kişilerin görüşleri alınarak belirlenebilir. Bundan sonra oluşan veri seti üzerinde geliştirdiğimiz modeli test edip kullanabiliriz. Geliştirdiğimiz bu model Fibonacci ve aritmetik gösterim kullanıldı. Fibonacci gösterimi ile daha iyi sonuçların elde edildiği görüldü. Bundan sonraki çalışmalarda yazılım ve bilişim sektörünün ileri gelenleri ile delphi tekniği kullanılarak yapılacak bir çalışma ile yeni notasyonlar için çözümler geliştirilebilir. Geliştirilen bu notasyon ile Scrum tekniği kullanılarak geliştirilen yazılımlar için

daha iyi bir maliyet kestirimi modeli geliştirilebilir. Bu Tez çalışmasında bütüncül bir yazılım maliyet tahmini için üç farklı model ortaya konuldu. Bu konuda yapılan çalışmalara ve geliştirilen modeller ayrıntılı bir şekilde açıklandı. Geliştirilen modellerin bugüne kadar yapılan çalışmalara karşı başarısı şekiller ve Tabloler yardımı ile gösterildi. Son olarak ise geliştirilen modellerde ileriki zamanlarda nasıl iyileştirmeler yapılabileceği konusu tartışıldı.

Yazılım maliyetinin hesaplanması kendi içinde küçük bir projedir. Yazılım projelerinin başarıya ulaşmasında projeye başlamadan önce yapılan maliyet kestirimleri projenin başarıya ulaşmasında çok önemlidir. Bu tez çalışmasında yazılım maliyetine bütüncül bir çözüm sunmaya çalıştık. Sadece kod satırı sayısına bakarak maliyet hesaplamanın olabileceği durumları değil yazılımın geliştirildiği ortamı da hesaba katarak yazılım maliyet hesaplaması yaptık. Bu nedenle üç farklı yazılım maliyet kestirim modeli ortaya koyduk.

Kaynakça

[1] Oya K. and Murat A., Yazilim Gelitirme Projelerinde Maliyet Tahminleme Çalimasinda Kullanilabilinecek Bir Ölçev Kümesi ve Bir Yapay Sinir Aği Topolojisi Önerisi, 2018.

[2] Reddy C. S., Raju K., Concise neural network model for estimating software effort, International Journal of Recent Trends in Engineering, 1-188-193, 2009.

[3] Boehm, B. W., Software Engineering Economics, Prentice Hall, 1981.

[4] Fedotova O., Teixeira L. and Alvelos H., Software Effort Estimation with Multiple Linear Regression: Review and Practical Application, J. Inf. Sci. Eng., 29(5), 925-945.,2013

[5] Georgia M., Kapitsaki and Marios C., Where Is Scrum in the Current Agile World? ,9th International Conference on Evaluation of Novel Approaches to Software Engineering. Pages 101-108, DOI: 10.5220/0004867701010108 ,2014.

[6] Beck, K., et al. (2001) The Agile Manifesto. Agile Alliance. http://agilemanifesto.org/

[7] B. Michael, S. Blumberg and J. Laartz, "Delivering large-scale IT projects on time, on budget and on value", Tech. Rep., 2012.

[8] Hu Y., Zhang X., Ngai E. W. T., Cai R. and Liu M., Software project risk analysis using Bayesian networks with causality constraint: Decision Support Systems, 56, 439-449.,2013.

[9] The CHAOS Manifesto, Think Big, Act Small by The Standish Group International, 2013.

[10] Wanjala Munialo, Samson and Muketha Geoffrey, A Review of Agile Software Effort Estimation Methods, International Journal of Computer Applications Technology and Research, 5. 612-618. 10.7753/IJCATR0509.1009, 2016.

[11] PMBOK (2000). A Guide to the Project Management Body of Knowledge, Project Management Institute, 2000.

[12] Conchúir E. Ó., Ågerfalk P. J., Olsson H. H. and Fitzgerald B., Global software development: where are the benefits? Communications of the ACM, 52(8), 127-131,2009.

[13] NASA– Handbook for Software Cost Estimation, NASA, Jet Propulsion Laboratory, Pasadena, California, 2003.

[14] McConnell, S., Software Estimation: Demystifying the Black Art, Microsoft Press, Redmond, Washington, 2006.

[15] Brooks F., The Mythical Man-Month, Addison-Wesley, Reading, MA, United States, 1975.

