BANKACILIK ALANINDA KİŞİSEL YAZILIM TEST EFORUNU TAHMİN ETMEK İÇİN PROXY TABANLI BİR METOT VE VAKA ÇALIŞMASI

A PROXY METHOD FOR ESTIMATING PERSONAL SOFTWARE TEST EFFORT IN BANKING DOMAIN AND ITS CASE STUDY

Gizem Kahveci

Prof. Dr. Mehmet Önder Efe
Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin
Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü
YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2017

Gizem Kahveci'nin hazırladığı "**Bankacılık Alanında Kişisel Yazılım Test Eforunu Tahmin Etmek İçin Proxy Tabanlı Bir Metot ve Vaka Çalışması**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Mehmet Önder Efe

Danışman ……

Doç. Dr. Ahmet Burak Can

Üye ……

Yrd. Doç. Dr. Ayça Tarhan

Üye ……

Yrd. Doç. Dr. Fuat Akal

Üye ……

Yrd. Doç. Dr. Engin Demir

Üye ……

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak onaylanmıştır.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU

Fen Bilimleri Enstitüsü Müdürü

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

☒ **Tezimin/Raporumun tamamı dünya çapında erişime açılabilir ve bir kısmı veya tamamının fotokopisi alınabilir.**
(Bu seçenekle teziniz arama motorlarında indekslenebilecek, daha sonra tezinizin erişim statüsünün değiştirilmesini talep etseniz ve kütüphane bu talebinizi yerine getirse bile, tezinin arama motorlarının önbelleklerinde kalmaya devam edebilecektir.)

☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç Kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
(Bu sürenin sonunda uzatma için başvuruda bulunmadığım taktirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı ve ya tamamının fotokopisi alınabilir)

☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını istemiyorum, ancak kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisinin alınmasını onaylıyorum.**

☐ **Serbest Seçenek/Yazarın Seçimi**

03/01/2018

(İmza)

Öğrencinin Adı Soyadı

Gizem IGA AYDIN

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım-kurallarına uygun olarak hazırladığım bu tez çalışmasında,

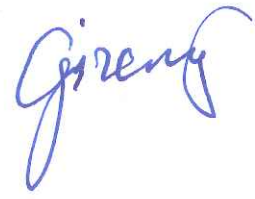
- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçlar bilimsel ahlak kurallarıma uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

29/12/2017

İMZA

Gizem Kahveci



ÖZET

BANKACILIK ALANINDA KİŞİSEL YAZILIM TEST EFORUNU TAHMİN ETMEK İÇİN PROXY TABANLI BİR METOT VE BİR VAKA ÇALIŞMASI

Gizem Kahveci

Yüksek Lisans Tezi, Bilgisayar Mühendisliği

Tez Danışman: Prof. Dr. Mehmet Önder Efe

Aralık 2017, 78 sayfa

Bu çalışmada özel bir bankanın yazılım proje yöneticilerine yönelik olarak, kodlama ve birim testleri tamamlanan yazılım birimlerinin fonksiyonel testleri için gerekecek test süresi/eforunu tahmin ederlerken kullanabilecekleri özgün bir yöntem geliştirilmesi amaçlanmıştır. Bu yöntem ile, bir test analistine atanan her bir yazılım birimi için gerekecek test çalıştırma kişisel eforunun (dolayısıyla süresini) test sürecinin ön safhalarında iken doğru bir şekilde tahmin edilebilmesi sağlanmıştır. Geliştirilen yöntemin temelinde, test analistinin kendi kişisel test sürecini Humphrey'in Kişisel Yazılım Süreci'nde (Personal Software Process - PSP) tanımlandığı gibi modelleyerek ölçümlemesi ve bu ölçümlerin analizi suretiyle kişisel tahmin veritabanını oluşturması bulunmaktadır. Bu yöntemin geliştirme ve sınama çalışmaları için bankanın yazılım uygulamalarından elde edilen gerçek veriler kullanılmıştır. Geliştirilen yöntemin çalışabilirliği ve hassasiyeti test sürecinin ön safhalarında yapılan tahminler ile gerçekleşen efor/süre değerleri karşılaştırılmak suretiyle değerlendirilmiştir. Ortam ve uygulama ile ilgili belli parametrelerin aynı kalması durumunda tahmin hatalarının %12 bandını aşmadığı, çoğunlukla çok daha küçük olduğu görülmüştür.

Anahtar kelimeler: Test süresi tahmini, Test eforu tahmini, Proxy-tabanlı tahmin, Kişisel test süreci, Bankacılık yazılım uygulamaları.

ABSTRACT

A PROXY METHOD FOR ESTIMATING PERSONAL SOFTWARE TEST EFFORT IN BANKING DOMAIN AND ITS CASE STUDY

Gizem Kahveci

Master of Science, Department of Computer Engineering

Supervisor: Prof. Dr. Prof. Dr. Mehmet Önder Efe

December 2017, 78 pages

In this study, a new method was developed to support the software project managers of a private local bank in estimating the required functional test effort/duration of the developed software units. This method allows the project manager to predict the required test run effort (hence duration) of a test analyst for an assigned software unit at the initial test stages. On the basis, the test analyst models and measures her personal test process (PTP) as described in Humphrey's Personal Software Process (or PSP) and generate a personal estimation database by analyzing these measurements. For the development and testing of this method, real data from the bank's software development projects were used. The feasibility and sensitivity of the developed method were evaluated by comparing the estimations made at the earlier stages of the test process with the actual values of test effort. It has been found that if certain parameters related to environment and application remain stable, prediction errors do not exceed 12% band, and in most cases much smaller.

Key Words: Software test estimation, Proxy-based estimation, Personal Test Process, Banking software applications.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

PROBE: Proxy Based Estimating Method

PSP: Personal Software Process

TSP: Team Software Process

LOC: Lines Of Code

ŞEKİL LİSTESİ

TABLO LİSTESİ

GİRİŞ

Sanayinin gelişme kaydetmiş olduğu her sektörde ürün veya hizmet özelinde kalite olgusu gerek üreticiler gerekse tüketiciler için önem teşkil etmektedir [32]. Bu doğrultuda yazılım sektörü de ürün ve hizmet kalitesini arttırmak için yoğun çaba harcanan bir sektördür. Bu çaba yazılım gereklerinin dokümantasyonu aşamasından başlayarak yazılımın kullanımdan çekilmesine kadar yoğun olarak devam etmektedir. Bilhassa günümüzde yazılımların gittikçe başka yazılımlar ile birlikte yazılım eko sistemi halinde çalışma zorunluğunda olması [32] ve sistemdeki bir yazılımın kalitesindeki zafiyetin ekosistem içindeki diğer tüm yazılımları etkilemesi yazılım projelerinde kalite olgusunun önemini daha fazla arttırmıştır.

Kaliteyi sağlamak için kullanılan en önemli araçlardan birisi olan yazılım testleri, yazılımın en olabilecek biçimde, en az hata ve eksik ile üretiminin gerçekleşip müşteriye sunulmasını hedefleyen işlemlerin tamamıdır [32]. Yazılım testleri bir yazılımın sonsuz sayıdaki çalışma alanından, sınırlı sayıda ve uygun şekilde seçilmiş testler ile beklenen davranışlarını karşılamaya yönelik, dinamik olarak yapılan doğrulama faaliyetlerini kapsamaktadır [25]. Yazılım testleri ile müşterinin gereksinimlerini tam manasıyla ve doğru biçimde karşılanırken, aynı zamanda yazılımdaki hataların bulunup ayıklanarak kusursuza yakın ve mümkün olduğunca kaliteli bir ürün ortaya çıkarılması hedeflenir.

Yazılım projelerinde yazılım testleri için harcanan işgücü bir taraftan kaliteyi yükseltirken, diğer taraftan da projenin süre ve maliyetine etki etmektedir. Kullanıcıların beklentilerini karşılayacak yüksek kalitede yazılımlar üretilirken süre ve maliyet sınırlamaları dikkate alınmalıdır.

Proje yöneticileri için yazılım test faaliyetleri için harcanacak zaman ve eforun doğru ve tutarlı bir şekilde tahmini oldukça önemlidir. Zamanında dünyanın ikinci büyük uluslararası havaalanı olan Denver Havaalanı Otomatik Bagaj Sistemi Projesi kapsamında, 186 milyon dolarlık bir yazılımla bagajların otomatik yönetimi sağlanarak 31 Ekim 1993'de açılması planlanmıştır. Ancak bagaj sisteminde ortaya çıkan yazılım hataları nedeniyle sistemin hizmete alınması gecikmeli olarak 28 Şubat 1995 tarihinde gerçekleşmiştir. Bu gecikmenin maliyetinin günlük 1 milyon dolara yakın olduğu ve gecikme nedeniyle oluşan toplam zararın 340 milyon doları bulduğu hesaplanmıştır [27]. Bu projede olduğu gibi gerçekleşen değerlerin tahmin edilen

değerler ile örtüşmemesi projenin ciddi manada başarısız olmasına sebep olabilir. Dolayısıyla, eldeki belli kaynaklar ile test için ayrılacak süre hakkında tutarlı bir tahmin yapabilme, proje yöneticisinin önemli işlevlerinden bir tanesidir. Ancak bu özellikle erken safhalarda, kolay bir iş değildir. Yazılım testlerine ilişkin süre tahmininin zorluğu ve bunun projenin diğer planlama parametrelerine olan doğrudan alakası dolayısıyla proje yöneticilerinin karar verme mekanizmalarını destekleyecek yöntem ve araçlara ihtiyaç duyulmaktadır.

Bu çalışmada bankacılık alanında hizmet vermekte olan özel bir bankanın merkezi yazılım departmanında yürütülen yazılım test faaliyetleri esas alınarak bu doğrultuda yapılan bir üniversite-sanayi işbirliği ile gerçekleştirilen bir yazılım süreç iyileştirme çalışması ve bu kapsamda geliştirilen kişisel test süreci modeli anlatılacaktır. Finans sektöründeki firmaların yazılım ve donanım maliyetleri diğer sektörlere göre daha fazla olmaktadır. Forrester Research kurumunun Amerika, Avrupa ve Asya da yaptığı geniş çaplı araştırmaya göre Bankaların yazılım ve donanım (IT) giderlerinin eş büyüklükteki diğer sektörlere göre 7.3% daha fazla olduğu bulunmuştur. Diğer kaynaklar da, bankalarda yüksek yazılım ve donanım giderleri olduğunu ispatlamaktadır. 2013 yılında, Finansal hizmetler firmalarının yazılım ve donanım faaliyetleri için 270 milyar dolar ile 460 milyar dolar arasında giderleri olacağı düşünülmektedir [11]. Bu kuruluşların maliyetlerinin fazla olmasının nedenleri, sektörün getirmiş olduğu büyük riskleri minimize etme zaruretinin yanında, yazılım geliştirme ekiplerinin verimliliğini artıramamayı da içermektedir. Bankalar aynı zamanda yenilikçilik (inovasyon) kültürüne sahip olan ve mevcut sorunlara çözüm bulma veya farklı yollarla çözümlere daha hızlı ulaşabilmesi maksadıyla bir takım iyileştirmeler ve gelişmelere açık olan kuruluşlardır. Dolayısıyla süreç iyileştirme konusundaki bu tür ar-ge çalışmalarına da destek vermektedirler.

Hedef kuruluşta konu bazında ayrılmış (Nakit Yönetimi, İnternet Şubesi, Mobil Bankacılık, Belge Yönetimi, vb.) farklı geliştirme projelerinden sorumlu Proje Yöneticileri vardır. Bunlara bağlı olarak her bir Proje Yöneticisi ile entegre çalışan Bankacılık Yazılım Geliştirme, Bankacılık İş Danışmanlığı (Analistler) ve Kalite Güvence (Test Analist) ekipleri bulunmaktadır. Proje Yöneticisi tarafından atanan bir işi Bankacılık İş Danışmanlığı ekiplerinde yer alan İş Danışmanlığı Analistleri analiz ederler. Akabinde projenin gereksinimleri ve kapsamını detaylı bir şekilde anlatan İhtiyaç Analiz Dokümanlarını hazırlarlar. Bankacılık Yazılım Geliştirme

Ekibinde yer alan yazılım geliştiricileri bu dokümanı baz alarak ilgili işin geliştirmesini yaparlar. Geliştirilmesi yapılan iş teste verilmeye hazır olduğunda Kalite Güvence Ekibinde yer alan Test Analistlerine ilgili iş atanır. Burada bahsedilen iş Test Analistleri arasında yaygın olarak Yazılım Testi Projesi olarak ifade edilmektedir. Bu atama ile birlikte Test Analistleri, İhtiyaç Analiz Dokümanlarını baz alarak projelerini yürütürler.

Bankada süregiden bir problem proje yöneticileri tarafından Test Analistlerine atanan test projeleri için verilen sürelerin gerçekçi olmaması ve sürekli uzatılmak zorunda kalınmasıdır. Ancak test süresi içerisinde birçok farklı değişken barındırmakta olduğu için tahmin edilmesi oldukça zordur. Literatüre bakıldığı zaman bu sürenin tahmini için var olan çalışmaların yetersiz kaldığı görülmektedir. Bu sürenin tahmininin yaygın olarak uzman görüşüne bağlı olduğu gözlemlenmiştir. Bu çalışmada Test Analistinin kişisel bazda test çalıştırma süresini daha testin analiz ve tasarım aşamasında iken tahmin edebilmesi suretiyle, test projesinin ilk aşamasında olası bir süre uzatımının gerekliliğine karar verilebilmesinin ve buna yönelik önlem alınabilmesinin sağlanması hedeflenmektedir.

Birinci bölümde test süresi tahmin edilmesine yönelik geliştirilen yöntemler incelenmiştir. İkinci bölümde hedef kuruluşta var olan test faaliyetleri anlatılmıştır. Üçüncü bölümde Kişisel Yazılım Süreci ve Proxy Tabanlı Tahmin Yöntemi ile ilgili bilgi verilmiştir. Dördüncü bölümde Test Analistinin kendi kişisel test sürecini analiz etmek suretiyle, kendi süreç metriklerinden kendisine atanan test işlerine ilişkin test süresini hassas şekilde tahmin edebilmesi için geliştirilen Proxy tabanlı bir tahmin yöntemi tanımlanmıştır. Beşinci bölümde hata geri yayma yöntemi ile tahmin veri tabanının iyileştirilmesi ile ilgili bilgi verilmiştir. Altıncı bölümde geliştirilen yöntemin çalışabilirliği ve hassasiyetinin ölçülmesi için yapılan pilot çalışma anlatılmıştır ve son bölümde sonuçlar verilmiştir.

BÖLÜM I

İLGİLİ ÇALIŞMALAR

Bu bölümde, yazılım testi eforu tahmin etmeye yönelik literatürdeki araştırmalar/çalışmalar incelenmektedir. Literatürde yazılım testi eforu tahmini alanında yapılmış çalışmaların genellikle yazılım geliştirme eforu tahmini yöntemleri uyarlanarak türetilmiş olduğu görülmüştür.

K.R. Jayakumar, ve A. Abran yaptığı çalışmada Yazılım Testi Eforu tahmin etmeye yönelik metotları Uzman görüşüne dayalı yöntemler (Judgment and Rule of Thumb), Analoji ve iş kırılım yöntemleri (Analogy and Work Breakdown Techniques), Faktör ve Ağırlık tabanlı yöntemler (Factor and Weight-Base Estimation), Yazılım Büyüklüğüne dayalı yöntemler (Software Size-Based Estimation), Bulanık Mantık ve Diğer Modellere dayalı yöntemler (Fuzzy Inference and Other Models) olmak üzere beş ana grupta toplamaktadır [17].

1.1. Uzman Görüşüne Dayalı Yöntemler

1.1.1. Delfi Yöntemi (Delphi)

Delfi [18]: uzmanların klasik tahmin teknikleri içinde bireysel tahminlerin belirlenmesinde önemli rol üstlenirler. Deneme yanılma durumu, bir deneyim olarak görülmektedir. Birden fazla iterasyon gerçekleşir.

Uzmanlar işin mantığını diğer uzmanlardan öğrenirler ve sonraki iterasyonlarda tahminlerini yeniden yaparlar. Son tahmini daralan değer aralığından alınır, son yineleme uzmanları tarafından tahmin edilmektedir.

Delfi metodunun başarıyla izlediği grafiği üzerine, öncelikle bilgisayarlı delfi (computerized delphi method) yazılımları geliştirilmiştir. Yazılımların asıl hedefi, delfi metodunun çalışma zamanını hızlandırmak, turları ve tahminleri sorgulanabilir kayıtlar şekline dönüştürmek ve yönetici rolünün bazı kısımlarını yarı otomatize veya tam otomatize biçime dönüştürmektir. Bu konuyu bir örnek ile açıklamak gerekirse ki, yapılan tahminlerin toplanıp yalnızca ortalamalarının alınmış olduğu bir alanda yöneticinin rolünü bir bilgisayar yazılımının üstlenmesi gayet basit ve kolay bir olaydır. Bu bağlamda da yalnızca eksperlerin bulunduğu bir delfi metodu kolaylıkla uygulanabilir.

Bilgisayarlı delfinin daha sonraları gelişen internet ve web teknolojilerine uyarlanmış haline de web tabanlı delfi ismi verilmektedir (web based delphi) [24]. Bu yöntemlerin en temel farkı, eksperlerin aynı ortamda bulunması gereğini ortadan kaldırmış olmasıdır. Hatta eksperler farklı zamanlarda sisteme girerek görüş bildirebilir ve aynı anda online olmaları gerekmeyebilir.

Web tabanlı delfi metodu uygulama olanağı, metodun gelişme kaydetmesinde ciddi bir öneme sahiptir. Mesela çok fazla bireyin bu metot dahilinde olması mümkün hale gelmiş ve metodun elektronik demokrasi uygulamaları için kullanılması için kapı aralamıştır [23]. Bunun yanında da birden çok sürecin birbirine paralel olarak delfi metodunda ilerlemesi de mümkün kılınmıştır. Örneğin uzmanların uzmanlık alanlarına göre birimlerine ayrışması ve her uzmana kendi alanında sorular sorulması ve hatta farklı uzmanlıkların konuya etkilerine göre değişik oranlarda ağırlıklarla sonuca etki etmesi mümkün hale gelmiştir. Bulanık Delfi metoduna göre (Fuzzy Delphi Method), uzmanların farklı uzmanlık alanlarında edindikleri tecrübelere dayalı olarak farklı miktarda sonuca etki etmesi istenmektedir. Web tabanlı delfi uygulamaları da bu ve benzeri karmaşık uygulamalarda hız kazandırmakta ve maliyeti düşürmektedir.

1.1.2. Geniş-Band Delfi Yöntemi (Wide-Band Delphi)

Geniş Bant Delphi (Wide Band Delphi) tekniği ile projelerde büyüklük tahmini yapılması 1970'lerden itibaren gündeme gelmiştir. Esas itibariyle Delphi tekniğinin bir türevi olan bu yöntemi bir koordinatör (tercihen proje yöneticisi) yönetir ve konu uzmanları katılırlar. Alttaki adımları içerir:

- Koordinatör bir toplantı daveti ile ilgili uzmanları toplar ve onlara istenenler hakkında bilgilendirme yaparak tahminleme formu dağıtır.
- Uzmanlar formu bireysel olarak doldurur ve tahminlerini yazıp koordinatöre verirler ve ekip dağılır.
- Koordinatör ortaya çıkan tahminleri uzmanlarla paylaşır ve ekibi yeniden toplantıya çağırır.
- Toplantıda tahminlerdeki farklılıklar konuşulur.
- Uzmanlara yeniden formlar dağıtılıp, yeni tahminleri alınır. Eğer tahminler istendiği kadar yakın değilse, 3-5 adımlar tekrarlanır.

