COSMIC İŞLEVSEL BÜYÜKLÜĞÜN JAVA İŞ UYGULAMALARINA ÖLÇME KODU ENSTRÜMANTASYONU YOLUYLA ÇALIŞMA ZAMANINDA ÖLÇÜLMESİ

RUN-TIME MEASURING OF COSMIC FUNCTIONAL SIZE VIA MEASUREMENT CODE INSTRUMENTATION INTO JAVA BUSINESS APPLICATIONS

RANA GÖNÜLTAŞ

YRD. DOÇ. DR. AYÇA TARHAN
Tez Danışmanı

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

2015

RANA GÖNÜLTAŞ'ın hazırladığı "**COSMIC İşlevsel Büyüklüğün Java İş Uygulamalarına Ölçme Kodu Enstrümantasyonu Yoluyla Çalışma Zamanında Ölçülmesi**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Murat Caner TESTİK
Başkan ……

Yrd. Doç. Dr. Ayça TARHAN
Danışman ……

Doç. Dr. Vahid GAROUSİ
Üye ……

Doç. Dr. Pınar KARAGÖZ
Üye ……

Yrd. Doç. Dr. Kıvanç DİNÇER
Üye ……

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

Prof. Dr. Fatma SEVİN DÜZ
Fen Bilimleri Enstitüsü Müdürü

Aileme...

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 sonuçları bilimsel ahlak kurallarına 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 üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

09/06/2015

RANA GÖNÜLTAŞ

ÖZET

COSMIC İŞLEVSEL BÜYÜKLÜĞÜN JAVA İŞ UYGULAMALARINA ÖLÇME KODU ENSTRÜMANTASYONU YOLUYLA ÇALIŞMA ZAMANINDA ÖLÇÜLMESİ

Rana GÖNÜLTAŞ
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Danışmanı: Yrd. Doç. Dr. Ayça TARHAN
Haziran 2015, 85 sayfa

Hızla gelişen bilgi teknolojileri dünyasında yazılımların işlevsel büyüklüklerinin ölçülmesi, yazılım projelerinin yönetiminde düşünülmesi gereken önemli bir konudur. İşlevsel büyüklük ölçümü, yazılım projelerinin süresi boyunca planlama parametrelerini tahmin etme ve süreci takip etmek açısından sağlam bir zemin hazırlar. Ancak manuel olarak yapılan ölçümler maliyeti artırmakta ve zaman kaybına yol açabilmektedir. Ayrıca yazılım projelerinde projenin başından sonuna kadar bu ölçümün sağlanabilmesi ve doğru bir şekilde ölçülebilmesi karmaşık uygulamalar için bazen zorlaşmakta ve yapılan ölçümler kişiden kişiye farklılık gösterebilmektedir. Bu sebeplerden ötürü ölçümün otomatikleştirilmesi fikri önem kazanmaktadır.

Bu tez kapsamında, üç katmanlı mimariye sahip Java web uygulamalarının COSMIC işlevsel büyüklüğünün otomatik olarak hesaplanması amaçlanmıştır. Bu hedef doğrultusunda, geliştirilen kütüphanenin ilgili metotları otomatik olarak uygulamanın kaynak kodu içine yerleştirilmiştir. Daha sonra çalışma zamanında kullanıcı senaryolarına bağlı olarak büyüklük hesaplanması işlemi gerçekleştirilmiştir. Önerdiğimiz bu yöntem, ülkemizdeki bir kamu kurumu tarafından aktif olarak kullanılan bir sistemde işlevsel büyüklüğü ölçmek için test

edilmiştir. Otomatik ölçme sonuçlarının doğruluğunun karşılaştırılabilmesi amacıyla aynı uygulama için ayrıca manuel ölçüm de gerçekleştirilmiştir. Otomatik ölçülen ve manuel olarak hesaplanan COSMIC işlevsel büyüklüklerinin %96 oranında yakınsadığı görülmüştür. Ayrıca, otomatik olarak yapılan ölçüm süresi manuel olarak yapılan ölçüm süresinin 1/27 si olarak yaklaşık 10 dakika sürmüştür.

Geliştirilen modelin, yazılımların işlevsel büyüklüklerinin otomatik olarak ölçülmesi konusunda izlediği yöntem bakımından literatürde ender bulunan çalışmalardan biri olması sebebiyle, bu alanda yapılacak diğer çalışmalar için örnek teşkil etmesi beklenmektedir. Gelecek çalışmalarda, önerilen model geliştirilerek daha geniş kapsamda farklı sistemler için de kullanılabilir.

Anahtar Kelimeler: COSMIC, işlevsel büyüklük, otomatik hesaplama, ölçüm kütüphanesi, kod ekleme, çalışma zamanı hesaplama, Java iş uygulamaları.

ABSTRACT

RUN-TIME MEASURING OF COSMIC FUNCTIONAL SIZE VIA MEASUREMENT CODE INSTRUMENTATION INTO JAVA BUSINESS APPLICATIONS

Rana GÖNÜLTAŞ
Master of Science, Department of Computer Engineering
Supervisior: Asst. Prof. Dr. Ayça TARHAN
June 2015, 85 pages

With the rapid development of information technologies in world, measuring functional size of software is an important issue must be considered in management of software projects. Functional size measurement provides a solid ground throughout software projects to estimate planning parameters and track progress. But, when functional size measurement is made manually, it is time-consuming and costly. Moreover, providing measurement from early development to end and measuring accurately is become difficult for complicated projects and measurement results may differ from person to person. For these reasons, automating the process of measurement has gained importance.

In this study, it is aimed that the measurement of COSMIC functional size of three-tier Java business applications automatically. In this purpose, measurement is done at run-time according to user scenarios with installing created Measurement Library methods to application source code automatically. Proposed procedure is tested for measuring functional size of system, which is used active by one of the government-based organization in our country. To compare automatic measurement result accuracy, manual measurement is also made for application. We report that functional sizes measured manually and automatically were %96

convergent and that automatic measurement took 10 minutes which was 1/27 of manual measurement effort.
Because of the fact that the developed model is the rare study in its field in terms of the method used for the functional size measurement of software automatically, it is expected that it may provide a basis for further studies. In future studies, proposed model may be extended and can use for different type of system in more extensive scope.

Keywords: COSMIC, functional size, automatic measurement, measurement library, code installment, run-time measurement, Java business applications.

TEŞEKKÜR

Tez çalışmasının gerçekleştirilmesi esnasında ilgi ve desteğini esirgemeyen, her türlü değerli bilgileri ile hep yanımda olan, çalışmalarımı her zaman destekleyerek ilerlememi sağlayan danışmanım Sayın Yrd. Doç. Dr. Ayça TARHAN'a sonsuz teşekkürlerimi sunarım.

İyi ve kötü günlerimde hep yanımda olan ve hiçbir desteğini esirgemeyen bütün dost ve arkadaşlarıma en içten dileklerimle teşekkür ederim.

TÜBİTAK – BİLGEM – Yazılım Teknolojileri Araştırma Enstitüsü'ne (YTE), akademik çalışmalarım için sağladıkları veriler ve yardımları için teşekkürlerimi bir borç bilirim.

Ayrıca sevgi ve desteklerini hiç esirgemeyen, zor anlarımda hep yanımda olan, bana her zaman güvenen ve beni bugünlere getiren canım aileme sonsuz teşekkürler.