[16] Heemstra F. J., "Software cost estimation". Information and Software Technology, 34(10), 627-639.,1992.

[17] A. Pohjolainen Pentti., The Development of Software Testing in Finland,1950–2000. 303. 271-282. 10.1007/978-3-642-03757-3_28, 2009

[18] Chulani, S., Boehm, B., & Steece, B., "Calibrating Software Cost Models Using Bayesian Analysis," IEEE Transactions on Software Engineering, Special Issue on Empirical Methods in Software Engineering, 1999.

[19] Freiman, F. R., & Park, R. E., The PRICE Software Cost Model. RCA Price Systems, Cherry Hill, NJ, 1979.

[20] Rubin, H., Interactive macro-estimation of software life cycle parameters via personal computer: a technique for improving customer/developer communication, in Proc. Symp. on application & assessment of automated tools for software development, IEEE, San Francisco, (1983).

[21] Jensen, R., An Improved Macrolevel Software Development Resource Estimation Model, Proceedings 5th ISPA Conference, 88-92, 1983.

[22] Narendra Sharma, Aman Bajpai, Mr. Ratnesh Litoriya, A Comparison of software cost estimation methods: A Survey, The International Journal of Computer Science & Applications (TIJCSA), No. 3, May 2012.

[23] Putnam L. H., A general empirical solution to the macro software sizing and estimating problem, IEEE transactions on Software Engineering, (4), 345-361, 1978

[24] Albrecht, A. J., Measuring application development productivity, Proceeding of the Joint SHARE, GUIDE and IBM application development symposium, IBM Corporation, 1979.

[25] Boehm B., Abts C. A., Chulani S., Clark B. K., Horowitz E., Madachy R., Reifer D. J. and Steece B., Software cost estimation with COCOMO II, Prentice Hall,2000

[26] Hihn J. And Habib Agahi H. Cost estimation of software intensive projects: A survey of current practices, In Proceedings of the 13th international conference on Software engineering (pp. 276-287). IEEE Computer Society Press, 1991

[27] Yogi, Nageswara, Delphi Technique for the Software Effort Estimation -- An outline for an Expert Judgment Method, International Journal of Engineering and Technical Research, 2015.

[28] Haugan G. T., Effective work breakdown structures, Berrett-Koehler Publishers,2001

[29] MIL-HDBK-881, Work Breakdown Structures for Defense Material Items, Department of Defense, United States of America,1998

[30] Shepperd M., Schofield C. and Kitchenham B., Effort estimation using analogy, Proceedings of the 18th international conference on Software engineering (pp. 170-178), IEEE Computer Society.

[31] Roberts, P., Guide to project management, Profile Books Limited/The Economist, 1997.

[32] Wolverton R. W., The cost of developing large-scale software, IEEE Transactions on Computers, 100(6), 615-636, 1974

[33] Idri A., Abran A. and Khoshgoftaar T. M., Fuzzy case-based reasoning models for software cost estimation, Int. Soft. Computing in Software Engineering (pp. 64-96). Springer Berlin Heidelberg,2004

[34] Boehm B., Abts C. and Chulani S., Software development cost estimation approaches—A survey. Annals of software engineering, 10(1-4), 177-205.

[35] Karunanithi N., Whitley D. and Malaiya Y. K., Using neural networks in reliability prediction, IEEE Software, 9(4), 53-59,1992

[36] Gray A. R. and MacDonell S. G., A comparison of techniques for developing predictive models of software metrics, Information and software technology, 39(6), 425-437,1997

[37] Wittig G. E. and Finnic G. R., Using artificial neural networks and function points to estimate 4GL software development effort, Australasian Journal of Information Systems, 1(2).1994

[38] Finnie G. R. and Wittig G. E., AI tools for software development effort estimation, Int. Software Engineering: Education and Practice. Proceedings. International Conference (pp. 346-353). IEEE.

[39] Forrester J., Industrial Dynamics, MIT Press, Cambridge,1961

[40] Madachy R., A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment, Doctoral dissertation, University of Southern California, United States.1994

[41] Brooks F., The Mythical Man-Month. Reading, Massachusetts,2006.

[42] Richardson G. P., System dynamics: Simulation for policy analysis from a feedback perspective. In Qualitative simulation modeling and analysis (pp. 144-169), Springer, New York, NY, 1991

[43] Abdel-Hamid T. K. and Madnick S. E., Software project dynamics: an integrated approach, Englewood Cliffs, NJ: Prentice Hall,1991.