Tahminlemeyi yapacak ekibin çok kalabalık olmaması (ideali 3-7 kişi arası) önemlidir. Hem toplantıların verimi, hem de uzlaşının sağlanması sayıyla doğrudan ilgilidir. Farklı ekiplerin çalışması muhtemel işlerde tüm ekiplerden katılımcı alınması atlanmaması gereken bir detaydır. Uzmanların oluşturduğu tahmini ve varsayımlarını proje yöneticisi gözden geçirerek son haliyle proje tahminlerine yansıtır. Bu yöntem genel yaklaşımı sebebiyle, farklı projelerde kullanılabilir [21].

1.1.3. Başparmak Kuralı (Rule of Thump)

Bireyler ya da uzmanlar tarafından önceden tasarlanmış olan ve oranlara dayanan tahminlerdir. Tahminciler, ancak belgelenmemiş ve bağımsız doğrulanabilir temelleri tahmin etmektedirler. Tahmin teknikleri grubu, işlevsel boyut bu durumda dikkate alınmaz. İyi belgelendirilmiş tarihsel verilerin analizi üzerine, kıyaslama yapılması mümkün değildir.

1.2. Analoji ve İş Kırılım Yöntemleri

1.2.1. Analoji Tabanlı Kestirim (Analogy-Based)

Analoji tabanlı kestirim yöntemi de Delphi yöntemi gibi yazılım büyüklüğünü ve eforunu tahmin etmek için kullanılan bir yöntem olmak ile beraber test projelerine de uyarlanmakta ve test eforu tahmininde de kullanılmaktadır [19].

Bu yöntem organizasyonlardaki önceki ve birbirine benzer projelere ait bilgileri kullanarak gelecek projelerdeki performansı tahmin etme fikrine dayanmaktadır [18]. En önemli adımını birbirine benzer projelerin seçilmesi ve gruplandırması oluşturmaktadır.

Bu tekniğin iyi bir şekilde kullanılması için aşağıdaki önkoşullar gereklidir [18]:

- Organizasyon geçmişte birden fazla projeyi gerçekleştirmiş olmalı,
- Organizasyon geçmiş projelerini titizlik ile kayıt altına almış olmalı,
- Organizasyon her proje için projenin işlem sonrası incelenmesine yol göstermeli ve varyanslarının sebepleri titiz yöntemler kullanılarak tanımlanmış olmalı. Projeden elde edilen gerçek veriler tanımlanmış sebeplere bağlı olarak doğrulanır. Doğrulanmış gerçek değerlerden gelen veriler sonradan organizasyonun tahmin edilen havuzuna (ya da bilgi havuzuna) eklenir ve gelecek referanslar için uygun hale getirilir.

- Organizasyon iyi düzenlenmiş, korunmuş ve benzer geçmiş projelerin yerini saptamak ve onaylanmış proje verilerini bulmak için uygulanabilir bilgi havuzuna sahip olmalıdır.
- Tahmin işlemini yapacak kişiler, analojileri doğru bir şekilde çizme konusunda, bilgi havuzuna erişmek için, onaylanmış verileri çıkarmak için ve mevcut projeyi değerlendirmek için eğitilmiş olmalıdır.

Seçilen projeler için belirlenmiş detaylı tahmin verileri mevcut olduktan sonra yeni tahmin yapmak için aşağıdaki adımlar takip edilmektedir [18].

1. Eski tahmin yeni tahmin olarak kayıt edilir.
2. Eski tahminler detaylı olarak incelenir:
    a. Eğer mevcut proje ile aynı detaylara sahip ise, saklanır.
    b. Eğer mevcut projeden tamamen farklı ise, silinir.
    c. Eğer mevcut proje kısmen aynı ayrıntılara sahip ise, bilgiler mevcut projenin gerçekliğini yansıtacak şekilde ayarlanır
    d. Yeni bir tahmin yapmak için yukarıdaki adımlar eski projenin tüm detayları için uygulanır.
3. Ayrıntılar incelenir, bunların herhangi birinin mevcut proje ile alakalı olduğu ve eski tahminden tamamen farklı olduğu belirlenir. Tahminleri ayrıntılar açısından kapsamlı kılmak için bu ayrıntılar tahmin içine eklenir.
4. Tahminde uygun ayarlamalar yapabilmek için aşağıdaki unsurlar incelenir.
    a. Geliştirme platformu
    b. Programlama dili
    c. Uygulama katman sayısı
    d. Geliştirme yeri
    e. Ara Katman
5. Şimdi tahmin akran değerlendirmesi (peer review) için hazır olmalıdır ve eğer varsa geri bildirim yapılır.
6. Yönetimsel değerlendirme, uygulama geri bildirimi ve eğer var ise onay almak için düzenleme yapılır.
7. Yapılan tahmin, organizasyonel tahmin sürecindeki standart biçimler kullanılarak talep eden kişiye gönderilir.

Sadece net olmayan tahminler var ise:

1. Yeni projenin her parametresi eski projeninki ile karşılaştırılarak incelenir.
2. Her bir parametreye ağırlık atanır.
    a. Ağırlık 0 ise, parametrenin mevcut projeye uygulanabilir olmadığını gösterir,
    b. Ağırlık 1 ise, parametrenin yeni proje ile aynı olduğunu gösterir,
    c. Ağırlık 0'dan fazla fakat 1'den küçük ise, bir parametrenin uygulanabilir olduğunu gösterir, ancak 100 % den az,
    d. Ağırlık 1 den fazla ise, bir parametrenin uygulanabilir olduğunu gösterir, ancak 100 % den fazla,
3. Tüm ağırlıklar toplanır, birleşik ağırlık faktörü (composite weight factor – CWF) elde etmek için aritmetik ortalaması bulunur.
4. Yeni tahmin elde etmek için geçmiş projenin brüt tahminleri CWF ile çarpılır.
5. Tahmin hazır olduğundan, akran değerlendirmesi (peer review) ve eğer varsa uygulama geribildirimi için düzenleme yapılır.
6. Yönetsel değerlendirme ve onay almak için düzenleme yapılır.
7. Tahmin, organizasyonel tahmin sürecinden gelen standart formatlar kullanılarak talep edilen kişiye gönderilir.

1.2.2. Görev Tabanlı Kestirim (Task-Based)

Görev tabanlı kestirim yöntemi yazılımın büyüklüğünü tahmin etmeden direk olarak yazılım geliştirme eforunu tahmin etmektedir. Bu yöntem Aktivite Tabanlı Kestirim olarak da bilinmektedir.

Görev Tabanlı Kestirim yönteminin metodolojisi aşağıdaki gibidir [18]:

1. Yazılım ürünü kurucu modüllerine ayrılarak başlanır. Bir modül, bir ürünün önemli bir parçasını oluşturur ve önemli bir fonksiyonel gruplama anlamına gelir. Birden çok ekran, rapor ve ilgili betiklerden oluşabilir. Örneğin malzeme yönetimi yazılımı için ana modüller aşağıdaki gibi olabilir.
    a. Depo modülü
    b. Satın alma modülü
    c. Envanter kontrol modülü
    d. Borçlar modülü
2. Her bir modül için:

a. Uygulanabilir tüm aşamalar listelenir.
b. Her aşamada uygulanabilir tüm görevler listelenir.
c. Her bir görev için, üç tahmin kullanılarak gerekli efor tahmin edilir; iyimser (en iyi durum – best case), kötümser (en kötü durum – worst case) and muhtemel (normal durum – normal case) – formül kullanılarak beklenen efor (Expected effort) hesaplanır.
   Beklenen Efor = ((En iyi durumdaki değeri + En iyi kötü durumdaki değeri) + (4*normal durumdaki değer))/6
d. Efor tahmini için tutarlı bir ölçü kullanılır, kişi-saat veya kişi-gün.
e. Önerilen, her görevin eforunun kişi-haftadan az olarak tasarlanmasıdır.
f. Daha doğru/kesin bir tahmin yapmak için faz başına daha fazla görev kullanılması ve ayrıntıların artırılması önerilmektedir.

3. Proje aşamalarının tamamı (proje başlangıcı, proje kapanışı, projenin planlaması) için gerekli efor bileşenlerin görevlerine ayrılması ve her bir görev için eforun tahmin edilmesi ile bulunmaktadır. Ardından, her faz için gerekli efor toplanır. Sonra modüle veya projenin eforunu bulmak için fazların eforu toplanır.
4. Tüm varsayımlar dokümante edilir. Tüm tahminler, teknoloji, gereksinimlerin tamamlanması, onaylar, açıklamalar, çevre vb. ile ilgili bazı varsayımlar yapmayı gerektirir. Bunlar gerçek eforu etkileyecektir. Tahmin sırasında yapılan tüm varsayımların kayıt edilmesi avantaj olabilmektedir. Çünkü projeyi bütçelendiren kişilerin iyi bir şekilde tahmini anlaması önemlidir.

1.2.3. Test Senaryosu Sayma Tabanlı Kestirim (Test Case Enumeration-Based)

Test Senaryosu Sayma Tabanlı Kestirim yöntemi denetlenebilir bir yöntemdir. Test senaryolarının sayılmasına dayanmaktadır. Test senaryolarının tek tek sayılması ve eforlanması detaylı bir bilgi sağlamaktadır. Bundan dolayı bu yöntem ile kapsamlı ve olabildiğince hassas tahminler yapılmaktadır. Diğer taraftan test senaryolarının tek tek sayılması uzun bir işlemdir ve tahmin etme süresini artırmaktadır.

Bu teknik aşağıdaki adımları içermektedir [18]:

1. Tüm test senaryoları sıralanır.

2. Tutarlı bir şekilde, adam-saat veya adam-gün olarak her bir test senaryosunun eforu tahmin edilir.
3. Her bir test senaryosuna gerekli efor'u tahmin etmek için iyi-durum (best-case), normal-durum (normal-case), ve kötü durum (worst-case) senaryoları kullanılır.
4. Beta dağılımı kullanılarak her bir test senaryosu için beklenen çaba aşağıdaki gibi hesaplanır.

   (En iyi durum + En kötü durum + (4*normal durum))/6

5. Aşağıdakiler toplanır:
   a. Proje için beklenen efor tahminini elde etmek için beklenen süreler.
   b. İyi durum efor tahminini elde etmek için iyi –durum süreleri
   c. Kötü durum efor tahminini elde etmek için kötü –durum süreleri
   d. Normal durum efor tahminini elde etmek için normal –durum süreleri

1.3. Faktör ve Ağırlık Tabanlı Yöntemler

1.3.1. Test Noktası Analiz (Test Point Analysis)

Test Noktası Analiz (Test Point Analysis – TPA) yöntemi sadece kara-kutu test tekniği ile yapılan testlerin süresini tahmin etmeye yöneliktir. Bu yöntem üç temel unsurdan oluşmaktadır: test edilecek sistemin büyüklüğü, test stratejisi ve verimlilik seviyesi [31]. İlk iki unsur birlikte testin hacmini vermektedir. Test hacmi test noktası ile ifade edilmektedir. Test noktalarının verimlilik ile çarpılması ile de test süresi saat olarak bulunmaktadır. Bahsedilen üç temel unsur ile ilgili detaylar aşağıdaki gibidir [31]:

- **Büyüklük (Size):** Dikkat edilmesi gereken ilk unsur sistemin boyutu olmaktadır. TPA amaçları için, bir bilgi sisteminin boyutu esas olarak kendisine atanan işlev noktalarının sayısına göre belirlenir. Bununla birlikte birkaç ekleme veya düzeltme yapılması gerekmektedir. Çünkü bazı faktörlerin işlev noktası için hiç etkisi olmayabilir veya az etkisi olabilir. Bu faktörler: Karmaşıklık (Complexity), Arayüz (Interfacing) ve tekdüzelik (Uniformity).
- **Test Stratejisi (Test strategy):** Geliştirme veya bakım sırasında, bilgi sistemi için kalite gerekleri belirtilmiş olacaktır. Test aktiviteleri bu gereksinimlerin ne ölçüde yerine getirildiğini belirlemektedir. Müşteri ile

irtibat halinde olarak test edilecek olan sistemin ve/veya alt sistemin kalite karakteristiği tanımlanır ve onların önem derecesi belirlenir. Her karakteristiğin önemi, ilgili test etkinliklerinin titizliğini etkiler. Kalite nitelikleri ne kadar önemli olursa, testler de o denli titiz ve kapsamlı olmaktadır. Böylece çalışma hacmi de büyük olmaktadır. Farklı karakteristiklerin önemi, test stratejisi oluşturulurken müşteri ile istişare edilerek belirlenmelidir, bilgiler daha sonra TPA girişi olarak kullanılabilir. TPA süreci boyunca, test çalışmasının hacmi test stratejisine dayanarak hesaplanır.

- **Verimlilik (Productivity):** TPA'da verimlilik, bilgi sistemi ve test stratejisi boyutuna göre belirlenmektedir ve bir test noktasının gerçekleştirilmesi ile ilgilidir. Verimliliğin iki bileşeni vardır:
    - Verimlilik Figürü (The productivity figure): Verimlilik figürü, test ekibinin bilgi ve becerisine dayanmaktadır. Bu nedenle bireysel organizasyona özgüdür.
    - Çevresel Faktör (The enviromental factor): Çevresel faktör, çevrenin test faaliyetlerin üretkenlik ile ne derece ilişkili olduğunu ve etkilediğini belirtir. Etkili çevresel faktörler, test araçlarının mevcut olması, ekibin test ortamıyla birlikte yaşadığı deneyimin miktarı, testin temel kalitesi ve test yazılımının bulunabilirliğini içerir.

1.3.2. Kullanım Noktaları (Use Case Test Points)

Kullanım Noktaları yöntemi Test Noktası yöntemine alternatif olarak Kullanım Noktalarına dayanarak Yazılım Geliştirme eforunu tahmin etme yönteminden türetilmiştir [17]. Yöntemin temelinde kullanım noktalarının test senaryolarına ve transactionlara bağlı olarak ağırlıklandırılmasına dayanmaktadır.

Kullanım Noktaları yöntemi ile test süresi tahmini sırası ile aşağıdaki adımları içermektedir [29]:

1. Sistemdeki aktör sayısı belirlenir. Bu, düzeltilmemiş aktör ağırlıklarını (UAW –the unadjusted actor weights) vermektedir. Aktörler, sistemin dışındadır ve onun ile ara yüz oluştururlar. Örnek: son kullanıcılar, diğer programlar, veri mağazaları vb. Aktörler üç tür olabilirler: basit (simple), ortalama (average) ve karmaşık (complex)

2. Sistemdeki kullanım durumlarının sayısı belirlenir ve UUCW (Unadjusted Use Case Weights – Düzeltilmemiş Kullanım Nokta Ağırlıkları) elde edilir. Kullanım durumlarına, işlem/senaryo sayısına bağlı olarak ağırlık verilir.
3. Düzeltilmemiş UCP (unadjusted UCP)'nin hesaplanması, düzeltilmemiş aktör ağırlığı ve önceki adımda belirlenen düzeltilmemiş kullanım durum ağırlığı (the unadjusted use case weights) toplanarak yapılır.
   UUCP = UAW + UUCW
4. Teknik ve çevresel faktörler hesaplanır.
5. TEF (Technical Complexity Factor – Teknik Komplekslik Faktörü) kullanılarak Düzeltilmiş UCP (adjusted UCP) hesaplanır. AUCP=UUCP*[0.65+(0.01*TEF)]
6. Düzeltilmiş UCP'nin bir dönüşüm faktörü ile çarpılması gerekmektedir. Bu dönüşüm faktörü dil/teknoloji kombinasyonu için gerekli adam-saat sayısını göstermektedir. Organizasyonun, çeşitli kombinasyonlar için bu dönüşüm faktörünü belirlemesi gerekmektedir.

1.3.3. Test Çalıştırma Noktası (Test Execution Points)

Test Çalıştırma Noktası yöntemi test çalıştırma eforunu sistemin büyüklüğüne dayalı olarak tahmin etmektedir [17]. İlk olarak test çalıştırma noktalarının ölçümlemesi ve akabinde ölçümlenmiş noktaları kullanarak Test Üretim Tabanlı Modeli veya Regrasyon Modelini kullanarak test çalıştırma eforunu tahmin etmeyi hedeflemektedir.

Test Çalıştırma Noktalarının (EP) ölçülmesi testin büyüklüğü ve test çalıştırma kompleksliği hakkında referans olmaktadır, EP'lerin ölçülmesi için sırası ile aşağıdaki adımlar takip edilmektedir [9]:

1. Test özellikleri ve her bir test aşamasının her adımının karakteristik özelliklerinin (C1 to Cn) bir listesine göre bireysel olarak analiz edilir. Yöntem tarafından ele alınan her karakteristik, testin boyutu ve yürütme karmaşıklığı üzerinde ve dolayısı ile testi gerçekleştirmek için gereken efor üzerinde etkili olur.
2. Bu etki, bir sıra ölçek kullanarak derecelendirilmiştir. (Düşük, Ortalama ve Yüksek).

3. Her karakteristik için EP'ler etki seviyesine göre atanır ve nitelik oranı (etki düzeyi) nicel bir değere dönüştürülür.
4. Her karakteristik için atanan puanlar toplanır.
5. Her bir test adımının EP'lerinin toplanması sureti ile testin boyutu ve yürütme karmaşıklığı ölçülmektedir.

Test Üretim Tabanlı Modeline göre [9], Tahmini efor, EP sayımı ve bir dönüşüm faktörü (a conversion factor - CF) temel alınarak hesaplanabilmektedir. CF, test yürütme çabası ile EP'ler arasındaki ilişkiyi temsil eder ve test ekibinin üretkenliğine göre değişir. CF, saniyedeki EP olarak verilir ve bir testin her bir EP'sinin çalıştırılması için gereken saniye sayısını belirtir. CF'yi hesaplamak için, testçiler testin boyutunu ve yürütme karmaşıklığını ölçebilirler. Ardından, testlerin yürütme zamanı tarihsel bir veri tabanında (var ise) toplanmalı veya yürütülmelidir.

Regresyon Modeline göre [9]; temel olarak, ana hedef tahmincilerin en iyi kümesini (maliyet belirleyicileri - cost drivers) bulmak ve yanıttaki (efor) değişimi oluşturan bir denklem oluşturmaktır:

$$\text{Efor} = \alpha + \boldsymbol{\beta_0}\text{EP} + \boldsymbol{\beta_1 D_1} + \ldots + \boldsymbol{\beta_n D_n} \quad (1.1)$$

- **EP**: testteki EP'lerin sayısıdır,
- **Efor:** testleri gerçekleştirme çabası/eforu
- $\boldsymbol{D_1}$**'den** $\boldsymbol{D_n}$**'e:** regresyon analizini göre iyi öngörücüler olarak seçilmiş olan test yürütme maliyet belirleyicileri (Örn: Takım deneyimi, yazılım istikrarı vb.)
- $\boldsymbol{\alpha}$ **ve** $\boldsymbol{\beta_0}$**'den** $\boldsymbol{\beta_n}'\boldsymbol{e}$: regresyon analizi ile tanımlanan model katsayıları

1.4. Yazılım Büyüklüğüne Dayalı Yöntemler

Büyüklük tabanlı kestirim (The Size-based) [17]: boyutun kullanıldığı bir tahmin yaklaşımıdır ve standart tanımlar benimsenerek toplanan geçmiş veriler temel alınarak oluşturulmuş regresyon modelidir. Burada, boyut, önemli girdi parametrelerinden biridir. Bu tekniklerden bazıları boyutu efora dönüştürmek için dönüşüm faktörleri kullanmaktadır. Boyut kullanılarak oluşturulmuş regresyon modelleri, geçmiş verilere dayanan efor tahminini doğrudan mümkün kılar.

1.4.1. Test Büyüklüğü Tabanlı Kestirim (Test Size Based)

Bir test projesinin başlangıç aşamasında yazılım boyutu zaten bilinmektedir. Bu yazılım boyutu test projesi boyutu olarak kabul edilir ve test projesini yürütmek için gereken efora ulaşabilmek için yazılım boyutu için bir ürün figürü tayin edilmektedir [18].