İÇİNDEKİLER

Sayfa

ÇİZELGELER

ŞEKİLLER

SİMGELER VE KISALTMALAR

Kısaltmalar

ACWP	The Actual Cost of Work Performed
AS	Araştırma Sorusu
BAC	Budget at Completion
BCWP	The Budgeted Cost of Work Performed
BCWS	The Budgeted Cost of Work Scheduled
CFP	COSMIC Function Point
CFSU	COSMIC Functional Size Unit
CMMI	Capability Maturity Model Integration
COCOMO	The Constructive Cost Model
COSMIC	Common Software Measurement International Consortium
CRUDL	Create/Read/Update/Delete/List
DAO	Data Access Object
DB	Database
DI	Degrees of Influence
E	Entry
ECF	The Environment Complexity Factor
EDF	Event Driven Framework
EI	External Inputs
EO	External Outputs
EQ	Early and Quick
EV	Earned Value
EVM	Earned Value Management
FFP	Full Function Points
FPA	Function Point Analysis
FSM	Functional Size Measurement
FUR	Functional User Requirements
GS	Genel Süreç
GUI	Graphical User Interface
HQL	Hibernate Query Language

HTTP	Hyper-Text Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IFPUG FPA	International Function Point Users Group Function Point Analysis
ISO	International Organization for Standardization
IT	Information Technology
İKG	İşlevsel Kullanıcı Gereksinimi
İS	İşlevsel Süreç
JCP	Java Community Process
JSF	Java Server Faces
KD	Kullanım Durumu
LoC	Lines of Code
MTTF	Mean Time to Failure
MVC	Model View Controller
NESMA	Netherlands Software Metrics Association
NFR	Non-Functional Requirements
ORM	Object Relational Mapping
PF	Productivity Factor
PSU	Project Size Unit
PV	Planned Value
R	Read
ROCOF	Rate of Occurrence of Failure
RRRT	Rational Rose RealTime
SEER-SEM	SEER for Software
SLIM	Software Life Cycle Management
SLOC	Source Lines of Code
TCF	The Technical Complexity Factor
UCP	Use Case Points
UFP	Unadjusted Function Points
UUCP	Unadjusted Use Case Points
UML	Unified Modeling Language
VT	Veri Tabanı
W	Write

X	Exit
XML	Extensible Markup Language

1. GİRİŞ

Ürün büyüklüğünün hesaplanması işlemi, yazılım geliştirme başlangıç aşamasından bakım sürecine kadar geçen süre zarfında riskleri yönetme, kapsam, program ve maliyet gibi kritik konuların farklı bir bakış açısıyla ele alınıp değerlendirilmesine olanak sağlar. Literatürde, yazılım geliştirmede ölçümün başlıca amaçları araştırıldığında yaygın olarak söz edilen amaçlar aşağıdaki gibi sıralanabilir:

- Kalite güvencesi (Kod kalitesi, hatalar, uygulama performansı)
- Performans ölçümü (kişilerin, ekibin ya da organizasyonun ne ölçüde başarılı olduğu)
- Süreç iyileştirmesidir. (Değişikliklere daha hızlı adapte olabilmek)

Yukarda belirtilen başlıca sebeplere ek olarak dolaylı yoldan ölçümün öneminin anlaşılmasını sağlayan bir gösterge, yazılım projelerinin başarı oranıdır. Standish Group'un CHAOS raporuna göre [1], her yıl Amerika'da Bilgi Teknolojileri ("Information Technology – IT") projelerinde yaklaşık 175.000 proje için 250 milyar dolardan fazla para harcanmaktadır. Harcanan yüksek maliyete karşılık başarı ne yazık ki istenilen düzeyde değildir. Şekil 1.1 1994-2012 yılları arasında projelerin başarı yüzdesini göstermektedir.

Şekil 1.1. 1994-2012 Yılları Arasındaki Proje Başarılarını Gösteren CHAOS Araştırma Sonuçları

Grafiğe bakılarak başarı oranının yıl geçtikçe artsa da, oranın istenilen düzeyde olmadığı anlaşılmaktadır. 2012 yılında başarıyla tamamlanan yazılım projelerinin yüzdesi %39, başarısızlıkla sonuçlanan proje yüzdesi %18, tamamlanan ancak bütçe üzerinde, gecikmeli ve istenilenden daha az özelliğe sahip bir şekilde teslim edilen projelerin oranı ise %43 tür. Başarının istenilen ölçüde olmaması proje başarısızlığına yol açan faktörlerin araştırılmasına sebebiyet verirken, ortaya çıkan sonuçlar genel olarak kaynak, kapsam ve tahminlemelerin yanlış yapılması üzerinde yoğunlaşmıştır. Buna göre, tahminlemeyi doğru yapıp diğer konuları da süreçler doğrultusunda iyi yönettikçe başarı oranı artacaktır.

Yazılım projelerinde beş temel ölçüt bulunmaktadır. Emek proje kapsamında ayda kaç kişi çalışacağı bilgisini, maliyet projenin ne kadara mal olacağını, zaman projenin kaç ayda tamamlanacağını, kalite ise tespit edilen hataları ifade etmektedir. Büyüklük ise genel olarak kod satır sayısı kavramıyla benzeşmiştir. Ancak, farklı programlama dilleri ve farklı teknolojiler kullanarak geliştirilmiş uygulamalar için kod satır sayısından büyüklük hesaplanması sağlıklı değerlendirme yapılmasını tam anlamıyla sağlamamaktadır.

Yazılım projelerinde zaman planlama, süreç yönetimi ve büyüklük hesaplama (kod satır sayısından yola çıkmayarak farklı biçimde) gibi konuların tahmin bazlı yapılabilmesi için belli başlı ölçümlere ihtiyaç duyulmaktadır. İşte tam bu noktada işlevsel büyüklük ölçümü kavramı önem kazanmaktadır.

1.1. İşlevsel Büyüklük Ölçme

İşlevsel Büyüklük Ölçümü kavramı, yazılımın kullanıcılarına sunduğu işlevselliğI ifade eder. Amaç; yazılımları, kullanılan dil ve platformdan bağımsız olarak yalnızca sağladıkları işlevsellik üzerinden ölçmektir. Allan Albrecht tarafından 1979 yılında ortaya atılan bu fikir [2], zamanla gelişerek ilerlemeye devam etmektedir.

İşlevsel büyüklük yöntemiyle büyüklük hesaplamanın önemi birçok çalışmaya konu olmuş ve literatürde geniş yer kaplamaktadır. Örneğin, işlevsel büyüklük ölçümü, işlevsel büyüklük üzerinden kazanılan değerin (“Earned Value”) kullanılması ile yazılım geliştirme sürecinin takibinde faydalı olabilir [3] [4] [5] [6]. Ayrıca, işlevsel büyüklüğün tekrarlamalı (“Iterative”) geliştirmede [7] [8] ya da çevik (“Agile”) geliştirmede [9] [10] kullanılmasının faydalı olduğu literatürdeki çalışmalarda anlatılmaktadır.

Mevcut durumda birçok işlevsel büyüklük ölçme metodu bulunmaktadır. Bunlardan Mk II Function Point Analysis (FPA) [11], Netherlands Software Metrics Users Association (NESMA) Functional Size Measurement (FSM) [12], International Function Point Users Group (IFPUG) FPA [13] ve Common Software Measurement International Consortium (COSMIC) Measurement Method [14] en yaygın olarak kullanılan metotlardır.

Bu tez çalışması kapsamında, yukarıda belirtilen işlevsel büyüklük ölçme metotlarından en popüler olan COSMIC ölçme metotu seçilmiş ve bu kapsamda kullanılmıştır. COSMIC işlevsel büyüklük ölçümünün, iş uygulamaları ve gerçek zamanlı yazılım sistemlerinin özelliklerinden ve işlevsel kullanıcı gereksinimlerinden yola çıkarak işlevsel büyüklüklerini ölçmede geçerli bir metot olduğu kanıtlanmıştır [15]. Metotun kullanılmasının esas avantajı, kullanıcıların gözünden yazılımın ölçülme olanağı ve herhangi bir teknik özellik ve kriterden bağımsız olmasıdır [16].

COSMIC işlevsel büyüklük ölçme yönteminin efor bağımlı olması ve nasıl ölçüldüğünün iyi düzeyde öğrenilmesi gerektiği sebeplerinden, ölçüm manuel olarak yapıldığında maliyetli olmaktadır. Bu sebeplerden ötürü ölçümün otomatikleştirilmesi maliyeti azaltabilmektedir [17] [18]. Analiz ve tasarım modelleri gibi erken yazılım geliştirme aşamaları [19] [20] [21] [22] ile kaynak koddan yola çıkarak [17] [23] [24] COSMIC işlevsel büyüklüğün otomatik olarak hesaplanması ile ilgili literatürde birçok çalışma yer almaktadır.