[44] Abdel-Hamid T. and Madnick S., Modeling the dynamics of software reuse: An integrating system dynamics perspective, Int. Presentation to the 6th Annual Workshop on Reuse, Owego, NY, November 1993.

[45] Keating G., The production and use of economic forecasts ,907, Routledge, 1985.

[46] Briand L. C., El Emam K., and Bomarius F., COBRA: a hybrid method for software cost estimation, benchmarking and risk assessment, Int. Software Engineering Proceedings of the International Conference, (pp. 390-399). IEEE,1998

[47] Ruhe M., Jeffery R.and Wieczorek I., Cost estimation for web applications in Software Engineering, Proceedings, 25th International Conference on (pp. 285-294). IEEE,2003.

[48] Trendowicz A., Heidrich J., Münch J., Ishigai Y., Yokoyama K. and Kikuchi N., Development of a hybrid cost estimation model in an iterative manner, In Proceedings of the 28th international conference on Software engineering (pp. 331-340). ACM. 2006.

[49] Schwaber Ken and Beedle Mike, Agile Software Development with Scrum. 2001

[50] Ardant E.L., Scrum is evil, http://ericlefevre.net/wordpress/2008/10/07/Scrum-is-evil/, 2008.

[51] Sutherland J. V. and Schwaber K., The SCRUM methodology in Business object design and implementation: OOPSLA workshop, 1995.

[52] Jiménez Victor, Afonso Paulo and Fernandes Gabriela, Project Management and Costing Systems: ABC SCRUM, 2018.

[53] Sutherland J.and Schwaber K., The Scrum guide. the definitive guide to Scrum: The rules of the game, ScrumGuides.com, 2013.

[54] Cohn M., Advice on conducting the Scrum of Scrums meeting. Mountain Goat Software Web site, 2007.

[55] Cho L., Adopting an Agile Culture A User Experience Team's Journey, Int. Agile Conference, AGILE'09. (pp. 416-421). IEEE, 2009.

[56] Satpathy T., A guide to the Scrum body of knowledge, 2013.

[57] J. Sayyad Shirabad and T.J. Menzies, The PROMISE Repository of Software Engineering Databases, School of Information Technology and Eng., Univ. Of Ottawa, http:// promise.site.uottawa.ca/SERepository, 2005,

[58] Barry Boehm, http://sunset.usc.edu/research/COCOMOII/Cocomo81_pgm/

[59] Choetkiertikul Morakot, Dam Hoa, Tran Truyen, Pham Trang, Ghose Aditya and Menzies Tim, A deep learning model for estimating Story Points, IEEE Transactions on Software Engineering. PP. 10.1109/TSE.2018.2792473, 2016.

[60] Gunning Robert, The Fog Index after Twenty Years. Journal of Business Communication- J Bus Comm, 6. 3-13. 10.1177/002194366900600202, 1969.

[61] S. Mandt, M. D. Hoffman, and D. M. Blei., A variational analysis of stochastic gradient algorithms, Int. ICML, 2016.

[62] Tao H., Hou C., Nie F., Zhu J. and Yi D., Scalable Multi-View Semi-Supervised Classification via Adaptive Regression, IEEE Transactions on Image Processing, 26(9), 4283-4296, 2017.

[63] Xu C., Tao D. and Xu C., Multi-view learning with incomplete views, IEEE Transactions on Image Processing, 24(12), 5812-5825, 2015.

[64] Peng X.and Huang Z., A Novel Popular Tourist Attraction Discovering Approach Based on Geo-Tagged Social Media Big Data, ISPRS International Journal of Geo-Information, 6(7), 216, 2017.

[65] Zhang X., Yu M., Wang L., Xiang S.and Liu C. L., Retargeted least squares regression algorithm, IEEE transactions on neural networks and learning systems, 26(9), 2206-2213, 2015.

[66] Zhen X., Yu M., Zheng F., Nachum I. B., Bhaduri M., Laidley D. and Li S., Multitarget Sparse Latent Regression, IEEE transactions on neural networks and learning systems. 2017.

[67] Zhen X., Yu M., He X. and Li S., Multi-target regression via robust low-rank learning, IEEE transactions on pattern analysis and machine intelligence, 40(2), 497-504, 2018.