Bu yöntem aşağıdaki adımlardan oluşmaktadır [19]:

1. Mevcutta olan geliştirilmiş yazılımın büyüklüğü kullanılır.
2. Uygulamanın türüne bağlı olan çevirme faktörleri kullanılarak yazılımın büyüklüğü Uyumlandırılmamış Test Noktalarına dönüştürülür. (Unadjusted Test Points – UTP)
3. Kompozit Ağırlık Vektörü (Composite Weight Factor – CWF) hesaplanır
    a. Seçilen testlerin bireysel ağırlıkları toplanır
    b. Uygulamanın ağırlığı ile çarpılır
    c. Seçilen Birim Testinin dilinin ağırlığı ile çarpılır
    d. Eğer bir araç kullanılmış ise aracın ağırlığı ile çarpılır
4. Uyumlandırılmamış test noktaları CWF ile çarpılarak testin boyutu test noktaları olarak elde edilir.
5. Verimlilik Faktörü (Productivity Figure) bir test mühendisinin bir test noktasını tamamlama süresi olarak ifade edilmektedir.
6. Test Eforu test noktası boyutu ile verimlilik faktörünün çarpılması ile Adam/saat cinsinden hesaplanır.

1.4.2. COCOMO ve SLIM

Cocomo ve Slim gibi yöntemler Yazılımın Büyüklüğüne Dayalı olarak Yazılım geliştirme eforunu tahmin etmeyi sağlamaktadır. Test eforunu tahmin etmek için de bu yöntemlerin test sürecine uyarlamaları geliştirilmiştir [17].

COCOMO (The COnstructive COst Model) [5]: COCOMO aslında giderek daha detaylı üç modelden oluşan bir hiyerarşik yapıdır. Sahip olan bir modeldir. Mevcut maliyet tahmin modelinin makul bir örneğini sağlamak için COCOMO'nun ara seviyesi aşağıdaki gibi açıklanmaktadır.

Ara (Intermediate) COCOMO önerilen bir yazılım ürününün maliyetini aşağıdaki şekilde tahmin etmektedir:

1. Sembolik geliştirme eforu (nominal development effort) teslim edilen ürüne ait bin mertebesinde kaynak yönergeleri (KDSI) ve projenin geliştirme modunun bir fonksiyonu olarak tahmin edilir.
2. Efor çarpanı kümesine, 15 maliyet sürücüsünün (cost drivers) özelliklerine dayalı olarak ürünün derecelendirilmesi ile karar verilir.
3. Tahmini geliştirme eforu, sembolik efor tahmininin tüm ürün çaba çarpanlarının çarpılması ile elde edilir.
4. Ücretlerin, geliştirme çizelgelerinin, faz ve maliyet dağılımlarının, bilgisayar maliyetlerinin ve yıllık bakım maliyetlerinin belirlenmesinde geliştirme efor tahminin den elde edilen ek faktörler kullanılabilir.

SLIM [5]: The Putnam SLIM Model piyasada kullanılabilen bir yazılım ürünüdür. Quantitative Software Management, firması tarafından desteklenmektedir. Bu ürün Putnam analizine dayanmaktadır. Rayleigh dağılımına göre personel seviyesi yönünden yazılım ömrünün zamana karşı analizidir. SLIM de kullanılan temel efor makro-tahmin modeli:

$$\mathbf{S_s = C_k K^{1/3} t_d^{4/3}} \quad (1.2)$$

- $\mathbf{S_s}$ = teslim edilen kaynak talimat sayısı (delivered source instructions)
- $\mathbf{K}$ = insan-yıl olarak yaşam döngüsü eforu
- $\mathbf{t_d}$ = yıllardaki geliştirme süresi
- $\mathbf{C_k}$ = teknoloji katsayısı

1.5. Bulanık Mantık ve Diğer Modellere Dayalı Yöntemler

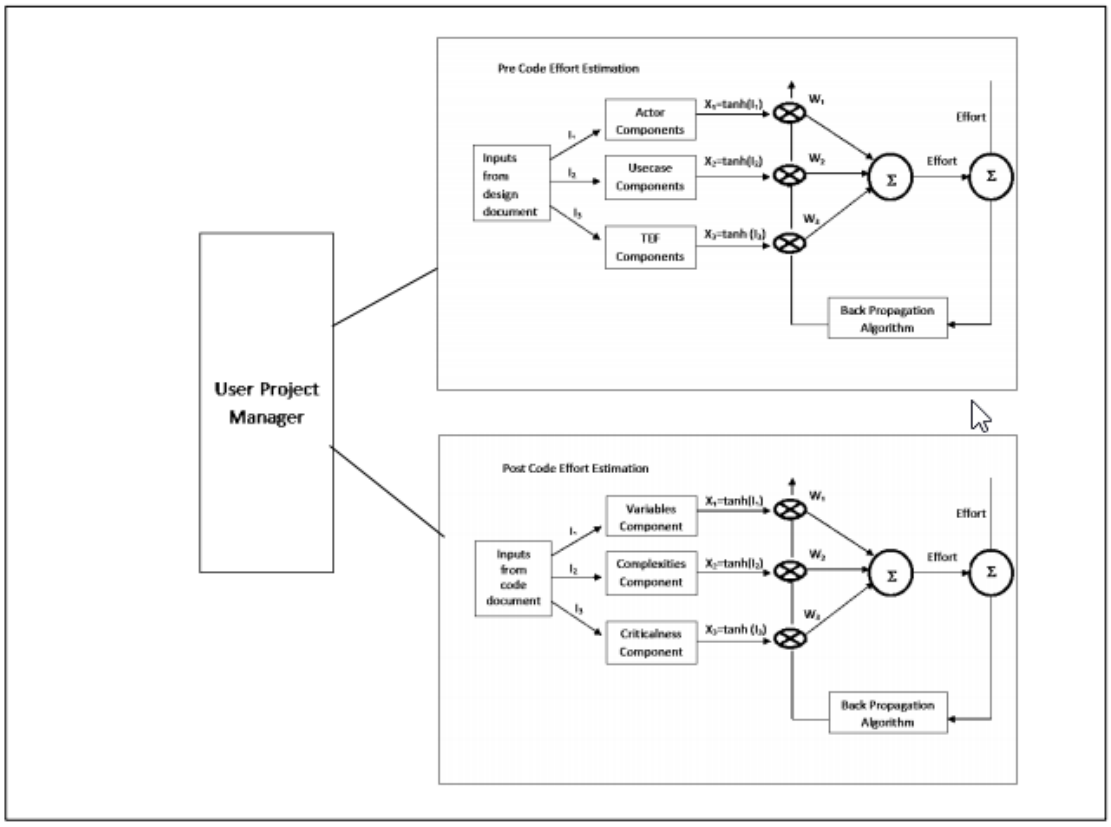
Literatürde Bulanık Mantık (Fuzzy Logic), Yapay Sinir Ağları (Artifical Neural Networks) ve Senaryo Tabanlı Akıl Yürütme (Case-Base Reasoning) yöntemleri kullanarak yazılım geliştirme eforunu tahmin etmeye yönelik deneysel çalışmaların olduğu görülmektedir. Bu çalışmaların çoğu Yazılım Geliştirme Eforunu Tahmin etmeye yönelik olmasına rağmen Yazılım Test Eforunu tahmin etmek içinde adapte edilebilmektedir [17].

C.Abhishek, V.Kumar ve diğerlerinin yaptıkları çalışmada Yapay Sinir Ağları yöntemini kullanarak Test Eforunu Tahmin etmektedirler. Bu çalışmada tasarlanan yapı Şekil-1 de gösterildiği gibi önce ve sonra efor tahmini olmak üzere iki

komponent içermektedir ve bu komponentlar ağa giriş değerleri olarak verilerek Geri Yayma Yöntemi (Backpropagation Algorithm) ile öğrenme sağlanmaktadır.

Ön kodlama (pre coding) efor tahmini üç girdi bileşeninden oluşmaktadır. Bu bileşenler tasarım dokümanında yer almaktadır. Üç bileşen aşağıdaki gibidir [6]:

- Aktör bileşenleri (Actor components)
- Kullanım-durumu bileşenleri (usecase component)
- Teknik ve çevresel faktörler (technical and environmental Factors component)



Şekil 1. Yapay sinir ağları kullanılarak test eforu tahmini için tasarlanan yapı

Sonraki kodlama (post coding) efor tahmini girdiyi kod dokümanından alır ve üç bileşenden oluşmaktadır. Bunlar [6]:

- Değişken bileşenleri (Variable component)
- Karmaşıklık bileşenleri (Complexity component)
- Kritiklik bileşenleri (Criticalness component)

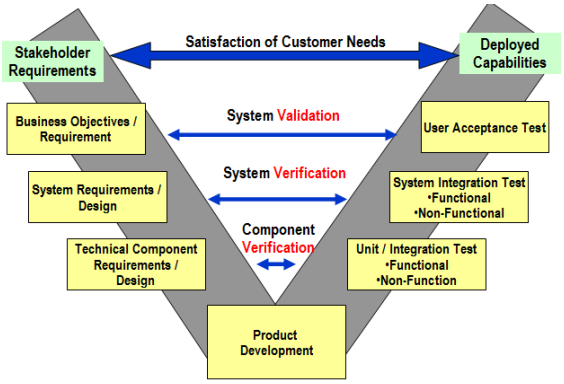
Geliştirilen model, kullanıcılardan (proje yöneticileri) gelen girdiler ile ara değerleri tahmin eder. Bu değerler sinir ağına verilir. Buradan üretilen bilgi ile nihai efor tahmini için veriler iyileştirilir [6].

BÖLÜM 2

BANKADA YÜRÜTÜLEN YAZILIM TEST FAALİYETLERİ

2.1 Yazilim Geliştirme Sürecinde Test

Test faaliyetlerinin, yazılım geliştirme sürecinin daha başlangıç safhalarından itibaren vazgeçilmez bir parçası olduğu açıktır. Bu noktada yazılımların da bu faaliyetlere destek verir nitelikte olmasının önemi ortaya çıkmaktadır. ISO/IEC 9126 standardı [13] yazılım kalite kriterlerini tanımlarken Bakım-İdame Edilebilirlik (Maintainability) ana başlığı altında "Test Edilebilirlik" (Testability) kriterini önemli bir kalite kriteri olarak sunmaktadır. Bu kriterin sağlanması için, yazılım gereksinimleri tanımlanırken ilişkili kalite kriterlerinin de belirlenmesi, geliştirme aşamasında da bu kriterlerin sağlanması için gerekli çalışmaların yürütülmesi gerekmektedir. Test faaliyetleri bu çalışmanın yapıldığı bankada standart olarak kullanılan V-modeline uygun olarak gerçekleşmektedir. (Bkz. Şekil-2).



Şekil-2 : Bankanın Kullandığı V Modeli

V model, geleneksel yazılım geliştirme modeli olan Şelale (Waterfall) Modelinin gelişmiş hali olarak olarak düşünülmektedir. Şekil-1'de gösterildiği gibi V model şelale modelinde (verification and validation) doğrulama ve geçerleme işlevlerinin vurgulanması ile elde edilmektedir. V modelin sağ tarafında yer alan aktiviteler sol tarafında yer alan aktivitelerin çıktılarını doğrulamakta veya geçerlemektedir. V'nin sol tarafı kullanıcı ihtiyaçlarını küçük yönetebilir parçalara bölen analiz faaliyetlerini temsil ederken, sağ tarafı bu faaliyetlere karşılık gelen kullanıcı ihtiyacını karşılayan sistemin test aktivitelerini göstermektedir [8].

V modelinin sağ tarafında üç geliştirme seviyesine karşılık Kullanıcı Kabul Testi (Acceptance Test), Sistem Entegrasyon Testi (System Integration Test), ve Birim/Bileşen Testi (Unit Integration Testing) olmak üzere üç test seviyesi bulunmaktadır. Test Analisti, test sürecini (Bkz. Bölüm 2.3) Sistem Entegrasyon Testi seviyesinde uygulamaktadır. Bu süreç aşağıdaki şekilde gerçekleşmektedir [14]:

- Sistem Entegrasyon Testinin planlaması, proje planlaması ile aynı anda gerçekleşir ve test kapanış işlemleri tamamlanana kadar devam eder.
- Sistem Entegrasyon Testinin analiz ve tasarımı, yazılımcının birim testini yapması ile paralel gerçekleştirilir.
- Sistem Testinin Yürütülmesi, birim testinin tamamlanması ile başlar ve test çıkış kriterleri karşılanana kadar devam eder.
- Sistem Testinin Kapanış Faaliyetleri, sistem test çıkış kriterleri karşılandıktan ve sistem test yürütmesinin tamamlandığı beyan edildikten sonra gerçekleştirilir. Ancak bazı durumlarda, kullanıcı kabul testi bitene ve tüm proje faaliyetleri son bulana kadar gecikebilir.

2.2. Metodoloji ve Yönetimi

Bu çalışmada ele alınan testler, kara kutu test yaklaşımı ile geliştirilen uygulama özelliklerinin kullanıcı tarafında talep edilen fonksiyonel gereksinimleri karşılayıp karşılamadığını görmek amacıyla Test Analisti tarafından yapılan fonksiyonel testlerdir.

Fonksiyonel testler kapsamında önemli olan uygulamanın doğruluğudur. Kullanıcının taleplerine bağlı olarak uygunluk, tamlık, birlikte çalışma ve hatalı girdi kontrollerini kapsamaktadır.

2.3 Test Süreci

Bu çalışmanın yürütüldüğü bankanın test süreci IBM'den [1] alınan danışmanlık kapsamında benimsenmiştir. Kullanılan test süreci aşağıdaki faaliyetleri kapsamaktadır.

- Test Planlama Kontrol
- Test Analizi ve Tasarımı

- Test Uyarlama ve Çalıştırma
- Raporlama ve Test Kapanış İşlemleri

Test analistinin bu faaliyetlerin yukarıdaki sıra ile uygulanması beklenmektedir. Ancak bazı özel durumlar ile karşılaşıldığında paralel uygulamasına izin verilebilmektedir.

2.3.1. Test Planlama Kontrol

Test Yöneticisi tarafından projeye ilişkin teknik analiz, iş analizi ve gereksinim dokümanları incelenir. Roller ve görevler herkes tarafından netliğe kavuşturulur. Risk analizi yapılır. Testte kritik ve öncelikli alanlar ilk olarak belirlenir. Test çıkış kriterleri ifade edilir. Bu aşamada Test Analistinin aktif rolü bulunmamaktadır.

2.3.2. Test Analizi ve Tasarımı

Test Analizi ve Tasarımı Aşamasında, genel test hedefleri somut test koşullarına ve test senaryolarına dönüştürülür [15]. Bu çalışmaya konu olan bankada hataları bulma, kalite seviyesi hakkında güven kazanma, karar verme için bilgi sağlama [15] gibi olan test hedeflerinin gerçeklenmesi için analistler tarafından hazırlanan İhtiyaç Analiz Dokümanları baz alınmaktadır (Functional Analysis Document - FAD). İhtiyaç Analiz Dokümanı, bir yazılım işinin nasıl yapıldığı, iş hedeflerini ve stratejilerini, işin süreçlerini ve bu süreçlerdeki normal, alternatif ve sıra dışı akışları, bu akışları yönlendiren iş kurallarını yani her türlü hesaplama, kısıtlama, tetikleme ve şart ifadelerini, süreçlerde bulunan insan rolleri ile bu rollerin sistem ile etkileşimini, süreçlerde iletişimde bulunulan yani entegre olunan diğer sistemleri, yani işin tüm insan olan ve olmayan tüm aktörlerini, sistemli bir şekilde çözümlenmesi bilgisini içeren dokümanlardır [2].

Yazılım geliştirme sürecinin girdilerini barındıran en önemli kaynak İhtiyaç Analiz Dokümanıdır. Yazılım geliştirme ekibi ürünü bu dokümana dayalı olarak tasarlamakta ve geliştirmektedir. Test Analistleri ise bu dokümana dayalı olarak ürünün tüm fonksiyon ve süreçlerini doğrulamak ve onaylamak ile sorumludur. Bundan dolayı, Test Analistleri, ihtiyaç analiz dokümanında yer alan girdiler, olaylar veya aksiyonlar ile bunların sonucunda oluşması beklenenlerin gerçekleştirilmesi için gerekli adımların açıklandığı dokümantasyonu [12] yani test senaryolarını, test analiz ve tasarım aşamasında hazırlamaları gerekmektedir. Test senaryoları, kara

kutu test yaklaşımında olduğu gibi, koda veya kodun tasarımına bakılmadan sadece ürünün beklenen özelliklerine göre tasarlanmaktadır.

Bir test senaryosu; Seviye, Test Adımı, Doğruluk Kriteri, Erişim Bilgisi, Senaryo Özelliği bilgisini içermelidir.

- Seviye: test senaryosunu önceliklendirmek için kullanılan değer,
- Test Adımı: test senaryosunun gerçekleştirilmesi için gerekli adımlar. Bunlar test nesnesinin analizine, özelliklerine, davranışına ve yazılımın yapısına dayanmalıdır [16].
- Doğruluk Kriteri: Bir test nesnesinin (fonksiyon) veya özelliğin başarılı veya başarısız olup olmadığını belirlemek için kullanılan karar verme kuralları [16].
- Senaryo Özelliği: Pozitif veya negatif olarak iki değerden birini alabilir. Pozitif olması durumunda test senaryosu sonunda başarılı bir işlem gerçekleşmiş, negatif olması durumunda ise başarısız bir işlem gerçekleşmiş olması beklenir.

Örnek bir test senaryosu aşağıdaki gibidir.

- **Seviye:** 7.4
- **Test Adımı:** Kullanıcı DYBDXXX yetkisi verilmez + birimi ilgili işlem referansını almaya yetkili olur + Yeni referans alma yetkisi verilir + İşlem Tipi Tanımlama Ekranında tanımlı olan bir referansı alır.
- **Doğruluk Kriteri:** Referansı alamadığı görülür.
- **Erişim Bilgisi:** B. D. Ekranı / D. R. İşlemleri
- **Case Özelliği:** Pozitif

Bu aşamada test senaryolarının hazırlanması ile birlikte test ortamının kurulumu tasarlanır, gerekli altyapı ve araçlar tanımlanır, test senaryolarını desteklemek için test verisi belirlenir [14].

2.3.3. Test Uyarlama ve Çalıştırma

Test Analisti, test senaryolarını ve test ortamının hazırlamasına paralel olarak yazılımcının ürününün birim testini yaptığı ve teste hazır hale getirdiği varsayılır. Test Uyarlama ve Çalıştırma aşamasında yazılımcının ürününü teste vermiş olduğu kabul edilir.

Yazılımcının ürünü (the product) bankacılık alanında yaygın olarak teste geldiği zaman test nesnesi (test object) olarak ifade edilmektedir. Test nesnesi ilk gereksinim tanımlarından müşteriye gönderilecek son ürüne kadar herşey olabilir. Testin ilk aşamalarında test nesnesi genellikle bir doküman (planlar, İhtiyaç Analiz Dokümanları, kod) olurken en sonunda, test çalıştırılırken kullanılabilen, çalışabilir bir yazılım olmaktadır. [3]. Bu çalışmada bahsedilen test nesnesi, test çalıştırma aşamasında test analistine gelen kullanılabilir çalışan bir yazılım olduğu kabul edilmektedir.

Test Uyarlaması testlerin çalıştırılmaya hazırlanmasını, test verilerinin ve test senaryosunun yürütülmeye başlayabilmesi için test ortamının nihai hale getirilmesini ve kaynak tahsisini içeren test çalıştırma çizelgesinin hazırlanmasını kapsar [14].

Test Uyarlama aşamasında test ortamı nihai haline getirildikten sonra test çalıştırılmaya başlanır. Testin Çalıştırılması Aşamasında, her bir test senaryosunun içerdiği test adımlarına karşılık gelen doğruluk kriterinin sağlandığı teyit edilir. Çalıştırılan test senaryoları ekran görüntüsü alınarak ile kayıt altına alınır.