Bu çalışmanın amacı, Java iş uygulamaları için COSMIC işlevsel büyüklük ölçme işleminin otomatikleştirilmesidir. Bu amaçla yapılan çalışmaların ayrıntısına izleyen bölümde değinilmiştir.

1.2. Tez Çalışmasının Hedefi ve Kapsamı

Tez çalışması kapsamında üç katmanlı mimariye sahip ve grafiksel kullanıcı arayüzü (“Graphical User Interface - GUI”) içeren web tabanlı Java iş uygulamaları için COSMIC işlevsel büyüklüğün otomatik olarak hesaplanması amaçlanmıştır. Ölçümü yapılan uygulamanın gerçeklemesi, 82 kişilik yazılım geliştirici ekibi ile CMMI Seviye 4 [25] sertifikasına sahip ve genellikle kamu kurumları için bilgi sistemi geliştiren bir kurumda yapılmıştır ve örnek uygulama mevcut durumda bakım sürecindedir.

Otomasyon işlemi için ilk olarak, COSMIC ölçüm kuralları [15] doğrultusunda Measurement Kütüphanesi geliştirilmiştir. Daha sonra, geliştirilen kütüphanenin metotları ayrıştırma işlemi ile ölçümü hedeflenen Java uygulamasının kaynak kodu içerisine otomatik olarak koyulmuştur. Uygulama kodundaki ilgili metotlar ilişkili oldukları COSMIC veri hareketi tiplerine (Giriş/Okuma/Yazma/Çıkış) göre ayrıştırılmış ve bu metotlara, içerdikleri türlere göre kütüphane metotları eklenmiştir. Uygulamanın işlevsel büyüklüğü, Java uygulamasının içerisine otomatik olarak yerleştirilen metot çağrılarından sonra, çalışma zamanında kullanıcı senaryoları doğrultusunda kullanıcı arayüzü üzerinden tetiklenen işlevsel süreçler ile hesaplanmaktadır.

Çalışmada özetle aşağıdaki adımlar izlenmiştir;

Araştırma Evresi:

- ISO kapsamında kabul görmüş ve yaygın olarak kullanılan işlevsel büyüklük ölçüm metotlarının incelenmesi, benzerlikleri ve farklılıkları ile avantajlarının araştırılması,
- Tez çalışması kapsamında kullanılan bir işlevsel büyüklük ölçümü metotu olan COSMIC metodunun detaylı bir şekilde araştırılarak, ölçme stratejisinin öğrenilmesi,
- COSMIC ölçme metodu kullanılarak manuel olarak hesaplama yapan çalışmaların incelenmesi,
- Literatürde yer alan COSMIC ölçme metodu kullanarak otomatik olarak hesaplama yapan çalışmaların incelenmesi,
- Öngördüğümüz kütüphane ve ayrıştırma işleminin uygulanabilirliğinin değerlendirilmesi,
- COSMIC metodunun ilkelerinin gözden geçirilerek çalışmamıza özel kapsamın seçilmesi ve gerçekleştirme için sınırların çizilmesi.

Gerçekleme Evresi:

- COSMIC metodunu baz alarak ölçme işleminin otomatik olarak yapılması için Measurement Kütüphanesi'nin geliştirilmesi,

- Geliştirilen ölçüm kütüphanesi metotlarını, ölçümü yapılacak uygulamanın kaynak kodlarına otomatik olarak eklemek için ayrıştırma işlemi kodunun geliştirilmesi,
- Uygulamanın seçilen modülündeki belli başlı kullanım durumlarının çalışma zamanında işlevsel büyüklüklerinin geliştirilen model ile ölçülmesi,

Geçerleme Evresi:

- Geliştirilen modelin doğruluğunu test etmek amacıyla, ölçümün manuel olarak yapılması ve otomatik yapılan ölçümle karşılaştırılması,
- Manuel ve otomatik yöntemin hesaplama sürelerinin kaydedilerek karşılaştırılması,
- İşlevsel büyüklükleri hesaplanan kullanım durumlarının, kurumda mevcut durumda kullanılmakta olan COSMIC Tam İşlev Puanı (“Full Function Points - FFP”) biriminde, Erken ve Hızlı (“Early and Quick – EQ”) Tahminleme yönteminin kuruma özgü olarak uyarlanmış haliyle gerçekleştirilen otomatik ölçümle karşılaştırılması,
- Elde edilen sonuçları değerlendirerek önerilen modelin uygulanabilirliğinin tartışılması.

Çalışmanın ilerleyen kısımları şu şekilde organize edilmiştir. 2. Bölümde, gerçekleştirilen çalışmanın amacı belirtilerek yazılım mühendisliği alanında ölçümün amacı, yöntemleri ve metotları hakkında detaylı teknik bilgiler sunulmuştur. 3. Bölümde tez kapsamında seçilen COSMIC işlevsel büyüklük ölçümü detaylı olarak anlatılmıştır. 4. Bölümde literatürde yer alan ve bu tez kapsamında gerçekleştirilen çalışma ile benzer çalışma gerçekleştiren diğer çalışmalardan bahsedilmiş ve bu çalışmalar ile benzer ve farklı yönler anlatılmıştır. 5. Bölümde önerilen otomatik ölçme yönteminden bahsedilmiştir. 6. Bölümde önerilen modelin test edildiği örnek uygulama ile ölçme sonuçları paylaşılmıştır. 7. Bölümde işlevsel büyüklük ölçme yöntemlerinin farkındalığını anlamak amacıyla yapılmış anket uygulaması ve sonuçları belirtilmiştir. 8. ve son bölümde ise çalışmaya ait son görüşler paylaşılmış ve ileride gerçekleştirilebilecek çalışmalar hakkında önerilerde bulunulmuştur.

2. ÖN BİLGİ

2.1. Ölçme Kavramı

Ölçme, gerçek dünyadaki varlıkların özelliklerini tarif etmek için belirlenmiş kurallar çerçevesinde sayı veya sembollerin bu varlıklara ait özelliklere atanması işlemidir [26]. Yani ölçme, varlıkların nitelikleri hakkında bilgi verir. Gerçek dünyada varlık, bir obje (insan ya da oda gibi) ya da bir olaydır (seyahat ya da yazılım projesinin test aşaması gibi).

Ölçme hayatımızın her yerindedir. Alışverişte, ürünlerin değerinin ölçüsü olan fiyat sayesinde hesaplama yaparız. Kıyafetin bedenimize uygun gelmesi için genişlik ve beden ölçülerinden yararlanırız. Seyahatte rotamızı belirlemek, hızımızı ölçmek ve gideceğimiz yeri tahmin etmek için mesafeyi ölçeriz. Ölçme sayesinde, hayatı daha iyi anlar, çevremizdeki şeylerle etkileşimimizi sağlar ve yaşantımızı iyileştiririz [26].

Ünlü bilim adamı Galileo Galilei (1564-1642) “Ölçülebilen şeyleri ölçün, ölçülemeyecek şeyleri ölçülebilir yapın” demiştir. Buradaki üstü kapalı mesaj, ölçmenin kavramları daha görünür yapması ve böylece kavramların daha anlaşılır ve kontrol edilebilir olmasıdır.

Fizik bilimlerinde, tıpta, ekonomide ve hatta bazı sosyal bilimlerde önceden ölçülemeyecek şekilde düşünülen nitelikler artık ölçülmeye başlanmıştır. İnsan zekası, hava kalitesi, enflasyon endeksi gibi bazı ölçümler artık günlük yaşantımızı etkileyen önemli kararlarda temel oluşturmaktadır [26].

2.2. Yazılım Mühendisliği’nde Ölçme

Günlük yaşantımızda ölçme ne kadar önemli ise, yazılım mühendisliğinde de ölçmenin önemi büyüktür.