[68] Naseem I., Togneri R. and Bennamoun M., Linear regression for face recognition, IEEE transactions on pattern analysis and machine intelligence, 32(11), 2106-2112, 2010.

[69] Wright J., Yang A.Y., Ganesh A., Sastry S. S.and Ma Y., Robust face recognition via sparse representation, IEEE transactions on pattern analysis and machine intelligence, 31(2), 210-227, 2009.

[70] Hou C., Jiao Y., Nie F., Luo T.and Zhou Z. H., 2D Feature Selection by Sparse Matrix Regression, IEEE Transactions on Image Processing, 26(9), 4255-4268, 2017.

[71] Gui J., Sun Z., Ji S., Tao D.and Tan T., Feature selection based on structured sparsity: A comprehensive study, IEEE transactions on neural networks and learning systems, 28(7), 1490-1507, 2017.

[72] Knaub Jr, J. R., Weighted multiple regression estimation for survey model sampling, InterStat, 1996.

[73] Carroll R. J., Transformation and weighting in regression, Routledge, 2017.

[74] Ryan T. P., Modern regression methods, 655, John Wiley & Sons, 2008.

[75] Swain J. J., Venkatraman S. and Wilson J. R., Least-squares estimation of distribution functions in Johnson's translation system, Journal of Statistical Computation and Simulation, 29(4), 271-297, 1988.

[76] Csató L., A characterization of the logarithmic least square's method, arXiv preprint arXiv:1704.05321, 2017.

[77] Fichtner J., Some thoughts about the mathematics of the analytic hierarchy process, Inst. für Angewandte Systemforschung u. Operations-Research, 1984.

[78] De Graan J. G., Extentions to the Multiple Criteria Analysis Method of TL Saaty., National Institute for Water Supply, 1980.

[79] Bozóki S.and Tsyganok V., The logarithmic least squares optimality of the geometric mean of weight vectors calculated from all spanning trees for (in) complete pairwise comparison matrices, arXiv preprint arXiv:1701.04265, 2017.

[80] Patriota Alexandre, A q-Exponential regression model, Sankha ser. B. 74. 149-170. 10.1007/s13571-012-0051-2, 2012.

[81] Schreiber Peter, Forbrich Inke, Kutzbach Lars, Hormann A., Wolf Ulrike, Miglovec M., Pihlatie Mari, Christiansen Jesper and Wilmking Martin, A comparison of linear and exponential regression for estimating diffusive methane fluxes by closed-chamber-results from laboratory and field campaigns, 2009.

[82] Adalier O., Uğur A., Korukoğlu S. and Ertaş K., A new regression-based software cost estimation model using power values, In International Conference on Intelligent Data Engineering and Automated Learning (pp. 326-334). Springer, Berlin, Heidelberg, 2007.

[83] Rolfe T. J., Least Squares Fitting of Polynomials and Exponentials with Programming Examples, Mathematics and Computer Education, 16(2), 122-32, 1982.

[84] Ostertagova Eva, Modelling Using Polynomial Regression. Procedia Engineering, 48. 500–506. 10.1016/j.proeng.2012.09.545, 2012.

[85] A. Stimson James, Carmines Edward, A. Zeller Richard, Interpreting Polynomial Regression. Sociological Methods & Research- SOCIOL METHOD RES. 6. 515-524. 10.1177/004912417800600405, 1978.

[86] Kocaguneli E., Menzies T., Bener A. and Keung J.W., Exploiting the Essential Assumptions of Analogy-based Effort Estimation, Journal of IEEE Transactions on Software Engineering, 34, No.4, pp. 471-484, July 2010.

[87] Chiu N. And Huang S., The adjusted analogy-based software effort estimation based on similarity distances, Journal of Systems and Software. 80. 628-640. 10.1016/j.jss.2006.06.006, 2006.

[88] Ilango T. and Murugavalli S., Analogy based software effort estimation based on differential evolution and hybrid fuzzy logic and firefly algorithm, 15. 1484-1493. 10.3923/ajit.2016.1484.1493.

[89] Passakorn P., Robust comparison of similarity measures in analogy-based software effort estimation, Software Knowledge Information Management and Applications (SKIMA) 2017 11th International Conference on, pp. 1-7, 2017.