Her bir test senaryosu çalıştırılmasında görsel olarak elde edilen sonuçlar (örn: hata mesajları, proxy yanıtları) ve her bir çıkış değerinin (örn: reel bir sayı) bulunduğu yer başarılı ve ya başarısız sonuçlar için kayıt altına alınır [9].

Başarısız sonuçlar, gereksinimden, tasarımdan, kullanıcı dökümanından, standartlardan, beklenti, tecrübe veya algıdan sapma durumudur. Gözden geçirme, test, analiz, derleme veya ürünün kullanımı sırasında ortaya çıkabilir (IEEE 1044). Bunlar hata, bulgu, sapma, insan hatası/hata, kusur, arıza, olay, problem olarak ifade edilebilir [16]. Bu çalışmada testin uyarlama ve çalıştırma aşamasında tespit edilen bu başarısız sonuçlar bulgu olarak ifade edilmektedir.

2.3.4. Raporlama ve Test Kapanış İşlemleri

Raporlama ve Test Kapanış İşlemleri Aşamasında Ürünün ihtiyaçlara uygun hale gelmesi, tespit edilen bulguların giderilmesi, gerçek ortama alındığında risk içermemesi halinde test planındaki test çıkış kriterlerine de uygunluğu kontrol edilerek "Test Kapanış Raporu" hazırlanır ve test sonlandırılır [15]. Test Kapanış Raporu gerçek ortama alınacak uygulamadaki hataların ve yazıların test senaryolarının durumlarını gösteren rapordur. Test ortam bilgilerini içerir.

Çözülmemiş bulguların riski hakkında bilgi verilir. Bulguların kaç tanesi kapatıldı bilgisi ve grafiğini içerir. Senaryoların kaç tanesinin çalıştırıldığının bilgisi ve grafiğini içerir. Bununla birlikte test dokümanı test planı, test tasarım özelliklerini, test loğlarını içerebilmektedir [30].

BÖLÜM 3

PSP VE PROXY YÖNTEMİ

Yazılım geliştirme projelerinde çalışan kişilerin yetenek ve çalışma disiplinleri kaliteli ürün ortaya koyabilmek için büyük önem arz etmektedir. Bu bağlamda Watts S. Humprey, PSP (Personel Software Process - Bireysel Yazılım Geliştirme) konularındaki çalışmaları ile kişilerin çalışma sonuçlarını iyileştirmek için tanımlanmış süreci izleyerek disiplin içinde yazılım geliştirmeyi sağlayan bir çerçeve sunar [18]. Kişisel disiplini ve performansı artırmak için PSP kişilere: sürdürülebilir, kesin ve açık plan yapma; çalışma performansını ölçme ve izleme; ölçülen performansın artırıcı işlemler yapma; en az hata ile çalışma gibi yetenekleri kazandırmayı hedefler [10]. Bu kapsamda PSP, yazılımın büyüklüğü, yazılım geliştirme süresi, hata sayısı gibi ölçümlerin belirli formlar yardımı ile yapılarak bireysel performansın kayıt altına alınmasını öğretmektedir.

PSP kullanan yazılım geliştiricileri bireysel yazılım geliştirme süreçlerinin performansını topladıkları ölçümlere dayalı olarak analiz edebilirler. Buna bağlı olarak kendi geliştirme süreçlerini iyileştirebilir ve geliştirme işleriyle ilgili Proxy tabanlı kestirim yöntemi (PROBE - PROxy Based Estimating Method) ile hassas kestirimler yapabilirler [28].

3.1. PROxy Tabanlı Kestirim Yöntemi

Proxy tabanlı kestirim yöntemi, büyüklüğü tarif edebilmek için kullanılan vekil nesnelere (proxy) dayalı olarak ürünü küçük parçalara ayırarak geliştirme sürelerini öngörmeyi amaçlayan bir tahmin yöntemidir. PSP yöntemini benimseyen yazılım geliştiricileri ürünlerini tip ve büyüklük olarak küçük parçalara ayırarak kategori ederler. Her bir parça için geçmişe yönelik ölçümlemeler yaparak tahmin veri tabanını oluştururlar. Yeni bir tahmin yapılması gerektiği zaman, yazılım geliştiricisi oluşturmuş olduğu tahmin veri tabanında yer alan kategorilere göre geliştireceği ürünü küçük parçalara böler ve ilgili parça için tahmin veri tabanından büyüklüğünü seçerek tahmin yapar. Tahminlerin iyileştirilmesine yönelik olarak istatistik yöntemleri kullanılmaktadır.

3.1.1. Vekil Nesne Nedir?

Watts S. Humprey'in bahsettiği inşaat örneğine göre: Bir ev inşaatında, yaşam alanının metrekaresi inşaat maliyetlerini tahmin etmek için temel oluşturmaktadır. Ancak, çok az kişi metrekare bakımından istedikleri ev görselleştirebilir. Aynı sorun yazılımda da bulunmaktadır. Planlanan yazılım ürünündeki LOC (Line of Code) sayısı doğru olarak değerlendirilir ise geliştirme ile ilgili iyi tahminler yapılabilir. Ne yazık ki, çok az insan bir yazılım gereksinimi karşılamak için gerekli Kod Satır Sayısını (LOC) doğrudan doğruya anlayabilir [33].

Vekil nesneye olan ihtiyaç ürünün büyüklüğünü tahmin etmek için fonksiyonların tanımlanması ve görselleştirilmesi ile ilgilidir. Vekil Nesne (proxy) bir temsilci veya benzer olandır. Örnek vekil nesneler nesneler (objects), ekranlar, dosyalar, betikler veya işlev noktaları (function points) olabilmektedir. Vekil nesneler yazılım büyüklüğünü tarif edebilmek için kullanıldığı için vekil nesnelerin sayısal olarak ifade edilebilir olması veya boyutlaştırılabilir olması beklenmektedir.

3.1.2. Vekil Nesne Seçimi

PROBE metodunun en temel adımını Vekil Nesnelerin seçimi oluşturmaktadır. Vekil Nesnelerin seçiminin doğru yapılması bizim doğru tahmin sonuçları elde etmemizi birebir etkiyecek bir unsudur. Aşağıdaki kriterler göz önünde bulundurularak vekil nesne seçimi yapılmalıdır [33]:

- **Geliştirilecek proje için ihtiyaç duyulan eforu temsil edebilir olmalı:** Vekil nesne, kullanışlı olmalı, ürünü geliştirmek için gereken kaynaklar ile açık bir şekilde yakın ilişkisi olmalı. Vekil nesnenin büyüklüğü tahmin edilerek işin büyüklüğü doğru bir şekilde yargılanabilir. Geliştirilmiş olan bir dizi ürün ile elde edilen geçmiş veriler ile geliştirme maliyetleri karşılaştırılarak vekil nesnelerin etkinliği değerlendirilmeli. Seçilen vekil nesnenin kullanışlı olabilmesi için projenin büyüklüğü ile bağlantılı olmalı yani projenin büyüklüğü bu vekil nesne ile tarif edilebilmeli.
- **Otomatik olarak içeriği sayılabilmeli.** Yeni tahminler yapabilmek için geçmiş vekil nesnelere ihtiyaç duyulduğu için büyük miktarda vekil nesne verisinin olması arzu edilmektedir. Dolayısı ile, vekil nesnenin otomatik olarak sayılabilir olması ve vekil nesnenin tam olarak tanımlanabilen fiziksel

bir varlık olması gerekmektedir. Seçilen vekil nesnenin sayısal olarak ifade edilebilir olması ve algoritmik olarak tanımlanabilmesi tercih edilmektedir. Vekil nesnenin diğer vekil nesne kriterlerini karşıladığı sürece tanımlanmasının kolay olması gerekmez. Bir programın vekil nesne içeriği otomatik olarak sayılmıyor ise belirli geliştirme süreci ve tasarım için özelleştirilmiş tahmin faktörü üretmek için gereken istatistiksel verileri güvenilir bir şekilde elde etmenin uygun bir yolu yoktur.

- **Projenin başında kolayca görselleştirilebilir olmalı,** Vekil nesnenin, geliştirme için gerekli programcı maliyeti olarak görselleştirilmesi zor olacak ise doğrudan maliyet tahmini yapılabilir. Dolayısı ile vekil nesnenin kullanışlılığı geçmiş deneyimler ve tercihlere bağlıdır. Bir tahminde, geçmişe yönelik uygun veriler varsa birkaç farklı vekil nesne kullanılabilir.
- **Kullanan organizasyonun ihtiyaçları için özelleştirilebilir olmalı;** Organizasyonlar için en büyük zorluk, bir grup tarafında bir tahmin yöntemi veya geliştirme planlaması için tutulan/kullanılan verinin diğer gruplar tarafından da kullanılması gerekliliğidir. Bu esas olarak kaynak modellerinde bir problem olsa da, vekil nesnelerin kullanımı için de geçerlidir. Bunun için benzer projelere ait verileri toplamak ve kullanmak önemlidir. Bu, büyük kuruluşların her büyük yazılım ürün türüne ait büyüklük ve kaynak veri tabanına sahip olması gerektiğini önerir.
- **Uygulamanın varyasyonlarına hassas olmalı,** bu sorun zorlu bir dengeyi gerektirmektedir. Bir projenin başlangıcında en kolay görselleştirilen vekil nesneler girişler, çıkışlar, dosyalar, ekranlar ve raporlar gibi uygulamanın öğeleridir. Ne yazık ki, iyi bir geliştirme tahmininde inşa edilecek ürüne ve bunun inşa edilmesi ile ilgili yakından ilgilisi olan kuruluşlar gereklidir. Bu gereksinim aynı zamanda verilerin vekaleten ve her uygulama dili için ürün boyutunun, tasarım stilinin ve uygulama kategorisinin uygun olmasını gerektirir. Bu nedenle, bir projede kullanılacak dil, tasarım stilleri ve uygulama kategorilerinin kullanılacak tahmin faktörlerini hesaplamak için gerekli verilerin temsil edilen projede gösterilmesi esastır.

3.1.3. Olası Vekil Nesneler

3.1.2 inci bölümde belirtilen kriterler göz önüne alındığında bu kriterleri sağlayan birçok vekil nesne olabileceği görülmüştür. Bunlar;

- İşlev Noktaları (Function points)
- Nesneler (Objects)
- Ekranlar (Screens)
- Dosyalar (Files)
- Betikler (Scripts)
- Doküman Bölümleri (Document Chapters)
- Raporlar (Reports)

olabilmektedir. Bu bağlamda Watts S. Humprey yaptığı çalışmalarda kullandığı örnek PSP araştırma işi için nesne ve doküman bölümlerine ait veriler toplamıştır. Humprey'in çalışmaları için bu unsurlar genel olarak vekil nesne kriterlerini karşılamaktadır. Ancak genel sonuçlar çıkarmak için ekranlar, raporlar ve senaryolara ilişkin yeterli veri bulunmamaktadır. Ayrıca, bu vekillerin kullanışlığının uygulama ve programlama diline bağlı olduğu görülmektedir.

3.1.4. Verilerin Toplanması ve Ölçülmesi

Bireysel Yazılım geliştirme süreci verilere dayanmaktadır çünkü bireyler çalışma biçimlerini ve ne yaptıklarını anlamadıkları sürece iş süreçlerini geliştirememektedir. Toplanan veriler, iyileştirme gereksinimi olan alanları belirlemek için ve iyileşme sürecindeki değişimi ölçmek için referans noktası olarak kullanılmaktadır. Verilerin toplanması ve analizinin faydaları aşağıdaki gibidir [22]:

- Ürünler ve süreçler için standartların oluşturulması
- Belirli bir ürün veya sürecin belirlenen kriterleri karşılayıp karşılamadığına karar verilmesi
- Bireylerin çalışmalarını hassas bir şekilde kontrol etmesi
- Bireylerin performans göstergelerinin geliştirilmesi
- Üretilen ürünlerin kalitesinin yönetilmesi
- İşin ne zaman biteceğini tahmin etmek
- İşi hassas bir şekilde planlamak, izlemek ve raporlama

En kullanışlı verileri elde etmek için aşağıdaki yönergelerin takip edilmesi faydalı olmaktadır [22]:

- Veri toplama sürecinin belirli hedef ve planları olmalıdır.
- Gerçek veriler, bir modelin uygulaması veya bir hipotezin test edilmesi ile bağlantılı olarak seçilmelidir.
- Veriler, onu kullanacak insanlar tarafından toplanmalıdır ve veri toplamanın önemini anlamalı ve alakalı bilgileri toplamaya özen göstermelidir.
- Veri toplama sürecinde, veri toplamanın organizasyon ve çalışanlar üzerindeki etkileri göz önünde bulundurulmalıdır.
- Veri toplama planı yönetimsel desteğe sahip olmalıdır; yönetim, veri toplama işlemini, ürün geliştirme maliyetlerini ve çizelgelerini doğru bir şekilde tahmin edebilmenin yanı sıra, bir kuruluşun verimliliğini ve ürünlerin kalitesini iyileştirmek için bir temel sağlamak açısından potansiyel olarak yüksek kazanç sağlayan bir yatırım olarak görmelidir.

3.1.4.1. Zaman Ölçümü

İş yapılırken izlenmekte ve her birim işi için zaman bilgisi dakika cinsinden ölçülmektedir. Şekil 3 de verilmekte olan Zaman Kayıt Etme Formları kullanılarak kayıt edilmektedir.

Temel bileşenleri;

- Başlangıç günü (start date)
- Başlangıç zamanı (start time),
- Bitiş günü (end date),
- Bitiş zamanı (end time),
- Kesme zamanı (interrupt time),
- Görev dışı zaman (off-task time),
- Delta Süresi (delta time)

Her bir aşamadaki süre, işlemin belirli bir aşamasında harcanan planmış veya gerçeklemiş süredir [33]:

- Kesme süresi (Interrupt time) bir görev veya işlem aşaması için süre ölçümüne dahil değildir. İş sırasında kesinti var ise, bu zaman ölçümünden çıkarılır.

- Görev dışı zaman (Off-task time) planlanan proje görevlerinden başka işler için harcanan zamandır, Belirtilen zamanlama hedeflerinin karşılanmasına katkıda bulunmadığı için genellikle ölçülmez veya izlenmez. Görev dışı zaman, yönetim ve idari toplantılarda harcanan zamanı, eğitim sınıflarına katılmayı, mailleşme veya bir ekip üyesinin yapması gereken diğer önemli etkinliklerden herhangi birini içerir. Belirli bir görev veya çalışma periyodu için görev dışı zaman, bir görev üzerinde harcanan toplam geçme süresinden toplam delta süresinin çıkarılması ile bulunur.
- Delta süresi (Delta time) bir görev veya işlem aşamasını tamamlamak için gerekli gerçek süredir. Bitiş zamanı eksi başlangıç zamanı olarak hesaplanır.

Student Student 3 Date 1/19/94
Instructor Humphrey Program # 1A

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments
1/19	16:25	16:30	0	5	plann'g	
	16:35	17:05	0	30	design	
	17:05	17:40	3	32	code	phone call
	17:40	17:55	0	15	compile	
	17:55	18:00	0	5	test	
1/21	9:25	9:30	0	5	pm	
1/24	15:40	15:55	0	15	pm	
	17:15	17:25	0	10	pm	

Şekil-3: Zaman Kayıt Etme Formu [33]

3.1.4.2.Büyüklük Ölçümü

Büyüklük ölçüsü, bir çalışma ürününün ne kadar büyük olduğunu ölçmek için kullanılır. Büyüklük ölçümleri, çalışma ürününe uygun olacak şekilde seçilir. Örneğin, metin sayfaları için sayfalar kullanılır (kelimeler veya harfler vb.), yazılım bileşenleri için programlama görevleri ve dili kullanılabilir. Büyüklük ölçüm verileri olabildiğince gerçek zamanlı olarak toplanmalı. Çünkü olaydan sonra kayıt edilen verilerin hatalı olma olasılığı çok yüksektir. Büyüklük ölçümleri, yalnızca son teslim edilen ürünler için değil aynı zamanda ürünün parçaları ve ara sürümleri için de geçerlidir [22].

Büyük projelerde genellikle çok çeşit ürün bulmaktadır. Koda ek olarak, belgeler, veri tabanı veya diğer destek paketleri de olabilmektedir. Bu ürünlerin hepsi geliştirme eforu gerektirmektedir, bunun içinde ihtiyaç duyulan kaynakların planlanması gerekmektedir. Bu nedenle, çeşitli büyüklük ölçümü türlerini bulmak önemli olmaktadır. Örneğin; dokümantasyon sayfaları, test senaryoları veya veri tabanı kayıtları. Bu tür büyüklükler, bunları geliştirmek için ihtiyaç duyulan saatler ile makul derecede ilişkili olmalıdır. Ayrıca bu ürünlerin büyüklükleri tahmin edilebilir olmalıdır [33].

SEI'deki Yazılım Süreci Ölçüm Projesi (The Software Process Measurement Project) yazılım büyüklük ölçümünü tanımlamak için bir çerçeve geliştirmiştir. Bu çalışma için iki temel unsur aşağıdaki gibidir [26]:

- İletişim (Communication): Birisi bir ölçüyü tanımlamak ya da bir ölçüm sonucunu tanımlamak için yöntemlerimiz kullanıyorsa, başkaları ne ölçmüş ve neyi dahil etmiş veya neyi dahil etmemiş olduğunu biliyor olacak mı?
- Tekrarlanabilirlik (Repeatability): Bir başkası ölçümü tekrarlayıp aynı sonuca ulaşmış mı?

Bu hedefleri gerçekleştirirken SEI, LOC metriğini tam olarak tanımlamak için bir çerçeve oluşturmuştur. Geliştirilen formun basitleştirilmiş hali Şekil-4 de gösterilmektedir. LOC sayım standartları belirtilirken, bu form önemli maddeleri tanımlamak için kullanılmaktadır. Bu formdaki çeşitli girdilerin anlamları ve kullanımları aşağıdaki gibidir [26]:

- Tanım Adı (Definition name): Standartta verilen ad

- Dil (Language): Kullanılan dil, örneğin C, C++, veya Ada
- Yazar (Author): Yazarın ismi
- Tarih (Date): Standartın üretilme tarihi
- Sayma Tipi (Count Type): mantıksal (logical) veya fiziksel (physical) olarak iki seçenek bulunmaktadır.
    - Mantıksal LOC dil öğelerini sayar
    - Fiziksel LOC metin satırlarını sayar
- Durum Tipi (Statement Type):
    - Çalıştırılabilir (Executable): Çalıştırılabilirlik ile ilgili bilgiler Şekil-4 de verilmekte olan çizelgede Açıklamalar (Clarification) altında yer almaktadır.
    - Tanımlar (Declarations): Birçok seçenek bulunmaktadır. Bildirimler dahil edilmeye karar verildiğinde bunların nasıl sayılacağına karar verilmesi gerekmektedir. Örneğin, tek bir değişken, birden fazla değişkeni ifade ediyor ise bunu bir LOC olarak mı saymalı yoksa birden fazla mı? Ayrıca, başlıklar (headers) nasıl sayılmalı? Burada belirtilen her değişken ve her prosedür tanımlaması verilediği her zaman sayılmaktadır.
    - Derleyici Yönergeleri (Compiler Directives): Derleyici yönergeleri saymanın en iyi yolu ile ilgili net bir karar bulunmamaktadır. Sayılmayabilirde sayılabilir de. Bu çalışma kapsamında sayılmaktadır.
    - Yorumlar (Comments): Mantıksal dil öğeleri için olsa da fiziksel dil öğeleri için olasa da yorumlar LOC olarak sayılmamaktadır.
    - Boş Satırlar (Blank Lines): Boş satırlar sayılmamaktadır.