Yazılım mühendisliği terminolojisinin IEEE Standardı sözlüğüne [27] göre yazılım mühendisliği kavramı:

(1) Yazılımın geliştirilmesi, işletilmesi ve bakımı için sistematik, disiplinli, ölçülebilir bir yaklaşımın uygulanmasıdır, yani, yazılıma mühendisliğin uygulanmasıdır.

(2) (1). Maddede belirtilen yaklaşımlarla ilişkili çalışmalardır.

Yazılım mühendisliği aktiviteleri; yönetim, maliyetlendirme, planlama, modelleme, analiz, belirtme, tasarım yapma, uygulama, test etme, bakım gibi aktiviteleri içermektedir. “Mühendislik yaklaşımı” ifadesi her aktivitenin anlaşılıp kontrol edilmesi, böylece yazılımın analizi, tasarımı, yapılması ve bakımı gibi aşamalarında daha az sürprizle karşılaşılmasını ifade eder.

Yazılım yaşantımızın içerisinde olduğundan yazılım mühendisliğinin önemi anlaşılamamaktadır. Halbuki, fırın denetimlerinden arabalara, banka transferlerinden hava trafik kontrolüne, sofistike santrallerden sofistike silahlara kadar, yaşantımız ve hayat kalitemiz yazılıma bağlıdır. Literatür bütçeyi ve zamanı aşan proje örnekleriyle doludur. Daha kötüsü, yaşamları ve işleri riske sokan birçok yazılım hikayesi de bulunmaktadır [26].

Yazılım mühendisleri süreçleri ve ürünleri iyileştirmek için yeni teknik araçları ele almıştır. Ancak, ne yazık ki ölçüme yeteri kadar önem verilmemektedir. Ölçüm yapıldığında genellikle tutarsız, eksik ve nadir yapılan ölçüm sonuçları elde edilmektedir. Bu durum yazılım ürünü hakkında sağlıklı değerlendirme yapmayı engellemektedir. Planlamalar yanlış olabilmekte ve böylece zamanında teslim edilemeyen, bütçenin üstünde tamamlanmış projeler ortaya çıkmaktadır. Bu sebeplerden ötürü, ölçümün önemi anlaşılmalı ve bu doğrultuda aksiyonlar gerçekleştirilmelidir.

Yazılım mühendisliğinde birçok ölçüm çeşidi bulunmaktadır. İleriki bölümlerde bu ölçüm çeşitlerinden bazıları detaylı olarak anlatılmaktadır. Fenton ve Bieman’a [26] göre sıklıkla kullanılan ölçüm çeşitleri Çizelge 2.1’de özetlenmiştir. Buna göre, projenin planlamasında yapılacak işleri planlama adına ay ve günler ölçülüp adam/ay başına hesaplama yapılabilir. Program kodunun uzunluğuna bakarak bir ölçüm yapılabilir. Entegrasyon testlerine bakılarak bulunan hata oranlarına göre bir ölçüm yapılabilir. Test setlerine bakarak ise verim ve etkinlik niteliğine göre hata sayıları ölçülebilir. Ayrıca program koduna bakarak güvenilirlik kriterine göre hata bulunma yüzdeleri ve hatalar arasında geçen ortalama süre hesaplanabilir.

Çizelge 2.1. Yazılım Mühendisliğinde Kullanılan Spesifik Ölçüm Çeşitleri [26]

Varlık	Nitelik	Ölçüm
Tamamlanmış proje	Zaman	Başlangıçtan bitişe kadar olan aylar
Tamamlanmış proje	Zaman	Başlangıçtan bitişe kadar olan günler
Program kodu	Uzunluk	Kod satır sayısı (LOC)
Program kodu	Uzunluk	Çalıştırılabilir ifade sayısı
Entegrasyon testi süreci	Süre	Başlangıçtan bitişe kadar olan saatler
Entegrasyon testi süreci	Hata bulunma oranı	Bin satır başına bulunan hata sayısı
Test seti	Verim	Test durumları başına bulunan hata sayısı
Test seti	Etkinlik	Bin satır başına bulunan hata sayısı
Program kodu	Güvenilirlik	CPU saatinde hatalar arasında geçen ortalama sure (MTTF)
Program kodu	Güvenilirlik	CPU saatinde hata bulunma yüzdesi (ROCOF)

2.3. Yazılım Mühendisliği'nde Ölçme Amaçları

Her yazılım projesinin temel hedefi, müşterinin ihtiyaçlarını karşılayan, öngörülmüş bütçe ile zamanında teslim edilen hatasız bir yazılım geliştirmektir. Bu hedefe ulaşmak için planlamanın ve tahminlemenin iyi yapılması gerekir. Yazılımın ölçülebilmesi; harcanılan zaman, emek, proje büyüklüğü ve kalite gibi faktörlerin belirlenmesine olanak sağlamaktadır. Kurumlar bu verilere dayanarak projeler için kestirimler yapabilir ve böylece başarı ile tamamlanan projeleri gerçekleştirebilirler.

Ölçme, yazılım geliştirme sürecinin başında yapıldığı gibi, geliştirme sonlanana kadar devam edebilmelidir. Yazılım geliştirme sürecinin başında büyüklük, emek ve maliyet kestirimleri geliştiricilerin ve yöneticilerin karşılaştığı en önemli problemlerdendir. Proje geliştirme süreci boyunca bu ölçümleri devam ettirmek zordur. Ancak, proje sıkıntılı bir durumda olmasa bile, ölçme yapmaktan vazgeçilmemelidir. Çünkü, başarısızlığın sebepleri araştırıldığı kadar, başarının da sebepleri yeri gelince sorgulanacaktır. Ayrıca, herhangi bir ölçme yapmadan projenin başarılı mı başarısız mı olduğunu söylemek zordur.

Fenton ve Bieman'a [26] göre hem yönetici hem de yazılımcı perspektifinden yazılım geliştirme projelerini yönetmek ve anlamak için gerekli olan bazı bilgi çeşitleri vardır.

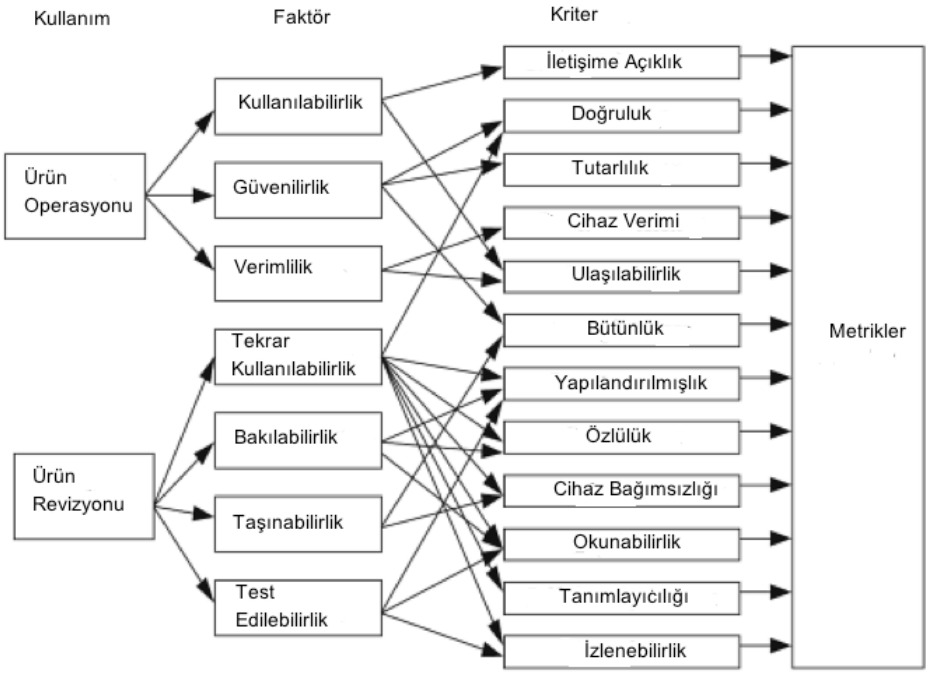
Yöneticiler için:

- Her bir süreç ne kadara mal oluyor?
- Ekip ne kadar üretici?
- Geliştirilen kod ne kadar iyi?
- Müşteri üründen memnun kalacak mı?
- Nasıl iyileştirme yapabiliriz?