[90] Odin C., Passakorn P., Yasutaka K., Pattara L., Arnon R., NaoyasuU. and Kenichi M. A., review and comparison of methods for determining the best analogies in analogy-based software effort estimation, Proceedings of the 31st Annual ACM Symposium on Applied Computing, April 04-08, Pisa, Italy, 2016.

[91] Popli R.and Chauhan N., Cost and effort estimation in agile software development. ICROIT 2014- Proceedings of the International Conference on Reliability, Optimization and Information Technology, 57-61. 10.1109/ICROIT.2014.6798284, 2014.

[92] Łabędzki M., Promiński P., Rybicki A.and Wolski M., Agile effort estimation in software development projects – case study, Central European Review of Economics and Management, 1, No. 3, 135-152, September 2017.

[93] Jitender C. and Ugrasen S., Story Points Based Effort Estimation Model for Software Maintenance, Procedia Technology, 4, 761-765, https:// doi.org /10.1016/ j.protcy. 2012.05.124, 2012.

[94] Boehm B., Abts C. A., Chulani S., Clark B. K., Horowitz E., Madachy R., Reifer D. J. and Steece B., Software cost estimation with COCOMO II, Prentice Hall, 2000.

[95] Putnam L., A general empirical solution to the macro software sizing and estimating problem, IEEE Transaction on Software Engineering, pp. 345-361, July 1978.

[96] Mohammed Usman, Emilia M., Francila W. and Ricardo B., Effort Estimation in Agile Software Development: A Systematic Literature Review, ACM,2014.

[97] Greening Daniel, Enterprise Scrum: Scaling Scrum to the Executive Level. 1- 10, 10.1109/HICSS.2010.186, 2010.

[98] Evita Coelho and Anirban Basu, Effort Estimation in Agile Software Development using Story Points, International Journal of Applied Information Systems (IJAIS), No.7, August 2012.

[99] Bilgaiyan S., Mishra S. and Das M., A Review of Software Cost Estimation in Agile Software Development using Soft Computing Techniques, 2nd International Conference on Computational Intelligence and Networks, IEEE, pp. 112–117, 2016.

[100] Alaa-El-deen Hamouda, Using Agile story points as Estimation Techniques in CMMI Organization, Agile Conference. 2014.

[101] Aditi Panda, Satapathy S. M. and Santanu Kumar R., Empirical Validation of Neural Network Models for Agile, Science Direct, Procedia Computer science, 2015.

[102] Jitender C. and Dr. Ugrasen Suman, Story Point based Effort Estimation Model for Software Maintenance, Science Direct, Procedia Technology 761-765, 2012.

[103] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Trany, Trang Phamy, Aditya Ghose and Tim Menzies, A Deep Learning Model for Estimating Story Points, [CS.SE] 6 sep. 2016.

[104] Simon P., Alessamdro M., Serge D., Michele M. and Roberto T., Estimating Story Points from Issue Reports, ACM, 2016.

[105] Rashmi P. and Naresh C., Estimation in Agile Environment using Resistance Factor, IEEE, 2014.

[106] Christophe C., Alain Abran, Rachida D., Effort Estimation with Story Points and COSMIC Function Point- An Industry Case Study, Software Measurement News, 21, No.1, Pages 25-36, 2016.

[107] Satapathy S.M., Aditi Panda and Santanu Kumar R., Story Point Approach based Agile Software Effort Estimation using various SVR Kernel methods.

[108] Hindh Z., Mohammed A., Adjusting Story Point Calculation in Scrum Effort and Time Estimation, IEEE, 2015.

[109] Binish T., Liliana G. and Martin U.L.F., Understanding and improving Effort Estimation in Agile Software Development, -An Industry Case Study, ACM, 2016.

[110] Yves W., Samedi H., Manuel K., Isabelle M. and Stephen P., Building a Rationale Diagram for Evaluating User Story Search., IEEE, 2016.

[111] Raza Ahmed Ali, Tayyab Muhammad, Bhatt Shahid, J. Alzahrani, Abdullah Babar and Muhammad Imran, Impact of Story Point Estimation on Product using Metrics in Scrum Development Process, International Journal of Advanced Computer Science and Applications. 8. 7. 10.14569/IJACSA.2017.080452, 2017.

[112] Breiman L., Machine Learning, https://doi.org/10.1023/ A:1010933404324, 2001.

[113] Popescu M., Balas V.E., Perescu- L. and Mastorakis N., Multilayer Perceptron and Neural Networks, Wseas Transactions on Circuits and Systems, Issue 7, July 2009.