Definition Name: Example C++ LOC Std. Language: C++

Author: W. S. Humphrey Date: 12/20/93

Count Type	**Type**	**Comments**
Physical/Logical	Logical	
Statement Type	**Included**	**Comments**
Executable	yes	
Nonexecutable:		
Declarations	yes, notes 3,4	
Compiler directives	yes, note 4	
Comments	no	
On own lines	no	
With source	no	
Banners	no	
Blank lines	no	
Clarifications		**Examples/Cases**
Nulls		
Empty statements	yes	";;", lone ;'s, etc.
Generic instantiators		
Begin. . .end	note 1	when executable
Begin. . .end	note 1	when not executable
Test conditions	yes	
Expression evaluation	yes	when used as sub program arguments
End symbols	notes 1, 2	when terminating executable statements
End symbols	notes 1, 2	when terminating declarations or bodies
Then, else, otherwise	note 1	
Elseif	yes	
Keywords	yes	procedure division, interface, implementation
Labels	yes	branch destinations when on separate lines

Şekil-4: Büyüklük Kayıt Etme Formu [33]

3.1.5. Tahmin Veri Tabanının Oluşturulması

Probe tahmin yöntemi geliştirdiğimiz nesnelerin boyutları hakkında geçmiş verileri bulundurmayı ve bu verilerin kategorilere bölünmesini gerektirir [33]. Planlanan yeni ürünün büyüklüğünü değerlendirmek ve bir çerçeve oluşturmak için nesnenin boyutunu temsil edecek kategorilere ihtiyaç duyulmaktadır. Örneğin, bir programcı daha önce geliştirdiği tüm nesnelerin boyutları biliniyor olsaydı geliştirilecek yeni bir nesnenin boyutunu daha iyi görebilirdi. [33]:

Nesnelerin vekil nesne olarak kullanılması için verilerin kategorilere ve büyüklük aralıklarına bölünmesi gerekmektedir. Bölüm 3.1.1. de verilen ev inşaatı örneği bunun neden yapıldığını açıklamaktadır. Evin büyüklüğü tahmin edilirken, Müteahhit'in büyük odaların büyüklüğünü ve küçük odaların büyüklüğünü ve alıcıların hangisini tercih edeceğini düşünmeye ihtiyacı vardır. Müteahhit'in oda kategorileri hakkında düşünmesi bile önemlidir; örneğin büyük banyo en küçük oturma odasından bile küçük olacaktır.

Tablo 1. PSP Örnek Tahmin Veri tabanı [33]

C++ Class Size in LOC per Item					
Category	**Very Small**	**Small**	**Medium**	**Large**	**Very Large**
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

Bir nesnenin ne kadar LOC'e sahip olduğu tahmin edilirken, ev inşaatı örneğine benzer şekilde işlevsel kategorilere ayrılması gerekir. Daha sonra, yeni ürün için kaç tane nesne gerektiğini ve kategorideki her bir nesnenin göreli boyutu değerlendirilerek tahmin yapılır.

Planlanan ürün için, yeni neslenelerin büyüklüğünü değerlendirmek ve bir çerçeve sağlamak için nesne boyutu kategorilerine ihtiyaç duyulmaktadır. Örneğin, daha önce geliştirilen tüm metin nesnelerinin büyüklüğü biliniyor olsaydı, yeni bir metin nesnesinin boyutu daha iyi görülebilirdi.

Watts S. Humprey'nin çalışmasındaki örnekte C++ metin nesnelerini 6 fonksiyonel kategoriye bölmüştür (Bknz. Tablo 1). Her bir kategoriyi kendi içinde değerlendirerek 5 büyüklük kategorisine ayırmıştır.

Her bir kategorinin büyüklüğü kendi içerisinde değerlendirilir ve verilerin dağılımı göz önünde bulundurularak kaç alt gruba ayrılacağına karar verilir. Bu sınıflandırmayı yapmak için her bir veri grubunun ortalaması ve standart sapması bulunur. Elimizdeki verilerin durumuna göre burada logaritmik standart sapma da kullanılabilinir. Bunun için aşağıdaki adımlar takip edilerek her bir grubun büyüklüğünü temsil edecek sayısal değer bulunmuştur.

1. Veri grubunun ortalaması bulunur.
2. Veri grubunun Standart Sapma ve Varyansı bulunur.

   Var = $\boldsymbol{\sigma^2} = (\frac{1}{n}) \sum_{i=1}^{n} (x_i - x_{avg})^k$

   StdDev = $\sqrt{Var} = \sigma$
3. Veri grubu 5 kategoriye ayrılır. Burada Humprey'in örneği göz önünde bulundurularak 5 kategoriye ayrılmasından bahsedilmektedir.
   - Çok Küçük (Very Small) = Ortalama - 2 σ
   - Küçük (Small) = Ortalama – σ
   - Ortalama (Medium) = Ortalama
   - Büyük (Large) = Ortalama + σ
   - Çok Büyük (Very Large) = Ortalama + 2σ.

3.1.6. PROBE Yöntemi ve Adımları

The PROxy-Based Estimating (PROBE) methodu vekil nesne olarak nesneleri (objects) kullanarak geliştirilecek yazılımın ne kadar sürede tamamlanacağına yönelik olarak zaman ve geliştirilecek yazılımın büyüklüğünü tahmin etmektedir. Yöntemin adımları aşağıdaki gibidir.

1. Kavramsal Tasarım (Conceptual design)
2. Nesnelerin Tanımlanması (Identify Objects)
3. Öngörülen LOC'in Hesaplanması (Calculate projected LOC)
4. Programın Büyüklüğünün Tahmini (Estimate Program Size)
5. Geliştirme Süresinin Tahmini (Estimate Development Time)

3.1.6.1. Kavramsal Tasarım

Büyüklüğünü tahmin etmek istediğimiz ürünün doğru bir şekilde yansıtılması için kavramsal bir tasarım ile başlamak gerekmektedir. Bu tasarım bir ön tasarım yaklaşımı oluşturur ve beklenen ürün nesneleri ve işlevleri adlandırılır. Buradaki amaç, eksiksiz bir tasarım yapmak değildir, ihtiyaç duyulan nesneler ve gerçekleştirebileceği işlemler için varsayımda bulunmaktır [33].

Kavramsal tasarım, bir ürünün öğelerinin ve işlevlerinin üst düzeyde bir önermesidir. Kavramsal tasarım, arzulanan bir ürünü büyük bölümlerine ayırır. Kavramsal tasarım yalnızca büyüklük ve efor tahmini üretmek için bir temel olarak kullanılır. Gerçek ürünün nasıl tasarlandığını ve inşa edildiğini bire-bir yansıtmayabilir [22].

3.1.6.2. Nesnelerin Tanımlanması

Kavramsal tasarım yapıldı ve nesneler belirlendi. Daha sonra tahmin veri tabanında bu nesnelerin her birine en çok benzeyen nesne bulunur. Her bir yeni nesne için, tahmin veri tabanında kendi kategorisindeki nesnelerin büyüklükleri karşılaştırılır. Bu karşılaştırmaya dayanarak yeni nesnenin hangi büyüklük aralığına düşeceğine kabaca karar verilir.

Örneğin, Şekil 5 deki büyüklük tahmin örneğinde gösterilen Öğrenci 12 (Student 12) için 10A isimli C++ programına yönelik yapılan tahmin düşünüldüğü zaman kavramsal tasarım kullanılarak, öğrenci her bir nesnenin kategorisini ve büyüklüğünü belirler. İlk nesne, Matrix veri türündendir. Öğrenci, daha sonra bu nesnenin kaç tane yöntem veya işleme ihtiyaç duyabileceğini tahmin eder. Bu örnekte öğrenci, Matrix nesnesinin Data yöntemine sahip olabileceğini ve büyüklüğünün Orta (Medium) olacağını öngörür, daha sonra Tablo-1'deki verilmekte olan Tahmin Veri tabanından bakarak metot başına 8.84 LOC olacağına karar verir. Bu durum için verilen yöntem sayısı ile çarpıldığında toplamda 114.9 LOC olduğu görülür. Sonrasında öğrenci bu süreci tüm yeni nesneler için uygular.

3.1.6.3. Öngörülen LOC'in Hesaplanması

Yeni bir nesne için tahmin yapılırken temel programın büyüklüğüne ve yapılan değişiklerin büyüklüğüne karar verilir.

Şekil-5 verilmekte olan Humprey'in örnek büyüklük tahmin şablonuna göre 12. Öğrencinin 10A programı için LOC hesaplaması aşağıdaki gibidir.

- The Base (B) = 695 LOC
- Deleted (D) = 0
- Modified (M) = 5
- The Base Addition (BA) = 0.
- 3 tane yeni nesne (New Objects - NO) bulunmaktadır. Bunlar toplamda 361 LOC dir. Ve bunların 49 LOC tanesi yeni kullanılmıştır (New Reused).
- Yani Estimated Object LOC (E) = 0+361+5 = 366

Student	Student 12	Date	5/1/94
Instructor	Humphrey	Program #	10A

BASE PROGRAM LOC	ESTIMATE	ACTUAL
BASE SIZE (B) => => => => => =>	695	695
LOC DELETED (D) => => => => => =>	0	0
LOC MODIFIED (M) => => => => => =>	5	18

OBJECT LOC

BASE ADDITIONS	TYPE[1]	METHODS	REL. SIZE	LOC	LOC
TOTAL BASE ADDITIONS (BA) => => => => => =>					

NEW OBJECTS	TYPE	METHODS	REL. SIZE	LOC (New Reused*)	
Matrix	Date	13	Medium	115	136
Linear System	Calc.	8	Large	197	226
Linked List	Data	3	Large	49*	54*
Control	Logic	2			48
TOTAL NEW OBJECTS (NO) => => => => => =>				361	464

REUSED OBJECTS

Linked List	73	73
Data Entry	96	96
REUSED TOTAL (R) => => => => => =>	169	169

		SIZE	TIME
Estimated Object LOC (E):	E = BA + NO + M	366	
Regression Parameters:	β_0 (size and time)	62	108
Regression Parameters:	β_1 (size and time)	1.3	2.95
Estimated New and Changed LOC (N):	N = $\beta_0 + \beta_1$*E	538	
Estimated Total LOC:	T = N + B − D − M + R	1397	
Estimated Total New Reuse (sum of * LOC):		49	
Estimated Total Development Time:	Time = $\beta_0 + \beta_1$*E		1186
Prediction Range:	Range	235	431
Upper Prediction Interval:	UPI = N + Range	773	1617
Lower Prediction Interval:	LPI = N − Range	303	755
Prediction Interval Percent:		90%	90%

[1]L-Logic, I-I/O, C-Calculation, T-Text, D-Data, S-Set-up

Şekil-5: Büyüklük Tahmin Örneği [33]

3.1.6.4. Programın Büyüklüğünün Tahmini

Kavramsal olarak tasarladığımız her bir nesnenin yukarıda bahsedilen adımlar uygulanarak LOC cinsinden büyüklüğü bulunur. Tüm objelerin büyüklükleri toplanır bu değer Tahmini Nesne LOC'si (Estimated Object LOC) olarak adlandırılır. Aslında bu büyüklük bize tahmini olarak projemizin büyüklüğünü vermektedir. Ancak, geçmiş veriler kullanılarak büyüklüğün daha güvenilir ve hata payını düşük tahmin edilmesi mümkündür. Bunun için lineer regresyon yöntemleri kullanılarak tahminde iyileştirilme yapılabilmektedir. Formül (3) kullanılarak geçmişte kayıt altına alınan, Tahmini Nesne LOC'si karşılık gelen Gerçek Toplam Program LOC (Actual Total Program LOC) arasındaki bağ bulunur. Böylelikle **tahminleri** iyileştirmek için kullanılabilecek tahmin parametreleri $\boldsymbol{\beta_0}$ ve $\boldsymbol{\beta_1}$ elde edilmektedir.

$$\text{Program Büyüklüğü} = \boldsymbol{\beta_0} + \boldsymbol{\beta_1} * E \qquad (3.1)$$

Daha genel olarak ifade edilirse $\boldsymbol{\beta_0}$ ve $\boldsymbol{\beta_1}$ tahmin parametreleri aşağıda ifade edilen genel lineer regresyon yöntemleri hesaplanılmaktadır.

$$\mathbf{y_k} = \boldsymbol{\beta_0} + \mathbf{x_k}\,\boldsymbol{\beta_1} \qquad (3.2)$$

$$\boldsymbol{\beta_1} = \frac{\sum_{i=1}^{n} \mathbf{x_i y_i} - \mathbf{n x_{avg} y_{avg}}}{\sum_{i=1}^{n} \mathbf{x_i^2} - \mathbf{n x_{avg}^2}} \qquad (3.3)$$

$$\boldsymbol{\beta_0} = \mathbf{y_k} - \mathbf{x_k}\,\boldsymbol{\beta_1} \qquad (3.4)$$

3.1.6.5. Geliştirme Süresinin Tahmini

Programın büyüklüğü LOC cinsinden tahmin edildikten sonra geliştirme süresinin tahmini için sırası ile aşağıdaki adımlar uygulanmaktadır [22].

1. Programın büyüklüğü LOC cinsinden tahmin edilir.
2. Geçmişe yönelik tutulmuş yeterli verinin olduğu kontrol edilir.
3. Regresyon hesaplaması için yeterli tahmini nesne LOC verisi olduğu kontrol edilir.
4. Regresyon hesaplanması için yeterli tahmin verisinin bulunup bulunmadığı belirlenir.
5. Geçmişe yönelik verimlilik LOC-Saat olarak hesaplanır.
6. Yeni program için gerekli süre hesaplanır.
7. En kısa ve en uzun olası süreler hesaplanır

8. Süre tahmini yapılır ve aralık belirlenir
9. Tahmini yeni ve değiştirilmiş LOC ve gerçek saatler ile regresyon hesaplaması yapılır.
10. Yeni programı geliştirmek için gerekli süre hesaplanır.
11. Bu Tahmin için tahmin aralığı hesaplanır.
12. Süre tahmini ve aralığı belirlenir.
13. Tahmini nesne LOC ile geliştirme saatleri arasında regresyon yapılır
14. Yeni program için gerekli süre hesaplanır
15. Bu tahmin için tahmin aralığı hesaplanır
16. Süre ve aralık belirlenir.

BÖLÜM 4

GELİŞTİRİLEN METOT

Bu çalışmada bankacılık alanında çalışmakta olan bir test analistinin kişisel test çalıştırma süresinin testin analiz ve tasarım aşamasında iken öngörülmesi (tahmin edilmesi) hedeflenmektedir. Buna yönelik olarak bir önceki bölümde özetlenen Kişisel Yazılım Süreci (PSP) ve Proxy Tabanlı Tahmin Yöntemi test sürecine uyarlanarak test çalıştırma süresi tahmin metodu oluşturulmuştur. Diğer bir deyişle çalışmada amaçlanan test analistinin kendi kişisel sürecini ölçümleyerek kendi test çalıştırma süresini tahmin etmesi sağlamaktır.

4.1. Test Çalıştırma Aşamasının Bileşenleri ve Ölçülmesi

Süreçler girdileri bir ya da birden fazla dönüşüme uğratarak çıktılara çeviren işlemler dizisidir. Bir süreci tanımlamak için sürecin bileşenlerinin belirlenmesi gerekir. Sürecin bileşenleri aslında sürecin kimlik kartıdır. Test çalıştırma aşaması da kendi içerisinde test girdileri ve bunlara bağlı olarak çıktılar içeren bir süreçtir. Üzerinde çalıştığımız yöntem içinde test çalıştırma sürecinin süre içerikli faaliyetlerinin tespit edilmesi gerekmektedir.

Bu amaç ile söz konusu test analistinin test çalıştırma süreci izlenmiş ve aşağıdaki faaliyetlerden oluştuğu tespit edilmiştir.

- **İdeal şartlarda test çalıştırma süresi**, test analistinin bilgi gereksinimi olmadan ve test nesnesine ait bulgu çıkmadan testin çalıştırıldığı süredir.
- **Bulgu düzeltme süresi**, test nesnesine ait bulgu çıkması durumunda yazılımcının bulguyu düzeltme süresidir.
- **Kayıplar**, test çalıştırma sürecinde yazılım ve analist ekipleri ile yapılan bilgi alışverişleri, mailleşmeler; bulguların raporlanması kayıp süre olarak değerlendirilmektedir.

Test çalıştırma sürecinin izlenmesi ve bileşenlerinin çıkarılması, takriben bu bileşenlerin test çalıştırma sürecine katkılarının tespit edilmesi için bazı verilere ihtiyaç duyulmaktadır. Bu veriler, test çalıştırma süreci ölçümlenerek elde edilmektedir.

Ölçümlemenin sağlıklı yapılabilmesi için PSP'nin yazılım geliştirme süreci için önermekte olduğu Süre Kayıt Etme Formu (Time Recording Log) test sürecine uyarlanmış olup Test Analisti bu formu kullanarak her bir bileşenle ilgili harcadığı süreleri kendisine atanan test projeleri için kayıt altına almaktadır. Test çalıştırma esnasında yer alan üç faz kayıt edilmektedir:

- **Test etme süresi**, Bir test senaryosunun çalıştırıldığı süredir. Bu süre, test süresi kayıt etme formuna (Bkz. Şekil 3) fazı test olarak işlenmektedir.
- **Bulgu düzeltme süresi**, Bulgu çıkması durumunda yazılımcının ilgili test senaryosunda çıkan bulguya ne kadar zamanda döndüğünü gösteren süredir. Bu süre bulgu raporlandıktan yani yazılımcının eline ulaştıktan sonra başlatılmaktadır. Bu süre, test süresi kayıt etme formuna (Bkz. Şekil 3) fazı bulgu olarak işlenmektedir.
- **Yeniden test etme süresi**, Düzeltilen bulgu ile ilgili senaryonun yeniden test edildiği süredir. Bu süre, test süresi kayıt etme formuna (Bkz. Şekil 3) fazı yeniden test etme olarak işlenmektedir.

Test Süresi Kayıt Etme Formu her bir proje için ayrı tutulmaktadır. Formun başlık kısmında yer almakta olan bölümlere geliştirmesi yapılmış olan projenin isim bilgisi, projeye başlanan tarih ve projeyi geliştiren yazılımcı bilgisi not edilmekte olup, bu çalışmada her bir projenin tek bir yazılımcısı olduğu varsayılmaktadır. Forma veriler her bir test senaryosu için kayıt edilmektedir.

TEST SÜRESİ KAYIT ETME FORMU

Test Analisti: G.K. **Tarih:** 16 Ocak 2017

Yazılımcı: Yazılımcı-3 **Proje İsmi:** Proje-5

Tarih	Başlangıç	Bitiş	Test Senaryo Numarası	Test Senaryo Kategorisi	Fazı	Açıklama
Ocak-17	10:00	10:30	1	MK	Test	
	10:45	10:50	2	MK	Test	bulgu çıktı yazılımcıya bildirilecek
	11:50	15:00	2	MK	Bulgu	
	15:00	15:25	2	MK	Re-Test	
	15:45	16:06	3	BC	Test	
	16:00	18:00	-	-	-	toplantı
Ocak-18	09:35	09:40	5	EF	Test	
	09:44	09:50	6	EF	Test	
	09:51	09:59	7	EF	Test	
	10:10	10:12	8	EF	Test	bulgu çıktı yazılımcıya bildirilecek
	10:15	10:30	8	EF	Bulgu	
	10:30	10:39	8	EF	Re-Test	bulgu çıktı yazılımcıya bildirilecek
	10:53	11:05	8	EF	Bulgu	
	11:11	11:25	8	EF	Re-Test	
	11:36	12:10	9	MK	Test	
	13:15	13:40	10	MK	Test	
	14:00	14:35	11	MK	Test	
	15:05	15:40	12	MK	Test	

Şekil-6: Test Süresi Kayıt Etme Formu

4.2. İdeal Şartlarda Test Çalıştırma Süresinin Tahmini

Bir test analistinin bilgi ihtiyacı olmadan ve bulgu çıkmadan testi çalıştırdığı süre ideal şartlarda test çalıştırma süresi olarak tanımlanmaktadır.

Bu sürenin tahmini için Watts Humprey'in PSP yaklaşımı kapsamında yazılım geliştirme sürecinde büyüklük ve süre tahminine yönelik önerdiği PROxy Tabanlı Tahmin Yöntemi test süresince uyarlanmaktadır. Bu kapsamda test sürecini temsil edebilecek vekil nesne(ler) belirlenmektedir. Akabinde vekil nesne büyüklüğünü ölçmek ve ifade etmek üzere kullanılacak parametre tespit edilmektedir.