Yazılımcılar için:

- Gereksinimler test edilebilir seviyede mi?
- Tüm hataları bulduk mu?
- Ürün ya da sürecin amacını karşıladık mı?
- Gelecekte ne olacak?

Yukarıda belirtilen soruların cevabını doğru bir şekilde vermek için gerekli olan en önemli şey ölçümdür. Şekil 2.1 yazılım ölçüsünü içeren metrikler için yazılım kalite modelini göstermektedir.



Şekil 2.1. Yazılım Kalite Modeli [26]

2.4. Yazılım Büyüklük Ölçümleri

Yazılım büyüklük ölçümlerini Fenton [26] dört ana başlık altında toplamıştır. Bunlar, *uzunluk* (fiziksel büyüklük), *işlevsellik* (kullanıcının aslında ne aldığı), *karmaşıklık* (yazılımın çözdüğü altta yatan problem) ve *tekrar kullanılabilirliktir* (ortaya çıkan ürünün ne kadarının yeni özellikler eklenerek oluşturulduğu).

2.4.1. Uzunluk

Fenton'a [26] göre üç önemli geliştirme ürününün büyüklüğünün bilinmesi yararlıdır. Bunlar, belirtim ("specification"), tasarım ve koddur. Belirtimlerin uzunluklarının ölçülmesi, tasarımının ne büyüklükte olacağı için yararlı bir gösterge olabilir, ayrıca kod uzunluğunun tahmininde kullanılabilir. Benzer şekilde, erken ürünlerin uzunluğu, sonrakiler için ne kadar efor harcanması gerektiğinin göstergesi olabilir [26].

2.4.1.1. Kod Satır Sayısı

Kod satır sayısı, yazılım projeleri için yazılımın büyüklüğünün göstergesi olarak en yaygın kullanılan metriktir [28]. 1960 yılı civarında ortaya çıkan bu yöntem ayrıca COCOMO, SLIM ve SEER-SEM gibi önemli birçok maliyet tahmin modelleri için de ana girdi olmuştur. Kod satır sayısı üç ana amaç için yarar sağlamaktadır [29] :

- Geliştirme ya da bakım maliyeti için tahminleme
- Diğer ölçümler için bir değişken olmak, onları aynı kod yoğunluğuna göre normalleştirme (“normalizing”)
- Diğer metriklerin değerlendirilmesinden farklı olan bir standart.

Kod satır sayısı metriğinin iki temel tipi bulunmaktadır: fiziksel kod satır sayısı (“physical LOC”) ve mantıksal kod satır sayısı (“logical LOC”) [30]. Fiziksel kod satır sayısı, program kaynak kodundaki “boşluksuz, yorumsuz satırlarının” toplamını ifade eder. Mantıksal kod satır sayısı, kaynak koddaki “ifade” lerin sayısının ölçümünde kullanılır. Fiziksel kod satır sayısını hesaplamak ve ifade etmek daha kolaydır. Ancak fiziksel kod satır sayısı metriği mantıksal olarak birbirleriyle ilgili olmayan formatlama ve stillere karşı hassastır. Bazı kaynaklar, bu iki temel kod satır sayısı metrik tipine ek olarak fiziksel satır başları (“physical carriage returns”) diye bir tipi de bu kapsamda incelemektedir. Fiziksel satır başları, fiziksel kod satır sayısından farklı olarak yorum ve boş satırları da hesaplamaya katar.

Nassif, Capretz ve Ho’ya göre [31] C++ programlama dilinde hesaplanan kod satır sayısı öçümü bu üç tip bazında Çizelge 2.2’de karşılaştırılmıştır.

Çizelge 2.2. Fiziksel Satır Başları, Fiziksel Kod Satır Sayısı ve Mantıksal Kod Satır Sayısının Karşılaştırılması [31]

C++ Dilinde Örnek	Fiziksel Satır Başları	Fiziksel Kod Satır Sayısı	Mantıksal Kod Satır Sayısı
double m_area(double b, double h)	1	1	1
{	2	2	
//comment	3		
//xyzxyz	4		
if(b==h)	5	3	2
//do nothing	6		
b=h;	7	4	3
double area; //area	8	5	4
area= (b*h) /2;	9	6	5
	10		
return area;	11	7	6
	12		
}	13	8	

Kod satır sayısı metriğinin bazı avantajları bulunmaktadır [32]. Bunlar:

- Sayma Otomasyon Kapsamı: Kod satır sayısı fiziksel bir varlık olduğundan manuel hesaplama maliyeti otomatik hesaplama sürecinden kolaylıkla ayrılabilir.
- Sezgisel Metrik Oluşu: Yazılımın büyüklüğünün ölçülmesinde sezgisel bir metrik şeklinde davranır. Çünkü görülebilir ve etkisi görselleştirilebilir. Böylece, az tecrübeli olan programcılar tarafından da yazılımın büyüklüğü ifade edilebilir.

En eski metriklerden biri olan kod satır sayısının yukarıda belirtilen avantajlarının yanı sıra yetersiz kaldığı bazı durumlar da bulunmaktadır. Bunlar [33]:

- *Programlama dili:* Farklı programlama dilleriyle yazılan uygulamalarda kod satır sayısını karşılaştırmak güçleşmektedir. Günümüzde birçok programlama dili bulunmaktadır. Yazılımcıların uygulamanın ihtiyaçlarına göre tercih ettiği farklı diller aynı koşullarda bir değerlendirme yapılmasını engellemektedir. Spesifik bir fonksiyon yazmak bir dilde 10 satır almakta iken, başka bir dilde bu fonksiyon 30 satır sayısı olabilmektedir.
- *Kodlama stili:* Değişik kodlama stilleri ile hesaplama tam anlamıyla doğru olmamaktadır. Geliştiriciden geliştiriciye fark eden tecrübe, alışkanlık gibi faktörler aynı özelliğe sahip farklı kod satır sayılarının çıkmasına sebebiyet vermektedir. Bir geliştirici 10 satırda bir işlevselliği ifade eden durumu yazabilmekte iken, başka bir geliştirici aynı işlevselliği 5 satırda yazabilmektedir. Ancak bu durum az üretkenlikle karıştırılmamalıdır.
- *Kod satır sayısının ölçülme şekli:* Kod satır sayısı nasıl ölçülmektedir? Boş satırlar ve yorumlar ölçüme dahil edilmiş midir? Eğer kod satır sayısı ölçümü diğer sistemlerin kod satır sayılarıyla karşılaştırılacak ise bu bilgi önemlidir.

Yukarıda belirtilen sorunlara çözüm olarak Park [34], kurumlar tarafından ölçülecek kod satır sayısının ölçümünü tarif etmek amacıyla kontrol listelerini içeren bir yapı hazırlamıştır. Bu listeler, yorumlamayı etkileyecek tüm olası faktörleri (yorumlar, dil vb.) tarif etmektedir.

2.4.1.2. Belirtimler ve Tasarım

Yazılımın erken yaşam döngülerinde kullanılan belirtim ve tasarım dokümanları genellikle yazı, grafik ve özel matematiksel diyagram ile sembolleri içermektedir. Sektörde dokümanların sahip oldukları sayfa sayıları, projenin büyüklüğünü ifade edecek metrik olarak görülmektedir. Ancak, aynı doküman içinde hem yazı hem diyagram olabileceği ve bunun yanında diyagramların çeşidinin de kendi içinde farklılaşabilmesi sebebiyle, atomik öğe olarak daha genel seçimler yapılması gerekliliği ortaya çıkmıştır [26].