[114] Mielniczuk Jan and Tyrcha Joanna, Consistency of multilayer perceptron regression estimators, Neural Networks, 6. 1019-1022. 10.1016/S0893-6080(09)80011-7, 1993.

[115] Anjali Sharma K., Empirical validation of random forest for Agile software effort estimation based on Story Points. International journal of engineering sciences & research technology, 5(7), 1437–1446. http://doi.org/10.5281/zenodo.58604 , 2016.

[116] Choetkiertikul Morakot, Dam Hoa, Tran Truyen, Pham Trang, Ghose Aditya and Menzies Tim, A deep learning model for estimating Story Points. IEEE Transactions on Software Engineering. PP. 10.1109/TSE.2018.2792473, 2016

[117] Smith P., Ganesh S. and Liu P., A Comparison of Random Forest Regression and Multiple Linear Regression for Prediction in Neuroscience. Journal of neuroscience methods. 220. 10.1016/j.jneumeth.2013.08.024, 2013.

[118] Satapathy, Shashank Mouli and Rath Santanu, Empirical assessment of machine learning models for Agile software development effort estimation using Story Points, Innovations in Systems and Software Engineering. 1-10. 10.1007/s11334-017-0288-z, 2017.

[119] Krawczak Maciej, Introduction to Multilayer Neural Networks. 10.1007/978-3-319-00248-4_1, 2013.

[120] Ziauddin Zia, Khan Abdur Rashid, Zaman Khair, Software cost estimation for component based fourth-generation-language software applications. Software, IET. 5. 103- 110. 10.1049/iet-sen.2010.0027, 2011.

[121] Van Koten Chikako, A comparison of software effort prediction models using small datasets, 2018.

[122] Serluca C., An Investigation Into Software Effort Estimation using a Back-Propagation Neural Network, M.Sc. Thesis, Bournemouth University, 1995.

[123] Srinivasan K. and D. Fisher, Machine Learning Approaches to Estimating Software Dev. Effort. IEEE Transactions on Software Engineering, 21(2), pp126-136, 1995.

[124] Hughes R.T., An Evaluation of Machine Learning Techniques for Software Effort Estimation. University of Brighton, 1996.

[125] LIU Q. and Mintram R.C., Preliminary Data Analysis Methods in Software Estimation, Software Quality Journal, 13, pp: 91-115, 2005.

[126] Stensrud E. and Myrtveit I., Human Performance Estimating with Analogy and Regression Models: An Empirical Validation, Fifth International Symposium on Software Metrics (METRICS'98), 1998.

[127] HU Q., Plant R.T. and Hertz D.B., Software Cost Estimation Using Economic Production Models, Journal of Management Information System, No: 1, pp-143-163, 1998.

[128] Anandhi, V. And Chezian R. Manicka, A Comparison of the forecasting techniques–Arima, Ann and SVM -A Review, International Journal of Computer Engineering and Technology, 0976-6375. 4. 370-376, IAEME, 2013.

Tezden Üretilmiş Yayınlar

İletişim Bilgisi: gultekin.muaz@gmail.com

Makaleler

1. Muaz G. and Oya K., Story Point-Based Effort Estimation Model with Machine Learning Techniques, Int'l Journal of Software Engineering and Knowledge Engineering, May 2019.
2. Muaz G. ve Oya K., Yapay Sinir Ağları Tabanlı Yazılım Efor Tahmini, 1, Sayı:3, Yıl:2016, Sayfa 246-253, ISSN: 2148-3752 ,2018.

Konferans Bildirileri

1. Muaz G. and Oya K., Software Cost Estimation by Using Artificial Neural Networks for Software Developed in Scrum and Traditional Software Development Methodology, IEEE Profes, 2019.
2. Şeyma Nur Kolak, Gizem Çavuş, Muaz Gultekin, Oya Kalipsiz, Çevik Model Kullanan Projelere Ait Ver Seti ile İş Gücü Kestirimi, Ulusal Yazılım Mühendisliği Sempozyumu, UYMS ,2018, İstanbul Türkiye,2018.
3. Muaz G. ve Oya K., Dış Destekli Projelerde Maliyet Tahmini ve Kalite Yönetimi, 2. Ulusal Yönetim Bilişim Sistemleri Kongresi, İstanbul Türkiye,2015.