Yazılım geliştirme süreci düşünüldüğünde iyi bir vekil nesnenin aşağıdaki özellikleri taşıması gerekmektedir [33]:

- Vekil nesnenin büyüklüğü (büyüklük ölçümü) geliştirme için gerekli eforu yakın şekilde temsil edebilmeli,
- Ürünün sahip olduğu vekil nesne adedi otomatik olarak sayılabilir olmalı,
- Vekil nesne projenin başında tahayyül edilebilir olmalı,
- Vekil nesne organizasyonun/projenin ihtiyaçlarına göre uyarlanabilir olmalı,
- Vekil nesne geliştirme eforunu veya maliyetini etkileyebilecek uygulama değişikliklerine karşı duyarlı olmalı.

Bu çerçevede bakıldığı zaman, test süreci için vekil nesnelerin test senaryoları, geliştirilmiş ekranlar, veri tabanı tabloları, ihtiyaç analiz dokümanı bölümleri olabileceği düşünülmüştür. Ancak test analistine gelen yazılım test projeleri incelendiğinde bazı projelerin ekranı bulunmadığı gözlemlenmiştir. Aynı şekilde bazı projeler için de veri tabanı bağlantısı olmadığı tespit edilmiştir. Ekranlar ve veri tabanı tablolarının bütün projelerin ihtiyaçlarına uyarlanabilir olmaması sebebi ile vekil nesne olmaya uygun olmadığı görülmüştür. İhtiyaç analiz dokümanı bölümlerinin (örn. her bir paragraf) büyüklüğünün satır sayısı ile ifade edilerek iyi bir vekil nesne olabileceği düşünülmüştür. Ancak ihtiyaç analiz dokümanı bölümünün büyüklüğü satır sayısı olarak ifade edilse bile bu büyüklük test süresini doğru bir şekilde temsil etmemektedir. Analistlerin ihtiyaç analiz dokümanlarında yazdıkları bir satır bazen bir günlük test süresi gerektirirken bazen de bir sayfalık doküman 1 saatlik test süresi gerektirmektedir. İhtiyaç analiz dokümanı bölümlerinin büyüklük ölçümü test için gerekli süreyi yakından temsil edebilir olmadığından vekil nesne için uygun olmadığı tespit edilmiştir.

Test sürecinin temel prensibi test senaryolarının tasarlanarak bu senaryoların çalıştırılmasıdır. Bu çerçeveden bakıldığı zaman test senaryolarının iyi bir vekil nesne olabileceği düşünülmüştür. Bu durumda bir test senaryosunun büyüklüğü ile bir test senaryosunun dakika olarak çalıştırılma süresi arasında bir ilişki kurulur.

Örnek olarak Bankada görevli bir Test Analistinin kişisel test süreci incelenmiş ve bu analiste atanan test senaryolarının aşağıda listelendiği gibi altı farklı kategoride olduğu gözlemlenmiştir. Her bir test senaryosunun aşağıda verilen kategorilerden sadece birine girecek şekilde ayrıntılı olarak tasarlandığı varsayılmaktadır:

- Eşleşme (MK)
- Birlikte Çalışabilirlik (BC)
- Ekran Tasarımı (ET)
- Ekran Fonksiyonları (EF)
- Çıktıların Kontrolü (CK)
- Uyarı ve Hata Mesajları (UHM)

Söz konusu kategorilere ait senaryo adetlerinin proje bazında gösterdikleri dağılım Tablo 2 de paylaşılmaktadır. Şekil-3'te verilmekte olan form yardımı ile 18 projeye ilişkin 1598 adet test senaryosunun (126 adet Eşleşme, 560 adet Birlikte Çalışabilirlik, 86 adet Ekran Tasarımı, 244 adet Ekran Fonksiyonu, 91 adet Çıktıların Kontrolü, 222 adet Uyarı ve Hata Mesajları) çalıştırma süreleri ölçülmüştür. Akabinde her bir kategoride yer alan ölçüm sonuçları tespit edilmiştir. Kategori bazında toplanan ölçüm sonuçlarının dağılımı analiz edilerek test senaryo büyüklüklerinin Küçük (S), Orta (M) ve Büyük (L) olmak üzere 3 kategoriye ayrılmasına karar verilmiştir (Bkz. Tablo 3).

Toplanan ölçüm sonuçlarının ortalama değer ve standart sapmasının hesaplanması vasıtası ile her bir kategoride yer alan Tablo 3'de verilen Tahmin Veri tabanı elde edilmiştir. Tahmin Veri tabanında yer alan değerler bir test senaryosunun kategorisine göre ortalama çalıştırılma süresini dakika cinsinden göstermektedir.

Tablo 3 de verilmekte olan tahmin veri tabanı bahsedilen 18 projeye ait ölçüm sonuçlarının genel istatistik yöntemleri kullanılması ile elde edilmiştir. Burada yer alan değerler kati değerler değildir. Son projeye ait ölçüm sonuçlarının toplanması durumunda bu değerler değişecektir.

Tablo 2: Proje Bazlı Vekil Nesnelerin Dağılımı

İndeks	Proje Adı	Toplam Senaryo Adedi	Gerçeklenen Senaryo Adedi	Kategorilerde Yer Alan Senaryo Adedi					İhtiyaç Analiz Dokümanı Sayfa Sayısı
				Eşleşme	Birlikte Çalışabilirlik	Ekran Tasarımı	Ekran Fonksiyonları	Çıktıların Kontrolü	Uyarı ve Hata Mesajları	
1	Proje-01	96	91	1	27	14	28	0	21	17
2	Proje-02	76	39	0	39	0	0	0	0	4
3	Proje-03	9	9	0	2	3	1	0	3	1
4	Proje-04	171	106	0	49	0	0	32	25	8
5	Proje-05	206	191	0	109	23	45	0	14	26
6	Proje-06	21	19	0	19	0	0	0	0	1
7	Proje-07	18	15	9	0	1	5	0	0	2
8	Proje-08	28	28	0	0	4	24	0	0	6
9	Proje-09	378	304	0	123	35	122	0	24	30
10	Proje-10	103	99	2	64	0	0	33	0	3
11	Proje-11	30	28	9	1	2	16	0	0	8
12	Proje-12	109	60	0	54	0	0	6	0	3
13	Proje-13	22	21	7	12	0	1	0	1	5
14	Proje-14	77	71	12	12	0	1	0	46	3
15	Proje-15	77	71	12	12	0	1	0	46	2
16	Proje-16	33	29	17	12	0	0	0	0	2
17	Proje-17	77	71	12	12	0	1	0	46	2
18	Proje-18	67	70	44	12	0	0	14	0	*

……..

Tablo 3: Tahmin Veri Tabanı

Her bir senaryonun çalıştırma süresi dakika cinsinden			
Kategori	**Küçük (S)**	**Orta (M)**	**Büyük (L)**
Eşleşme (MK)	8,83	20,38	31,93
Birlikte Çalışabilirlik (BC)	1,47	13,83	26,18
Ekran Tasarımı (ET)	1,13	2,38	5,01
Ekran Fonksiyonları (EF)	1,08	3,07	8,72
Çıktıların Kontrolü (CK)	1,67	5,99	10,30
Uyarı ve Hata Mesajları (UHM)	5,36	10,54	15,71

Her yeni proje için elde edilen ölçüm sonuçlarının daha iyi tahmin sonuçları verip veremeyeceğine yönelik bir hata analizi yapılmadan Tahmin Veri Tabanının bu değerler ile beslenmesi sağlanmaktadır. Ölçüm sonuçları çoğaldıkça Tahmin Veri Tabanının sağlamlaştığı farz edilmektedir.

Çalışmanın devamında genel istatistik yöntemleri yerine Geri Yayılma Algoritması (Backpropagation) kullanılarak ölçümlerin tahmin veri tabanını beslemesinin sağlanması amaçlanmaktadır. Geri yayılım algoritması hataları geriye doğru çıkıştan girişe azaltmaya çalışmaktadır. Böylelikle yeni projeye ait ölçüm sonuçları ile Tahmin Veri tabanının iyileşip iyileşmediği tespit edilebilecektir. İyileştiren değerlerin tahmin veri tabanına beslenmesi yolu ile tahmin veri tabanının sağlamlaştırılması hedeflenmektedir.

Geçmişe yönelik olarak toplanmış verilerin analiz edilmesi ile elde edilen Tahmin Veri tabanı kullanılarak, yeni gelen bir projenin İdeal Şartlarda Test Çalıştırma Süresinin (P) tahmin edilebilmesi hedeflenmektedir. Bu doğrultuda her bir (S) senaryosunun tipi (x) ve büyüklük kategorisi (y) belirlenir, hangi kategoriye girdiği belirlenen senaryonun tahmini çalıştırılma süresi Tahmin Veri tabanından bulunur. Bu işlem projedeki tüm senaryolar (n adet senaryo) için tekrarlanır. İdeal Şartlarda Test Çalıştırma Süresi (P) aşağıdaki gibi ifade edilmektedir.

$$P = \sum_{k=0}^{n} \mathbf{S}(x, y) \qquad (4.1)$$

Test çalıştırma süresi test senaryoları hazırlandıktan sonra tahmin edilebileceği gibi bazı durumlarda ise test senaryoları hazırlanmasının bir adım öncesinde yani ihtiyaç analiz dokümanı incelenirken öngörülmesi ihtiyacı doğmaktadır. Bu durumlarda test

analisti ihtiyaç analiz dokümanını incelerken kontrolü yapılacak durumları imgeleyerek de tahmin yapılabilmektedir. İmgeleyerek yapılan tahminlemelerde de test senaryoları üzerinden tahmin yaparken izlenen aynı adımlar takip edilmektedir.

4.3. Test Çalıştırma Süresinin Tahmini

Test çalıştırma süresi, test nesnesi Test Analistine teslim edildikten ve teste giriş kriterleri sağlandıktan sonra başlayıp, tanımlanmış test senaryolarının sistemli bir şekilde yürütülmesi için gerekli süredir. Bu süre, yazılımcı tarafından bulgunun düzeltilme süresini ve geri gönderilmesi akabinde yeniden test edilme süresini de içerisinde barındırmaktadır. Aynı zamanda bir bulgu tespit edildiği zaman testin doğruluğundan emin olmak için test dokümanlarının yeniden incelenmesi, bulguların raporlanması, bilgi alışverişi ve mailleşmeler de test çalıştırma süresine dahil olmaktadır.

Tespit edilen bulguların düzeltme sürelerinin test çalıştırma süresine etkisinin tespit edilebilmesi için, bu çalışmada 18 tane test projesine ait tespit edilen bulguların düzeltme süreleri kullanılmıştır (Bkz. Tablo 4.) Bahsi geçen 18 projenin her birinin tek bir yazılımcısı vardır. Geliştirilmekte olan model için de her bir projenin tek bir yazılımcısı olduğu varsayımında bulunulmuştur.

Tablo 4. Proje Bazında Bulgu Adetleri ve Ortalama Bulgu Düzeltme Süreleri

İndeks	Proje Adı	Yazılımcı	Bulgu Adedi	Bulgu Yoğunluğu (%)	Ortalama Bulgu Düzeltme Süresi (Saat)
1	Proje-01	Yazılımcı-1	72	79,1	2
2	Proje-02	Yazılımcı-1	8	20,5	3
3	Proje-03	Yazılımcı-1	1	11,1	2
4	Proje-04	Yazılımcı-2	1	0,9	1
5	Proje-05	Yazılımcı-3	35	18,3	1
6	Proje-06	Yazılımcı-2	1	5,3	1
7	Proje-07	Yazılımcı-3	3	20,0	6
8	Proje-08	Yazılımcı-3	5	17,9	2
9	Proje-09	Yazılımcı-4	42	13,8	2
10	Proje-10	Yazılımcı-5	11	11,1	9
11	Proje-11	Yazılımcı-6	8	28,6	10
12	Proje-12	Yazılımcı-4	13	21,7	1
13	Proje-13	Yazılımcı-6	4	19,0	3
14	Proje-14	Yazılımcı-6	6	8,5	2
15	Proje-15	Yazılımcı-6	2	2,8	5
16	Proje-16	Yazılımcı-5	3	10,3	6
17	Proje-17	Yazılımcı-6	6	8,5	4
18	Proje-18	Yazılımcı-7	2	2,9	4

Bulgu düzeltme süresinin iki temel etkene bağlı olduğu gözlemlenmiştir; Yazılımcının bulgu yoğunluğu (λ) ve bulguyu ortalama düzeltme süresi (q).. Buradan yola çıkarak bulgu düzeltme süresi (B) aşağıdaki gibi ifade edilmektedir:

$$B = \lambda * q * n \qquad (4.2)$$

Bulgu yoğunluğu test nesnesinde tespit edilen bulgu sayısının test senaryo sayısına oranıdır

Bulgu yoğunluğu = bulgu adedi / test senaryo sayısı

Tablo 5. Yazılımcı Bulgu yoğunluğu ve ortalama bulgu düzeltme süresi

Yazılımcı	Bulgu Yoğunluğu	Bulguya dönüş süresi
Yazılımcı-1	36,91	2
Yazılımcı-2	3,10	1
Yazılımcı-3	18,73	3
Yazılımcı-4	17,74	2
Yazılımcı-5	10,73	8
Yazılımcı-6	13,47	5
Yazılımcı-7	2,86	4

Çalışmada yer alan 18 proje için 7 farklı yazılımcı ile çalışılmıştır. Gözlemlenen değerlerden (Bkz. Tablo 4) yazılımcıların bulgu yoğunlukları ve ortalama bulguya dönüş süreleri çıkarılmıştır (Bkz. Tablo 5) Tabloda yer alan değerler yeni gelen projelerin dahil edilmesiyle değişebilecek olup nihai değerler değildir.

Bulgular düzeltilip test analistine döndükten sonra yapılan düzeltmelerin başarısını doğrulamak için ilgili test senaryolarının tekrardan çalıştırılması (re-test) ihtiyacı doğmaktadır.

Bir yazılımcının bulgu yoğunluğu oranında, yeniden test etme süresine ihtiyaç duyduğu varsayılmaktadır. Düzenleme yapıldıktan sonra koşulan test senaryolarından da m% oranında bulgu çıkabileceği ve bulgu çıkma oranının 0.001'e yakınsayana kadar t iterasyon boyunca gerçekleyebileceği farz edilmektedir. Bahsi geçen Yeniden Test Etme Süresi (R) aşağıdaki gibi ifade edilmektedir:

$$R = \sum_{k=0}^{t} \boldsymbol{\lambda} * \mathbf{P} * \left(\frac{\mathbf{m}}{\mathbf{100}}\right)^{\mathbf{k}} \qquad (4.3)$$

Çalışmada kayıt altına alınan senaryo başına yeniden test etme (Fazı Re-test olan Kayıtlar - Bkz Şekil 6) adetleri proje bazında değerlendirilerek yapılmakta olan çalışmada düzeltme yapılan senaryolardan 10% oranında (m=10) yeniden test etme ihtiyacı tespit edilmiştir.

Yukarıda bahsedilen sürelere ek olarak, Kayıplar olarak tabir edilen sürelerin Test Çalıştırma Süresine katkısı için sabit bir değer (K) vermek yerine, ideal şartlarda test çalıştırma süreleri dikkate alınmıştır. Bu çerçevede, ideal şartlardaki test çalıştırma sürelerinin [P] dağılımına göre kategoriye ayrılmış ve her bir kategori için K değerinin katkısı olmadan hesaplanan test yürütme süresi [P+B+R] ve Gerçek Test Yürütme sürelerinden de K değerleri hesaplanmıştır.

Yapılan çalışmada, Tablo 6'de verilmekte olan 18 proje için ideal şartlardaki test çalıştırma sürelerinin dağılımı incelendiğinde üç gruba ayrılabileceği öngörülmüştür. Bu doğrultuda elde edilen K değerleri aşağıdaki gibidir;

- K=01,29 saat eğer P < 3,50 saat
- K=14,57 saat eğer 3,50 < P < 13,28 saat
- K=27,84 saat eğer P > 13,28 saat

K değerleri 18 proje için elde edilmiş değerlerdir, toplanan verinin artması ile bu değer değişecek olup iyileşeceği düşünülmektedir.

Gerçek Test Yürütme Süresi Test Analistinin Bankacılık Veri tabanına her bir yazılım testi projesi için girmiş olduğu aktivite bilgilerinden alınmaktadır. Söz konusu süre sadece test faaliyetlerinin yapıldığı zamanı yansıtmaktadır. Proje sürecinde yer alan test analistinin eğitimlerini, izinlerini, molalarını vb. içermemektedir.

Yukarıda bahsedildiği gibi birbirini takip eden adımlardan oluşan ve paralelde başka bir iş için efor sarf edilmediği varsayıldığında Tahmini Test Çalıştırma Süresi (T) aşağıdaki gibi elde edilmektedir:

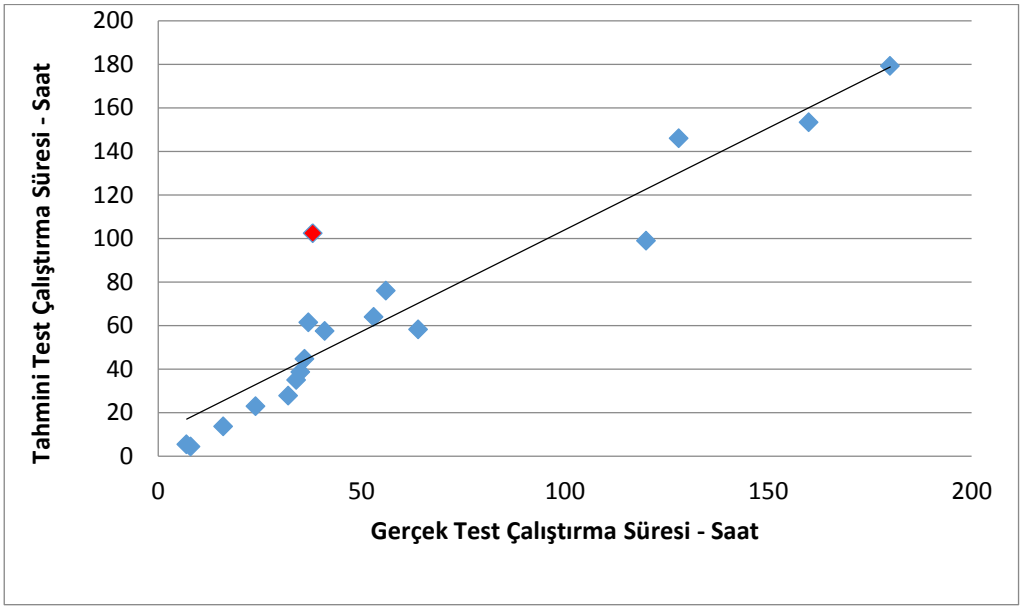
$$T= \sum_{\mathbf{k=0}}^{\mathbf{n}} \mathbf{S(x,y)} + \lambda * q * n + \sum_{\mathbf{k=0}}^{\mathbf{t}} \boldsymbol{\lambda} * \mathbf{P} * (\frac{\mathbf{m}}{\mathbf{100}})^{\mathbf{k}} + K \quad (4.4)$$

Tablo 6'de, geçmişe yönelik ölçümlerin elde ettiğimiz modele yerleştirilmesi sonucunda her bir proje için elde edilen Tahmini Test Çalıştırma Süreleri verilmektedir.