2.4.2. İşlevsellik

Uzunluk, çoğu yazılım mühendisine göre yanıltıcıdır. Ürünün özünde olan işlevsellik ürünün boyutu için daha iyi bir resim çizmektedir. Özellikle, geliştirme sürecinin erken aşamalarında ortaya çıkan iş ürünleri için işgücü ve süre tahmininde genellikle fiziksel büyüklükten ziyade, işlevsellik tahmini tercih edilmektedir. Farklı bir özellik olarak, işlevsellik sezgisel olarak, tamamlanan üründeki işlev miktarını veya tanımlamadaki ürünün nasıl olması gerektiğini yansıtır [26]. Yazılım uzunluğu daha çok geliştirici perspektifinden büyüklüğü yansıtırken, işlevsellik kullanıcı bakış açısından büyüklüğü yansıtmaktadır.

2.4.2.1. Hikaye Puanları

Hikaye puanları, çevik yazılım geliştirme sürecinde kullanılan bir metriktir.

Çevik yazılım süreçleri, 1950'lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımların, yazılım sektöründe bir uzantısı olarak ortaya çıkmıştır. Yazılım dünyasında çeşitli çevik yaklaşımlara 1970'lerden itibaren rastlanabilmekle birlikte, çevik yazılım metodolojilerinin kullanımı 1990'larda hız kazanmış ve geçtiğimiz son 7-8 yıl içerisinde de tüm dünyada başarılarını kanıtlayarak popülaritesini arttırmıştır. Şu anda, dünyadaki birçok yazılım şirketinde ve birçok yazılım projesinde yazılımlar, çevik yaklaşımlarla geliştirilmektedir [35].

Çevik yazılım geliştirme süreci kullanıcı hikayelerinden ("user story") oluşur. Kullanıcı hikayesi, sistemin ya da yazılımın kullanıcısı ya da müşterisi için değerli olan işlevselliğini ifade eder [36]. Hikayeler ekip tarafından tanımlanmalı ve

tahminlemeler bireysel kişiden ziyade, ekip tarafından yapılmalıdır. Kullanıcı hikayeleri üç yönden oluşmaktadır:

- Planlama ve hatırlatma için kullanılan hikayenin yazılı açıklaması
- Hikayenin detaylarını ayrıntılı bir şekilde anlatmak için hikaye hakkındaki konuşmalar
- Hikayenin ne zaman tamamlanacağına karar vermede kullanılabilecek ve detayları belgelemek ve iletmek için olan testler [36].

Çevik yazılım sürecinde hikaye puanları, Scrum takımları tarafından hikayeyi ("story") gerçekleştirmek için gerekli olan maliyeti ölçme amacıyla kullanılmaktadır. Hikaye puanları sayesinde hikayenin ne ölçüde büyük olduğu ifade edilmektedir. Hikaye puanları genel olarak 1, 2, 3, 5, 8, 13, 21, 34, 55 sayılarını içeren Fibonacci serisinden oluşmaktadır. Hikaye puanları gerçek saatlerle direkt olarak ilişkili değildir. Bu sebeple, Scrum takımlarının i hikayeyi tamamlamak için gerekli olan efor hakkında düşünmelerini kolaylaştırır. Cohn'a [36] göre bir hikayeye hikaye puanı verirken belirli özelikler göz önünde bulundurulmalıdır. Bunlar Şekil 2.2'de belirtildiği gibi, maliyet, süre ve karmaşıklık gibi durumlardır.

Şekil 2.2. Hikaye Puanı

Çevik yazılım geliştirmede sürümler, bir ya da daha fazla tekrarlamadan ("iteration") oluşur. Sürüm planı, proje zaman çizelgesi ve istenilen işlevsellik arasındaki dengeye karar verilmesini ifade eder. Tekrarlama planı, o tekrarlamada olacak hikayelerin seçilmesini ifade eder. Müşteriler ve yazılımcılar sürüm ve tekrarlama planlamalarında bulunurlar. Sürün planlanırken, müşteri ekibi hikayeleri önceliklendirir. Daha sonra, öncelik sırasına göre listelenen hikayelere, hikaye puanları atanarak planlama yapılmış olur. Hız ("velocity"), tekrarlama süresince

hedeflenen hikaye puanını ifade eder. Örneğin, projedeki tüm hikayeler Çizelge 2.3'teki gibi öncelik sırasına göre listelemektedir. Ekip, bu hikayelere göre belirledikleri bir hıza göre (on üç hikaye puanı gibi) Çizelge 2.4'teki gibi her bir tekrarlama için plan yapar.

Çizelge 2.3. Örnek Hikayeler ve Maliyetleri

Hikaye	Hikaye Puanları
Hikaye A	3
Hikaye B	5
Hikaye C	5
Hikaye D	3
Hikaye E	1
Hikaye F	8
Hikaye G	5
Hikaye H	5
Hikaye I	5
Hikaye J	2

Ekip, on üç hikaye puanlı bir tekrarlama planladığından, bu sınırı geçmemeye dikkat edilir ve tekrarlamalar on üç hikaye puanından fazla olmaz. İkinci ve üçüncü tekrarlamalar bu sebeple sınırı geçmemek amacıyla on iki hikaye puanı alacak şekilde planlanır.

Çizelge 2.4. Çizelge 2.3 için Sürüm Planı

Tekrarlama	Hikayeler	Hikaye Puanları
Tekrarlama 1	A,B,C	13
Tekrarlama 2	D,E,F	12
Tekrarlama 3	G,H,J	12
Tekrarlama 4	I	5

Tekrarlama sonunda ekip tamamlanan hikaye puanı sayısını toplar. Elde edilen sonuç, aynı uzunluktaki gelecek tekrarlamalar için ne kadar hikaye puanı tamamlanabileceğinin tahmin edilmesinde kullanırlar. Örneğin, ekibin iki hafta

süren tekrarlamada 30 hikaye puanını tamamladığını düşünelim. Bizim en iyi tahminimiz, sonraki tekrarlamada da 30 hikaye puanının tamamlanması şeklinde olmalıdır.

2.4.2.2. Kullanım Durumu Puanları

Kullanım durumu modeli, yazılım projelerinin iş süreçleri ve gereksinimlerini yakalamada kullanılan yaygın bir modeldir [37]. Bu metot, kullanım durumu diyagramlarının kullanım durumu puanı şeklinde ifade edilen büyüklük metriğine eşlenmesini temel alır. Kullanım durumu modeli ilk olarak Jacobson ve arkadaşları [38] tarafından ortaya atılmıştır. Kullanım durumu puanı fikri ise ilk olarak 1993 yılında Karner [39] tarafından ortaya çıkmıştır.

Kullanım durumu diyagramı, kullanıcıların sistemle nasıl etkileşimde olduğunu gösterir. Kullanım durumu diyagramı, kullanım durumları ve aktörlerden oluşur. Kullanıcının aktör rolüne sahip olduğu kullanım durumlarında, işlevsel gereksinimler ifade edilir.

Kullanım durumu puanı işlemi üç değişkenden meydana gelmektedir [37].

1. Düzeltilmemiş Kullanım Durumu Puanı (“Unadjusted Use Case Points-UUCP”)
2. Teknik Karmaşıklık Faktörü (“The Technical Complexity Factor - TCF”)
3. Ortam Karmaşıklık Faktörü (“Environment Complexity Factor – ECF”)

Her bir değişken ağırlıklı değerlerine, öznel değerlerine ve sabit kısıtlarına göre ayrı ayrı tanımlanır ve hesaplanır.

Bunlara ek olarak, zamanı ifade etmek için katsayı olarak verim gerektiğinde, projeyi tamamlamak için gerekli olan adam-ay sayısının tahmininde denklem kullanılabilir. Verim faktörü (“Productivity Factor– PF") aşağıdaki denklemle ifade edilir [37]:

$$UCP = UUCP * TCF * ECF * PF \quad (2.1)$$

Kullanım durumu puanı kullanmanın bazı avantajları vardır.

İlk avantaj, ölçme işleminin manuel yapılmasının yanı sıra var olan araçlarla otomatik olarak da yapılabilmesidir. Bazı kullanım durumu yönetimi araçları sistemdeki kullanım durumu puanlarını otomatik olarak sayar, böylece hesaplama için harcanan süre azalmış olur.

İkinci bir avantaj, şirketin kullanım durumu puanı başına ortalama gerçekleştirme zamanının belirlenmesidir. Gelecek programlar için tahmin yapılması gerektiğinde bu bilgi yararlı olacaktır. Ancak doğru bir varsayımda bulunabilmek için, tüm kullanım durumlarının aynı detay seviyesinde yazılması gerekmektedir. Kullanım durumunu yazan kişilerin farklı alışkanlıklarda olduğu düşünülürse bu durumu sağlamak zorlaşacaktır.

Üçüncü avantaj, kullanım durumu puanı kullanmanın çok sade ve basit bir işlem gerektirmesidir. Hesaplanacak olan uygulamanın büyüklüğü, takımın tecrübeleri, alışkanlıkları vb. özelliklerine bakmadan bir sonuç üretir.

Kullanım durumu puanının avantajları olduğu kadar dezavantajları da bulunmaktadır. Ana problem, hesaplamanın tüm kullanım durumlarının yazılması bitene kadar yapılamamasıdır. Günümüzdeki yazılım projelerinde, sürüm yetiştirme derdinde olan birçok ekip kullanım durumlarına gereken önemi vermediğinden kullanım durumlarına göre bakarak bir hesaplama yapmak zorlaşmaktadır. Planlama aşamasında kullanım durumlarının tam olarak yazıldığı varsayılsa bile, yazılıma yeni özellikler eklendiğinde, ya da var olan özellikler sistemden çıkarıldığında kullanım durumlarını güncel tutmak zordur.

Bir başka problem ise, bazı teknik faktörlerin tüm proje üzerinde gerçekten önemli bir etkisinin olmayışıdır. Hesaplama yapılırken katsayı olarak katılan bu değişkenlerin etkisi, sistemden sisteme değişebilmektedir.

2.4.2.3. Erken ve Hızlı İşlev Puan Analizi

Erken ve Hızlı (“Early&Quick - E&Q”) İşlev Puan analizi ilk olarak 1977 yılında Roberto Meli tarafından IFPUG İşlev Puan tahminini kolaylaştırmak amacıyla ESCOM 997 konferansında duyurulmuştur [40]. Projenin erken aşamalarında ölçüm detaylarının eksik olduğu ve gereksinimlerin tam anlamıyla netlik

kazanmadığı durumlarda, E&Q yaklaşımı proje planlama ihtiyacına büyük katkı sağlar.

E&Q tekniği yazılım sisteminin işlevsel büyüklüğünü daha iyi tahmin etmek için farklı kestirim yaklaşımlarını kombine eder. İşlevlerin (hareketler ve veri) hem analojik hem analitik sınıflandırmalarını kullanır. Ayrıca, sistemin farklı dalları için farklı detay seviyelerini kullanmaya izin verir (çok katlı yaklaşım). Son olarak teknik, tahminlerini statik ve analitik şeklinde geçerli tablo değerleriyle sunar [41].

Son yıllarda, ISO/IEC tarafından yazılım işlevsel büyüklüğü tanımınca E&Q tekniği gelişmiş ve genelleştirilmiş, uygulanabilirliği genişletilerek COSMIC işlev puan ölçüm metodunda tanım kümesi olmuştur [42]. Çizelge 2.5 E&Q tekniğinin, COSMIC işlev puan yöntemi cinsinden belirlenen aralıklar için karşılık gelen değerleri göstermektedir.

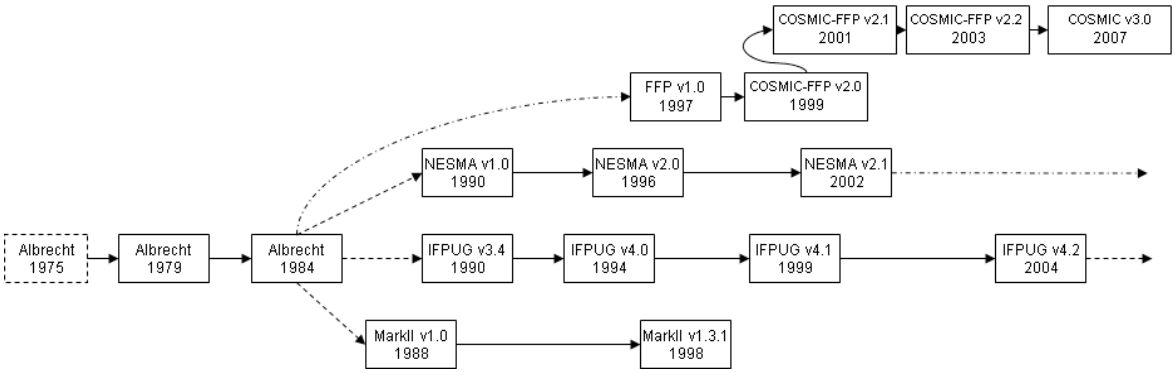
Çizelge 2.5. E&Q-COSMIC Dönüşümünde Bileşen Aralıkları ve Değerleri [42]

Tip	Seviye	Aralık/COSMIC Eşleniği	Min. CFSU	Ortalama CFSU	Max. CFSU
İşlevsel Süreç (İS)	Küçük	1-5 Veri hareketi	2.0	3.9	5.0
	Orta	5-8 Veri hareketi	5.0	6.9	8.0
	Büyük	8-14 Veri hareketi	8.0	10.5	14.0
	Çok Büyük	14+ Veri hareketi	14.0	23.7	30.0
Tipik Süreç	Küçük	CRUD (Küçük/Orta süreçler); CRUD + Listeleme (Küçük süreçler)	15.6	20.4	27.6
	Orta	CRUD (Orta/Büyük süreçler); CRUD + Listeleme (Orta süreçler) CRUD + Listeleme + Raporlama (Küçük süreçler)	27.6	32.3	42.0
	Büyük	CRUD (Büyük süreçler); CRUD + Listeleme (Orta/Büyük süreçler) CRUD + Listeleme + Raporlama (Orta süreçler)	42.0	48.5	63.0
Genel Süreç (GS)	Küçük	6-10 genel İS	20.0	60.0	110.0
	Orta	10-15 genel İS	40.0	95.0	160.0
	Büyük	15-20 genel İS	60.0	130.0	220.0
Makro Süreç	Küçük	2-4 genel GS	120.0	285.0	520.0
	Orta	4-6 genel GS	240.0	475.0	780.0
	Büyük	6-10 genel GS	360.0	760.0	1,300.0

2.4.2.4. İşlev Puanı

İşlevsel puan analizi (“Function Point Analysis - FPA”), yazılımın sunduğu işlevsellik yönüyle büyüklüğünün hesaplanmasıdır. İşlevsel büyüklük ölçme yöntemi, yazılımın işlevsel gereksinimlerini ölçer. Bu özelliğiyle, satır sayısı vb. metotlarından farklı olarak yazılım sürecinin başında da ölçme işlemi yapılmasına olanak sağlar. Bu yeteneği ile projenin başında kestirim yapmak mümkündür. Buna ek olarak, proje süresi zarfında ve proje bitiminde de ölçüm yapmaya olanak sağlar ve kodun güncel halinden yola çıkılarak hesaplama işlemi yapılabilen işlevsel büyüklük kestirim yöntemi sayesinde doğru sonuçlar alınabilir. Diğer teknik ölçme yöntemlerinden farklı olarak kullanıcılar tarafından daha kolay anlaşılır ve iş terimleri ile kolayca yorumlanır.

Şekil 2.3 işlevsel büyüklük kestirim yöntemlerinin tarihçesini göstermektedir. En yaygın kullanılan yöntemler, Mark II FPA [11], NESMA FPA [12], IFPUG FPA [13] ve COSMIC FFP [14] dir.



Şekil 2.3. İşlevsel Büyüklük Kestirim Yöntemlerinin Tarihçesi [43]

IFPUG İşlev Puan Analizi

İşlev Puan Yöntemi, ilk olarak 1979 yılında IBM araştırmacısı olan Allan Albrecht [2] tarafından tasarlanmıştır. Kod satır sayısı tabanlı ölçümlerdeki kısıtlamalara çözüm getirmek amacıyla geliştirilmiştir. İşlev puan yöntemi fikri, kullanıcıların gözünden işlevsel gereksinimler bazında büyüklüğe karar verme amacıyla ortaya çıkmıştır.