Tablo 6. Tahmini ve Gerçek Test Çalıştırma Süreleri

İndeks	Proje Adı	İdeal Şartlarda Gerçek Test Çalıştırma Süresi (Saat) [P]	Hesaplanan Test Çalıştırma Süresi (Kayıp Süre Eklenmeden) (Saat) [P + B + R]	Tahmini Test Çalıştırma Süresi (Kayıp Süre Eklenerek) (Saat) [T]	Gerçek Test Çalıştırma Süresi (Saat)	Hata Oranı (%)	Yönetici Tarafından Atanmış Test Süresi (Saat)
1	Müşteri Çağırma Ekranları	11	163,70	178,33	180	3,01	20
2	Müşteri Karşılama Ekranı İçin Step Ek. Verl. Ser.	5	30,03	44,65	36	3,11	10
3	Fiktif Bilet Üretme Ekranı	1	3,11	4,46	8	0,28	10
4	Migrator Gmatic İhtiyaçları	12	13,11	27,74	32	1,36	40
5	Kiosk İçin Önyüz Düzenlemeleri	30	70,50	98,40	120	25,92	40
6	Kiosk Randevulu Müşteri	3	4,16	5,50	7	0,10	8
7	Kiosk İzleme Ekranları	3	21,60	22,94	24	0,25	8
8	Gmatic Önyüz Barkod Okuyucu Entegrasyonu	2	12,36	13,70	16	0,37	8
9	Bilgilendirme Kural Tanım Ekranları	36	124,97	152,87	160	11,40	80
10	Maaş Müşterileri Bilgilendirme Süreci	17	117,89	145,79	128	22,77	40
11	Müşteri Bilgilendirmeleri İzleme Ekranları	6	87,71	*	38	*	32
12	Toplu Parametre Alanının Kullanımı	14	30,03	57,93	64	3,88	40
13	KKS	10	23,90	38,53	35	1,23	16
14	KKS-Cgen	22	35,86	63,76	53	5,70	16
15	KKS-Step	19	29,54	57,44	41	6,74	16
16	KKS-Arşiv	14	33,45	61,35	37	9,01	16
17	KKS-Ipad	22	47,86	75,76	56	11,07	16
18	Email Gönderimleri Header Bilgilerinin Step Akt.	12	20,34	34,97	34	0,33	-
Ortalama:		13,28	48,34	63,77	59,39	6,27	24,47
Maksimum:		36,00	163,70	178,33	180,00	25,92	80,00
Minimum:		1,00	3,11	4,46	7,00	0,10	8,00

Tablo 6'te verilmekte olan 18 projeye ait sonuçlar analiz edildiğinde ortaya çıkan Tahmini Test Çalıştırma Süresi ile Gerçek Test Çalıştırma süresi arasında doğrusal bir ilişkinin olduğu görülmektedir (Bkz. Şekil 7.) 11 No'lu proje değerlendirme dışı tutulduğunda Tahmini ve Gerçek Süreler arasındaki ortalama bağıl hata 6,27 % olarak bulunmaktadır. Bu çerçevede bakıldığında, incelenen çalışma için model üzerinde oluşturulan tahminlerin gerçek süreler ile örtüştüğü gözlenmektedir.



Şekil-7 : Tahmini ve Gerçek Test Çalıştırma Süreleri

11 No'lu projede ise farklı bir durum bulunmaktadır. Bahsedilen projenin yazılımcısının bulguları geç düzeltmesinden dolayı Test Analisti projeye ara vermiş ve başka bir iş almıştır. Geliştirilen modelde Test Analistinin sadece tek bir proje ile ilgilendiği varsayımında bulunulmasından dolayı K değeri eklenmeden hesaplanan test çalıştırma süresinin, gerçek test çalıştırma süresinden büyük olduğu gözlenmektedir. Yazılımcının bulguya dönüş süresinin ortalamanın çok dışında olmasının hatalara yol açtığı sonucuna varılmıştır.

BÖLÜM 5

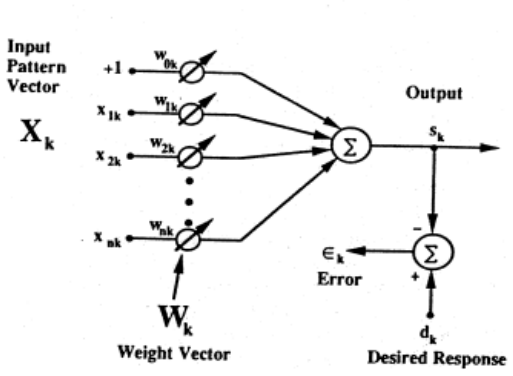
TAHMİN VERİ TABANININ İYİLEŞTİRİLMESİ

Bu çalışma kapsamında Tahmin veri tabanının oluşturulmasında genel istatistiksel (ortalama, standart sapma) yöntemler kullanılmaktadır. Ancak bu yöntemler ile gözlenmiş ideal şartlarda test çalıştırma süresi ile tahmin veri tabanı kullanılarak hesaplanan ideal şartlarda test çalıştırma süresi arasındaki farktan faydalanılmamaktadır. Gözlenmiş ve tahmini süreler arasındaki hatanın geri beslenmesi vasıtası ile Tahmin Veri Tabanındaki değerlerin iyileştirilebilineceği düşünülmektedir. Böylelikle ideal şartlardaki test çalıştırma süresi daha yüksek bir doğruluk ile tahmin edilebilecektir.

Hatanın geri beslenmesi için Yapay Sinir Ağlarının parametrelerinin güncellenmesi için literatürde en çok kullanılan yöntem olan hata geriye yayma yöntemi (Backpropagation) [20] ve Uygulanabilir Lineer Birleştirici (The Adaptive Linear Combiner)'ın bir uygulaması yapılmıştır.

5.1. Uygulanabilir Lineer Birleştirici (The Adaptive Linear Combiner)

Uygulanabilir Lineer Birleştirici (The Adaptive Linear Combiner) Yapay Sinir Ağlarının tüm uygulamalarında kullanılan en temel parçasıdır. [4]



Şekil-8: Uygulanabilir Lineer Birleştirici

Uygulanabilir Lineer Birleştirici şekil-8 de gösterildiği gibidir. Çıktı değeri ($\mathbf{s_k}$) giriş değerlerinin doğrusal bir birleşimidir. Sayısal bir uygulamada, k ıncı zamanda sistem giriş sinyalini veya giriş desen vektörünü $\mathbf{X_k} = [\mathbf{x_0}, \mathbf{x_{1k}}, \mathbf{x_{2k}}, \ldots \mathbf{x_{nk}}]^{\mathbf{T}}$ ve arzu edilen yanıtı $\mathbf{d_k}$ almaktadır. Giriş vektörünün elemanları uyarlanabilen katsayılar yani ağırlık vektörü $\mathbf{W_k} = [\mathbf{w_0}, \mathbf{w_{1k}}, \mathbf{w_{2k}}, \ldots \mathbf{w_{nk}}]^{\mathbf{T}}$ ile ağırlıklandırılmaktadır.

Ağırlıklı girdilerin toplamı daha sonra (5.1.) denklemi ile verilen lineer olan bir çıktı üretmektedir [4].

$$\mathbf{s_k} = \mathbf{X_k^T W_k} \qquad (5.1)$$

$\mathbf{X_k}$ nın elemanları sürekli analog değerler ya da binary değerler olabilir. Ağırlıklar sürekli olarak değişkenlik gösterir, pozitif ya da negatif değerler alabilir. Ağırlıklar uyarlanabilmesi için bir eğitim algoritması kullanılabilir böylece verilen eğitim kümesini deseni için geniş aralıkta yanıtlar alınabilir.

5.2. Hata Geri Yayma Yöntemi

Hata geri yayma yönetiminin temel felsefesi sistemin ürettiği çıkış değeri ile arzu edilen çıkış değeri (desired output) arasındaki farkı/hatayı minimuma indirmektir. Buradaki hata ağırlıklara bağlı olduğu içinde hata geri yayma yöntemi ağırlıkların iteratif olarak en uygun biçimde değiştirmesi işlemlerinden oluşmaktadır.

Gizli katmanı (Hidden layer) olmayan bir ağ yapısı için her bir ağırlık değerinin değişme miktarı (5.2.) de verilen parametre güncelleme kuralı (delta rule) ile bulunmaktadır. [7]

$$\nabla_k w_{ji} = \eta(d_{kj} - s_{kj})\, x_{ki} \qquad (5.2)$$

- $\mathbf{d_{kj}}$: arzu edilen çıkış değerleri
- $\mathbf{s_{kj}}$: sistemin ürettiği çıkış değerleri
- $\mathbf{x_{ki}}$: giriş değerleri
- $\mathbf{\nabla_k w_{ji}}$: ağırlıkta yapılacak değişiklik
- $\boldsymbol{\eta}$: momentum katsayısı

5.3. Tahmin Veri Tabanı İyileştirme

Bu bölümde tahmin veri tabanının hata geri yayma yöntemi ile iyileştirilmesi hedeflenmektedir. Bu amaçla gizli katmanı olmayan, on sekiz girişli ve bir çıkışlı bir ağ yapısı düşünülmüştür.

Ele alınan ağ yapısında giriş değerleri bir projenin 1 kategori ve büyüklüğüne ait ölçüm adetleridir. Tahmin veri tabanında toplamda 6 tane kategori ve 3 tane

büyüklük olduğu için ağın 18 tane girişi bulunmaktadır. $\mathbf{X_k} = [\mathbf{x_0}, \mathbf{x_{1k}}, \mathbf{x_{2k}}, \ldots \mathbf{x_{18k}}]^{\mathbf{T}}$

Giriş değerlerine karşılık gelen ağırlıklar ise tahmin veri tabanında yer alan her bir kategori ve büyüklüğe ait süre değerlerini temsil etmektedir. $\mathbf{W_k} = [\mathbf{w_0}, \mathbf{w_{1k}}, \mathbf{w_{2k}}, \ldots \mathbf{w_{18k}}]$

Ağırlık parametrelerinin ilk değerleri mevcut elde edilen tahmin veri tabanındaki değerler olarak seçilmiştir. Ağırlık parametrelerinin güncellenmesi için (5.2) de verilmekte olan parametre güncelleme kuralı kullanılmıştır.

Buradaki arzu edilen çıkış değeri ($\mathbf{d_{kj}}$) Tablo 5 de verilmekte olan İdeal Şartlardaki Gerçek Test Çalıştırma Süreleridir. Arzu edilen çıkış değerine karşılık gelen sistemin ürettiği çıkış değeri ($\mathbf{S_{kj}}$) İdeal Şartlardaki Tahmini Test Çalıştırma Sürelerini ifade etmektedir.

Sistemin ürettiği çıkış değeri hesaplanması için ağın giriş değerlerinin kategorisi belirli olmasına rağmen büyüklüğü belirli değildir. Büyüklüğünün belirlenmesi için mevcutta yer alan tahmin veri tabanındaki veriler kullanılarak her bir kategori için iki farklı eşik değer belirlenmiştir. Her bir kategorinin eşik değerine göre giriş değerinin hangi büyüklük grubuna gireceği belirlenmektedir. Yapılan çalışmada verilerin az olmasından dolayı eşik değerleri sabit kabul edilmiştir. Çünkü bu çalışmadaki eşik değerinin yaptığı salınım oldukça küçüktür. Ancak ağın çıkış değerine göre yapılan hata hesabı göz önünde bulundurularak eşik değerlerinin belli bir hata değerine kadar hesaplanarak dinamik bir yapının kurulması ile daha iyi sonuçlar elde edileceği düşünülmektedir.

Parametre güncelleme işlemi farklı momentum katsayısıları ($\boldsymbol{\eta}$) ile test edilmiştir. En iyi sonucun momentum katsayısının 0.001 olduğunda elde edildiği gözlenmiştir. Yapılan çalışmada momentum katsayısı 0.001 olarak kullanılmaktadır.

5.4. Sonuçlar

Bahsedilen ağ yapısında 18 projeye ait gözlenmiş değerler ile parametre güncelleme işlemi yapılmıştır. Farklı iterasyonlar ile test edilmiştir ve en iyi sonuçların üçüncü iterasyonda elde edildiği gözlenmiştir. Üçüncü iterasyonda elde edilen değerler Tablo 7 da gösterilmektedir.

Tablo 7: Geri Yayma Yöntemi İle Elde Edilen Tahmin Veri Tabanı

Her bir senaryonun çalıştırma süresi dakika cinsinden			
Kategori	**Küçük (S)**	**Orta (M)**	**Büyük (L)**
Eşleşme (MK)	8,31	23,00	34,51
Birlikte Çalışabilirlik (BC)	1,73	17,20	34,96
Ekran Tasarımı (ET)	2,11	2,66	4,89
Ekran Fonksiyonları (EF)	2,27	3,43	7,65
Çıktıların Kontrolü (CK)	0,07	7,98	14,18
Uyarı ve Hata Mesajları (UHM)	2,58	8,34	18,75

18 proje için mevcuttaki tahmin veri tabanındaki değerler ile hesaplanan ideal şartlarda ki test yürütme süresi ile gözlenmiş değerler ile hesaplanan ideal şartlarda ki test yürütme süresi arasındaki dakika cinsinden ortalama bağıl hata % 14,36 iken 18 proje için geri yayılım algoritması ile iyileştirilen tahmin veri tabanındaki değerler ile hesaplanan ideal şartlarda ki test yürütme süresi ile gözlenmiş değerler ile hesaplanan ideal şartlarda ki test yürütme süresi arasındaki dakika cinsinden ortalama bağıl hata % 10,01 olarak gözlenmiştir (Bknz Tablo 8). Ortalama bağıl hatalar değerlendirildiğinde geri yayılım algoritmasının tahmin veri tabanının iyileştirdiği görülmektedir.

Tablo 8: İdeal Şartlarda Test Çalıştırma Süresi - Dakika Cinsinden Hatalar

Proje-No	İdeal Şartlarda Gözlenen Test Çalıştırma Süresi (dk)	İdeal Şartlarda Mevcut Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi (dk)	İdeal Şartlarda İyileştirilmiş Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi (dk)	İdeal Şartlarda Gözlenen Test Çalıştırma Süresi ile İdeal Şartlarda Mevcut Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi Arasındaki Bağıl Hata	İdeal Şartlarda Gözlenen Test Çalıştırma Süresi ile İdeal Şartlarda İyileştirilmiş Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi Arasındaki Bağıl Hata
1	616	434,01	475,12	29,54	22,87
2	252	193,29	237,64	23,30	5,70
3	85	72,16	80,14	15,11	5,72
4	732	657,51	693,95	10,18	5,20
5	1811	1846,98	2256,55	1,99	24,60
6	152	102,09	125,69	32,84	17,31
7	166	161,73	167,74	2,57	1,05
8	80	75,88	91,94	5,15	14,93
9	2171	2261,77	2824,86	4,18	30,12
10	990	781,28	1003,67	21,08	1,38
11	325	335,95	347,29	3,37	6,86
12	836	526,22	666,1	37,06	20,32
13	570	421,06	505,34	26,13	11,34
14	1310	1209,72	1357,43	7,65	3,62
15	1150	1116,99	1217,7	2,87	5,89
16	820	672,17	822,03	18,03	0,25
17	1335	1189,49	1311,76	10,90	1,74
18	685	639,72	676,38	6,61	1,26
Ortalama:				14,36	10,01

18 proje için gözlenen ideal şartlardaki test etme süresi kullanılarak hesaplanan toplam test yürütme süresi ile gerçek test yürütme süresi arasındaki ortalama bağıl hata %7,12; Mevcuttaki tahmin veri tabanı kullanılarak hesaplanan ideal şartlardaki test etme süresi kullanılarak hesaplanan toplam test yürütme süresi ile gerçek test yürütme süresi arasındaki ortalama bağıl hata %7,46; İyileştirilmiş tahmin veri tabanı kullanılarak hesaplanan ideal şartlardaki test etme süresi kullanılarak hesaplanan toplam test yürütme süresi ile gerçek test yürütme süresi arasındaki ortalama bağıl hata %7,19 olarak gözlenmiştir (Bknz Tablo 9).

Toplam test yürütme süresi için ortalama bağıl hatalar değerlendirildiğinde iyileştirilen tahmin veri tabanı kullanılarak daha iyi tahminler yapılacağı görülmektedir.

Tahmin veri tabanındaki değerler proje sayısının artması ile daha iyileştirilmesi hedeflenmektedir.

Tablo 9: Toplam Test Çalıştırma Süresi – Saat Cinsinden Hatalar

Proje-No	Gerçek Toplam Test Çalıştırma Süresi (saat) [T]	İdeal Şartlarda Gözlenen Test Çalıştırma Süresi Kullanılarak Hesaplanan Toplam Test Çalıştırma Süresi (saat) [G]	İdeal Şartlarda Mevcut Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi Kullanılarak Elde edilen Toplam Test Çalıştırma Süresi (saat) [M]	İdeal Şartlarda İyileştirilmiş Tahmin Veri Tabanı ile Hesaplanan Test Çalıştırma Süresi Kullanılarak Elde edilen Toplam Test Çalıştırma Süresi (saat) [I]	T ile G arasındaki bağıl hata	T ile M arasındaki bağıl hata	T ile I arasındaki bağıl hata
1	180	178,13	174,11	172,98	3,37	10,60	12,64
2	36	44,00	44,48	42,97	2,88	3,05	2,51
3	8	5,33	6,69	17,61	0,21	0,11	0,77
4	32	28,17	28,60	26,79	1,22	1,09	1,67
5	120	99,27	101,76	105,65	24,88	21,89	17,22
6	7	5,42	19,33	17,32	0,11	0,86	0,72
7	24	23,12	37,82	35,52	0,21	3,32	2,77
8	16	13,34	14,85	25,94	0,43	0,18	1,59
9	160	153,68	157,19	163,70	10,11	4,49	5,91
10	128	145,48	143,35	143,19	22,38	19,64	19,44
11	38	101,98	103,90	101,73	24,31	25,04	24,22
12	64	58,23	40,41	40,88	3,69	15,10	14,80
13	35	38,35	37,03	36,31	1,17	0,71	0,46
14	53	63,83	63,77	62,14	5,74	5,71	4,84
15	41	57,71	58,92	56,32	6,85	7,35	6,28
16	37	61,18	47,02	47,38	8,95	3,71	3,84
17	56	76,29	75,40	73,31	11,36	10,87	9,69
18	34	34,62	35,53	33,74	0,21	0,52	0,09
Ortalama:					7,12	7,46	7,19

BÖLÜM 6

PİLOT ÇALIŞMA

Geliştirilen modelin ölçülen veriler kullanılarak elde edilen sonuçlar ile çalışabilir olduğu önceki bölümde gösterilmiştir. Bu bölümde, modelin ölçülen değerler olmadan, Test Analistine atanan bir proje için ileri yönelik tahmin yapılarak çalışabilirliği değerlendirilecektir.

Pilot çalışmaya konu olan test projesi; <E-posta Verilerinin Raporlanabilir Ortama Taşınması Aktivitesi> testinin gerçekleştirilmesidir. Proje kapsamında; E-posta gönderimlerine ait verinin banka çalışanları tarafından raporlanabilmesi amacı ile, mevcut E-posta verileri raporlanabilir ortamda saklanmaya başlanacaktır. Bu amaç ile hem mevcut verilerin hem de güncel verilerin raporlanabilir ortama aktarımının doğru bir şekilde sağlandığı test edilecektir.

Testi yapılacak olan projenin gereksinimleri aşağıdaki gibi özetlenmektedir:

- E-posta gönderimlerinin raporlanabilir ortama taşınması için gerekli tablo yapısının ve veri taşıma yöntemlerinin geliştirilmesi gerekmektedir.
- E-posta gönderimlerine ait verilerin Mars (Bankacılık Veri Tabanında e-posta kayıtları ile müşteri ilişkisini sağlayan özgün bir değer) ile ilişkili olarak tutulması için Mars bilgisinin mevcut tablo yapılarına eklenmesi gerekmektedir.

Bu çerçevede bakıldığı zaman iki farklı sunucuda bulunan güncel verilerin tutulduğu 8 tabloya ve mevcut datanın tutulduğu 5 tabloya ait toplamda 116 veri sahasının doğru şekilde aktarılması beklenmektedir. Aktarımın sağlayacak olan 5 farklı iş çalıştırılacaktır. Bu sahalardan 2 tanesi (biri güncel veri için diğeri de mevcut veri için) Mars bilgisini tutmak için kullanılacak olup kontrolleri için farklı iş çalıştırma ve farklı tabloların kontrol edilmesi gereksinimi doğmaktadır.

Aktarımın testi ile ilgili olarak;

- Çalıştırılacak olan 5 işin doğru çalıştığının kontrol edilmesi için birlikte çalışabilirlik kontrolü yapılacaktır. Bu kontrollerin büyüklüğünün L olacağı öngörülmektedir: 5 * S(BC, L)

- 116 sahanın doğru beslendiğinin görülmesi için eşleşme kontrolü yapılacaktır, bu sahalardan 2 tanesinin kontrolü için farklı tablolarında kontrolüne de ihtiyaç duyulduğu için büyüklüğünün L olacağı öngörülmektedir, kalan sahalardan 44 tanesi için büyüklüğü M olacağı düşünülmektedir, geriye kalan 70 sahanın kontrolünün ise S büyüklüğünde olması beklenmektedir: 2 * S(MK, L), 44 * S(MK, M), 70 * S(MK, S)

Toplamda 121 kontrol yapılmakta olup buradan testin senaryo sayısının da 121 olabileceği tespit edilmiştir. Tahmin Veri Tabanı kullanılarak ideal şartlarda test çalıştırma süresi 28,5 saat olarak hesaplanmaktadır.

Projenin geliştiren yazılımcı Yazılımcı-7'dir. Yazılımcı-7'nin bulgu yoğunluğu ve bulgu düzeltme süresi kullanılarak, bulgu düzeltme süresi 13,9 saat olarak ve yeniden test etme süresi 54,53 dakika olarak hesaplanmaktadır.

Elde edilen değerlerin toplanması ile Test Analistinin bahsedilen projeyi 71,2 saatte yani günde 8 saat çalıştığı varsayılarak yaklaşık 9 iş gününde bitirmesi öngörülmektedir. Test Analistinin Bankacılık Veri Tabanına girmiş olduğu gerçekleşen aktivite süreleri incelendiğinde projeyi 84 saatte yani 10,5 iş gününde bitirmiş olduğu görülmüştür.

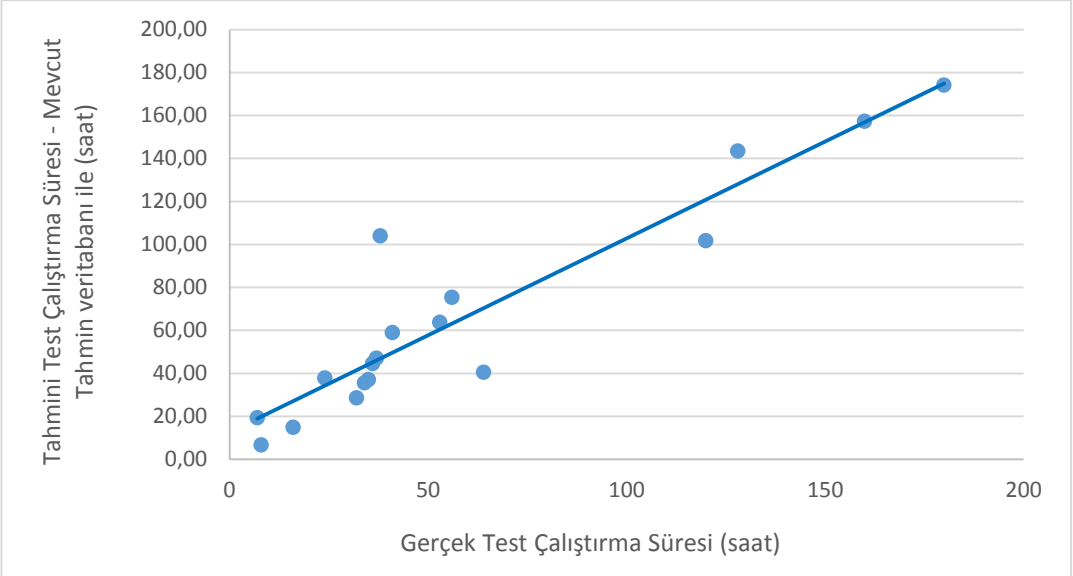
Pilot çalışmada test çalıştırma süresinin %84,85 doğrulukta tahmin edilebildiği gözlemlenmiştir. Her yeni projenin sağladığı veriler sayesinde gelişme yaşayan Tahmin Veri Tabanının sağlam bir temele oturması beklenmektedir. Dolayısı ile test çalıştırma süresi tahminindeki hatanın azalacağı düşünülmektedir.

BÖLÜM 7

SONUÇLAR

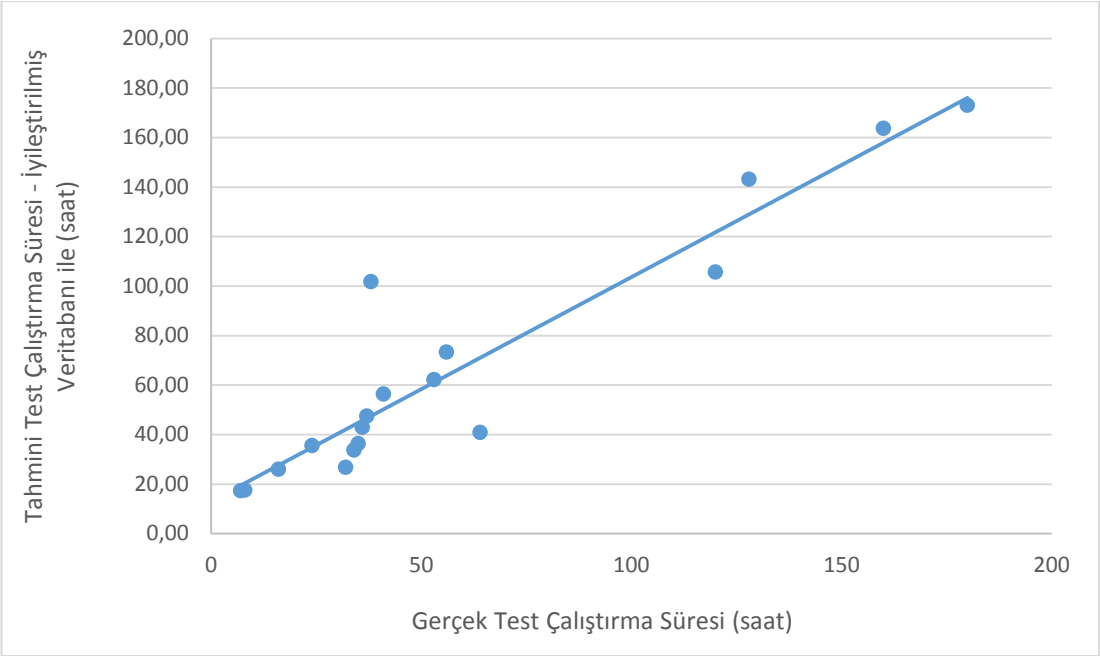
Pilot çalışması ile PROXY tabanlı metot kullanarak geçmişe yönelik veriler üzerinden yapılan test çalıştırma süresi tahmini için geliştirilen yöntemin işe yarar olduğu tespit edilmiştir. Tahmin ve gerçekleşme süresinin büyük oranda tutarlı olması (%84,85) geliştirilen yöntemin belirli bir alanda çalışan, atanan projelerin büyüklükleri birbirine yakın olan ve belirli bir yazılımcı kümesi ile çalışan bir Test Analisti için uygulanabilir ve gerçekçi olduğunu göstermektedir.

Pilot çalışmaya ek olarak, metottun geliştirilmesi sürecinde kullanılan 18 proje içinde gerçekleşme ve tahmin sürelerinin tutarlılığı değerlendirilmiştir. Mevcut veri tabanı kullanılarak hem de iyileştirilmiş veri tabanı kullanılarak elde edilen tahmin değerleri ile gözlenmiş (gerçek) değerler arasında doğrusal bir ilişki olduğu görülmüştür (Bknz Şekil 9).



Şekil – 9 Mevcut Veri tabanı Kullanılarak Elde Edilen Tahmini Test Çalıştırma Süresi ile Gerçek Test Çalıştırma Süreleri

Mevcut veri tabanındaki değerler ilk değerler olarak atanarak hata geri yayma yöntemi ile tahmin veri tabanı iyileştirilmiştir. İyileştirilmiş veri tabanı kullanılarak elde edilen tahmin değerleri ile gözlenmiş değerler arasında da doğrusal bir ilişki olduğu görülmüştür.



Şekil – 10 İyileştirilmiş Veri tabanı Kullanılarak Elde Edilen Tahmini Test Çalıştırma Süresi ile Gerçek Test Çalıştırma Süreleri

Mevcut tahmin veri tabanında ve iyileştirilmiş tahmin veri tabanında yer alan değerlerin kati değerler olmadığı ve bu değerlerin verisi ölçülen her bir yeni bir proje ile değişebilecek dinamik bir yapıya sahip olduğu görülmektedir. Söz konusu dinamik yapı test analistinin kişisel gelişim sürecinin de (deneyim süresinin artması, bilgi birikiminin artması vb.) tahminlere etkisini yansıtabileceği düşünülmektedir. Dolaylı olarak geliştirilen metodun, test analistinin kişisel gelişiminin etkilerini içermesi sağlanmaktadır. Çünkü test analistinin deneyimi ile birlikte senaryoları çalıştırma süresinin değişim gösterebileceği beklenmektedir.

Aynı şekilde yazılımcıların bulguya dönüş süreleri ve bulgu yoğunlukları kati değerler değildir. Bu değerler verisi ölçülen her yeni bir proje ile değişebilecek bir dinamik bir yapıya sahiptir. Bu dinamik yapı sayesinde yazılımcıların kişisel gelişim süreçlerinin (deneyim süresinin artması, bilgi birikiminin artması vb.) yapılan tahmine etkisini geliştirilen metodun dolaylı olarak içerdiği görülmektedir. Çünkü yazılımcıların kişisel gelişim süreçleri bulguya dönüş sürelerini ve bulgu yoğunluklarını etkileyebileceği beklenmektedir.

Kayıplar olarak bahsedilen bilgi alış veriş, bulgu raporlama ve mailleşme… gibi sürelerin de değişen diğer sürelere bağlı olarak değişim göstermesi beklenmektedir.

Kayıplar aslında geliştirilen metot için kontrol edilemeyen değişken görevi de görmektedir. Bu değer sayesinde hatanın yumuşatılması sağlanmaktadır.

Pilot çalışma ve pilot çalışmaya ek olarak geçmişe yönelik projeler için yapılan analiz göz önünde bulundurulduğunda Test Analistinin kişisel süreci için model üzerinden yapacağı tahminlerin gerçek sonuçlar ile örtüşmesi beklenmektedir.

Ancak bankacılık sektöründe zaman önemli bir kısıt olduğu için Test Analistlerinin kendi süreçlerini izlemeleri ve verilerini kayıt altına almaları beklenememektedir. Bundan dolayı tahmin veri tabanının oluşturulması zorlaşmaktadır. Dolayısıyla bildiride anlatılmakta olan modelin kullanılması için farklı alanda çalışan (mobil, özel projeler, nakit yönetimi, krediler vb.) birçok Test Analisti denek olarak belirlenmeli ve akabinde bu Test Analistlerinin kendi test süreçlerini izlemeleri ve ölçümlemeleri sağlanmalıdır. Böylelikle genel bir tahmin veri tabanı elde edilebilecektir.

Çalışmanın devamında ise elde edilen bu veri tabanına dayalı olarak geliştirilen metodun banka genelinde kullanılması için genelleştirilmesi ve rahat kullanımını sağlamak amacı ile web tabanlı bir araç geliştirilmesi planlanmaktadır.

KAYNAKÇA

[1] A. Chan, J. J. Wei, J. Boyer, P. Guo, H. Y. Wang, and H. L. Liu, "IBM Business Process Manager testing methodology , Part 1 : General testing guidelines," 2014.

[2] A. Kaldıroğlu, "Yazılım Projelerinde İhtiyaç Analizi – II," Kişisel Web Sitesi. [Online]. Available: http://www.javaturk.org/yazilim-projelerinde-ihtiyac-analizi-ii/. [Accessed: 18-Jun-2017].

[3] A. M. J. Hass, Guide to Advanced Software Testing. Artech House, 2008.

[4] B. Widrow and M. A. Lehr "Backpropagation and its Applications" Stanford University Department of Electirical Engineering, Stanford, CA 94305-4055

[5] B.W. Boehm "Software Engineering Economics" TRW Defense Systems Group, Redondo Beach, CA 90278, 1983

[6] C. Abhishek, V. P. Kumar, H. Vitta and P. R. Srivastava "Test Effort Estimation Using Neural Network" J. Software Engineering & Applications, 2010, 3: 331-340

[7] D. E. Rumelhart, G. E. Hinton, and R.J. Williams. "Learning internal representations by error propagation In D.E.Rumelhart and J.L. McClelland, editors, Parallel Distributed Processing, volume 1, chapter 8. The MIT Press, Cambridge, MA, 1986.

[8] D. Firesmith, "SEI Blog: Using V Models for Testing." Carnegei Mellon University,2013

[9] E. Aranha and P. Borba, "Estimating manual test execution effort and capacity based on execution points," Int. J. Comput. Appl., vol. 31, no. 3, pp. 167–172, 2009.

[10] E. Güler, "PSP ile Süreç Ölçümü." Kişisel Web Sayfası, p. 2017.

[11] H. Mai, "IT in Banks : What does it cost ?" , Deutsche Bank DB Research, 2012

[12] IEEE, "IEEE Std. 829-2008: Standard for Software and System Test Documentation," 2008

[13] ISO/IEC, "International Standard ISO/IEC 9126-1 Software engineering - Product quality - Part 1: Quality model," 2001.

[14] ISTQB, "Sertifikalı Test Uzmanı İleri Seviye Ders Programı Test Analisti ver.2012," 2012.

[15] ISTQB, "Sertifikalı Test Uzmanı Temel Seviye Ders Programı ver.2011," 2011.

[16] ISTQB, "Yazılım Testi Terimler Sözlüğü ver1.0," 2014.

[17] K. R. Jayakumar and A. Abran, "A Survey of Software Test Estimation Techniques," J. Softw. Eng. Appl., vol. 6, no. October, pp. 47–52, 2013.

[18] M. Chemuturi , Software Estimation Best Practices, Tools, & Techniques A Complete Guide for Software Project Estimators, ISBN: 978-1-60427-024-2, Florida, 2009.

[19] M. Chemuturi "Test Effort Estimation" Available: http://chemuturi.com/Test%20Effort%20Estimation.pdf [Accessed: 18-Jun-2017]

[20] M. Efe and O. Kaynak, "Yapay Sinir Ağları ve Uygulamaları",Boğaziçi Üniversitesi Yayınları, 2004

[21] M. Kerstner, "Software test effort estimation," SIGSOFT Softw. Eng. Notes. 2011.

[22] M. Pomeroy-Huff, R. Cannon, T. A. Chick, J. Mullaney and W. Nichols "The Personal Software Process Body of Knowledge, Version 2.0" Special Report CMU/SEI-2009-018, 2009

[23] Maurizio Bolognini (2001),Democrazia elettronica. Metodo Delphi e politiche pubbliche (Electronic Democracy. Delphi Method and Public Policy-Making) (in Italian), Rome: Carocci Editore, ISBN 88-430-2035-8. A summary is also in Jerome C. Glenn, Theodore J. Gordon (eds) (2009), The Millennium Project. Futures Research Methodology, New York: Amer Council for the United Nations, ISBN 978-0981894119, chap. 23.

[24] Murray Turoff, Starr Roxanne Hiltz, "Computer-based Delphi processes", in Michael Adler, Erio Ziglio (eds.), Gazing Into the Oracle, op. cit.

[25] N. Tiftik, H. Öztarak, G. Ercek, ve S. Özgün "Sistem/Yazılım Geliştirme Sürecinde Doğrulama Faaliyetleri", Ankara

[26] R. E . Park "Software Size Measurement: A Framework for Counting Source Statements" Technical Report CMU/SEI-92-TR-020 ESC-TR-92-020

[27] S. Akagündüz, S. Kurnaz, and M. Sari, "Yazılım Proje Yönetiminde Proje Başarısını Getiren Faktörler (Factors That Make The Success Of The Project In Software Project Management)," in Akademik Bilişim 2013, 2013.

[28] S. Çalışkan, E. Çetinkaya, K. Dinçer, E. Yılmaz, and H. Çakıcı, PSP Eğitimi için Kullanıcı Dostu bir Süreç Yönetim Aracı Geliştirme Denemesi (A Process Management Tool Development Trial for PSP Training)," in Proceedings of the 9th Turkish National Software Engineering Symposium (UYMS 2015), 2015, pp. 529–541.

[29] S. Nageswaran, "Test Effort Estimation Using Use Case Points," Quality Week 2001, San Francisco, pp. 1–6, 2001.

[30] SWEBOK, Guide to the Software Engineering Body of Knowledge, 2004 Version

[31] T. Veenendaal, van EPWM (Erik); Dekkers, "Test Point Analysis : a Method for Test Estimation," Proj. Control Softw. Qual., pp. 1–16, 1999.

[32] V. Ozdamar, “Yazılım Testi ve Kalite Neden Önemlidir?,” Kişisel Web Sayfası, 2017. [Online]. Available: http://volkanozdamar.com/yazilim-testi-ve-kalite-neden-onemlidir/. [Accessed: 18-Jun-2017].

[33] W. Humphrey, “A Discipline for Software Engineering,” Addison Wesley, Boston, 1995.

ÖZGEÇMİŞ

Kimlik Bilgileri

Ad, Soyad: Gizem Kahveci

Doğum Yeri: Ankara

E-mail: gizemkhvc@gmail.com

Adres: Hacettepe Üniversitesi, Beytepe Kampüsü, Bilgisayar Mühendisliği Bölümü, Ankara

Eğitim

Lisans: İstanbul Teknik Üniversitesi, Meteoroloji Mühendisliği, 2011

Lise: İncesu Anadolu Lisesi, Çankaya, Ankara, 2006

Yabancı Diller

İngilizce

İş Deneyimi

2013-2016: Devlet Su İşleri, Meteoroloji Mühendisi

2016-Halen: Garanti Teknoloji, Analist

Deneyim Alanları

Yazılım Mühendisliği, Yapay Sinir Ağları

Tezden Üretilmiş Projeler ve Bütçesi

--

Tezden Üretilmiş Yayınlar

G. Kahveci, K. Dinçer “PTP-Vision: Bankacılık Uygulamaları için Proxy Tabanlı Kişisel Yazılım Test Eforu Tahmin Metodu” 11. Ulusal Yazılım Mühendisliği Sempozyumu, 18-20 Ekim 2017

G. Kahveci, K. Dinçer “Estimating Testing Times of Individual Testers – A Proxy Based Method” International Conference On Recent Advances in Computer Science and Information Technology” (ICRACSIT) 23-24 th November, 2017

Tezden Üretilmiş Tebliğ ile Katıldığı Toplantılar

--

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS/DOKTORA TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 03/01/2018

Tez Başlığı / Konusu: Bankacılık Alanında Kişisel Yardım Test Eforunu Tahmin Etmek için Proxy Tabanlı Bir Metot ve Vaka Çalışması

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler ve d) Sonuç kısımlarından oluşan toplam 64+1. sayfalık kısmına ilişkin, 5/1/2018 tarihinde şahsım/tez danışmanım tarafından turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 6 'tür.

Uygulanan filtrelemeler:

1- Kaynakça hariç
2- Alıntılar ~~hariç~~/dâhil
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü Tez Çalışması Orjinallik Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

03.01.2018.
Tarih ve İmza

Adı Soyadı: Gizem KAHVECİ
Öğrenci No: N14329302
Anabilim Dalı: Bilgisayar Mühendisliği
Programı:
Statüsü: ☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Prof. Dr. M. Ender Efe
(Unvan, Ad Soyad, İmza)