Yönetim bilgi sistemleri için tasarlanan bu metot, daha sonra bu alanda pratikte yaygın bir şekilde kullanılan bir standarda dönüşmüştür [43]. Günümüzde yönetim

bilgi sistemlerine ek olarak, yönetim bilgi sistemi olmayan alanlarda (Gerçek-zamanlı, Web, Nesne tabanlı, Veri madenciliği sistemleri vb.) da bu metot kullanılmaktadır.

Bu yaklaşım, geliştirme projesi tarafından teslim edilen harici kullanıcı girdileri, sorguları, çıktıları ve ana dosyaları sayar ve listeler. Girdi ve çıktıların her bir kategorisi ayrı ayrı sayılır ve kullanıcıya/müşteriye ifade edilen işlev cinsinden sayıları ağırlıklı olarak hesaplanır. Girdi ve çıktıların ağırlıklı toplamı "işlev puanı" diye ifade edilir [44].

Albrecht'in İşlev Puanı yönteminde aşağıda özetlendiği gibi sistemin kendine has büyüklükte iki bileşeni hesaplanır [45] :

1. *Süreç büyüklüğü bilgisi*, ilk olarak son kullanıcı tarafından görülen sistem bileşenlerinin belirlenmesi ile karar verilir. Daha sonra bu bileşenler harici (mantıksal) girdiler, çıktılar, sorgular, diğer sistemlerle olan harici arayüzler ve dahili mantıksal dosyalar olarak beş parçaya ayrılır. Her bir bileşen ayrıca içerdikleri veri elementleri sayısına ve diğer faktörlere göre "basit", "orta" ve "karmaşık" şeklinde ayrılır. Her bir bileşene sahip olduğu tip ve içerdiği karmaşıklığa göre bir puan atanır ve tüm bu bileşenlerin toplamı "Düzeltilmemiş İşlev Puanlar" ("Unadjusted Function Points") şeklinde ifade edilir.
2. *Teknik Karmaşıklık Faktörü,* belirlenmiş 14 bileşenin genel uygulama karakteristiklerinin (Şekil 2.4) etki derecesinin tahmini ile karar verilir. Etki derecesi sıfırdan (etkisi yok anlamında) beşe (güçlü etki) kadardır. 14 özelliğin toplam skoru, toplam etki derecesi ("degrees of influence – DI") olarak ifade edilir. Toplam etki derecesi, teknik karmaşıklık faktörüne ("technical complexity factor" – "TCF") aşağıdaki formülle çevrilir.

$$TCF = 0.65 + 0.01 * DI \quad (2.2)$$

3. İşlev puanı ise aşağıda belirtilen formülle hesaplanır:

$$FP = UFP * TCF \quad (2.3)$$

İşlev puanı genellikle sistem geliştirme projelerinde bütçe hesaplamak amacıyla kullanılmaktadır. İşlev puanı sayesinde adam-ay değerine göre planlama kolaylıkla yapılabilir.

İşlev puanı yöntemi hızlıdır, düzgün hazırlanmış belgeler ile işlev puanını hesaplamak uzun sürmez. İşlev puanı kullanmak birçok avantaj sunar. Bunlar [46];

- Daha iyi ve erken proje maliyeti tahmini ve bütçeleme
- Projeleri daha iyi kontrol etme
- Verimliliği ölçme
- Bilgi sistemi kalitesini ölçme
- Sistem geliştirme sürecini iyileştirme.

İşlev puanının sağlamadığı özellikler ise;

- İşlev puanı yöntemi proje yönetimi metodu değildir
- İşlev puanı otomatik olarak hatasız proje tahmini sağlamaz
- İşlev puanı proje planlama metodu değildir.

İşlev Puanı:

Tür No	Tanım	Karmaşıklık			Toplam
		Basit	Orta	Kompleks	
IT	Harici Girdiler (External Input)	----x 3=----	----x 4=-----	----x 6=----	----------
OT	Harici Çıktılar (External Output)	----x 4=----	----x 5=-----	----x 7=----	----------
FT	Mantıksal İç Dosya (Logical Internal File)	----x 7=----	----x10=----	----x15=---	----------
EI	Harici Arayüz Dosyaları (External Interface File)	----x 5=----	----x 7=-----	----x10=---	----------
QT	Harici Sorgular (External Inquiry)	----x 3=----	----x 4=-----	----x 6=----	----------
FC	Düzeltilmemiş Toplam İşlev Puanları				----------

İşlem Karmaşıklığı:

No	Özellik	Etki Derecesi		No	Özellik	Etki Derecesi
C1	Veri İletişimi	--------		C8	Online Güncelleme	----------
C2	Dağıtımlı Fonksiyonlar	--------		C9	Kompleks İşlem	----------
C3	Performans	--------		C10	Tekrar Kullanılabilirlik	----------
C4	Yoğun Olarak Kullanılan Konfigürasyon	--------		C11	Kurulum Kolaylığı	----------
C5	İşlem hızı	--------		C12	Operasyonel Kolaylık	----------
C6	Online Veri Girişi	--------		C13	Çoklu Bölge	--------
C7	Son Kullanıcı Etkisi	--------		C14	Değişikliği Kolaylaştırma	--------
PC			Etki Derecesi Toplamı			----------

Etki Derecesi Değerleri:

- Mevcut değil, ya da etkisi yok = 0
- Önemsiz etki = 1
- Orta etki = 2
- Ortalama etki = 3
- Önemli etki = 4
- Güçlü etki = 5

Şekil 2.4. İşlev Puanı Hesaplama Çalışması [44]

2.4.3. Karmaşıklık

Bazı işlevsellik ölçerleri, gereksinim belirtimleri ile problemin karmaşıklığını ayarlamaya çalışmaktadır. Örneğin, Albrecht'in teknik kalite hesaplaması ("technical quality calculation") esas problemin ölçümünü varsayar. Ancak problemin birden fazla çözümü olabilir ve çözümler yaklaşımları yönünden farklılaşabilir ve böylece karmaşıklık yönlerinden de birbirlerinden farklı olurlar. Bu sebeple, çözümün karmaşıklığı, büyüklükten ayrı bir bileşen olarak düşünülmeli ve daha objektif bir yolla ifade edilmelidir [26].

İdeal olarak, çözümün karmaşıklığı, problemin karmaşıklığından daha büyük olmamalıdır, ancak bu her zaman böyle değildir.

Çözüm karmaşıklıkları iki farklı yönden ele alınabilir [26]:

- Zaman karmaşıklığı: Kaynak, sistem zamanıdır.
- Alan karmaşıklığı: Kaynak, sistem hafızasıdır.

2.4.4. Tekrar Kullanılabilirlik

Program geliştiriciler, dördüncü nesil diller ve pencere ortamları (“windowing environments”) ile diğer işgücü-tasarruf teknikleri, bazı görevlerin tekrarlanmasından yararlanmaktadırlar. Eğer geliştirilmiş ürünü, başka ürünler için de ifade edebiliyorsak yeni bir sıralama rutini, sensör-gözlemleme yazılımı yazmamıza gerek kalmamaktadır. Bu sayede sıklıkla kullandığımız işletim sistemleri, derleyiciler ve veri tabanı yönetim sistemlerini tekrar tekrar yazmadan kullanabiliyoruz. Yazılımların yeniden kullanılabilirliğI (gereksinim, tasarım, dokümantasyon, test verileri ve kod gibi olan senaryoları içeren), verimliliğimizi ve kalitemizi arttırır ve eski problemlerin çözümüne vakit harcamak yerine, yeni problemler üzerine yoğunlaşabilmemizi sağlar [26].

2.5. Büyüklük Kestirim Yöntemlerinin Karşılaştırılması

Yazılım projelerinde kaliteyi arttırmak, her şeyden önce doğru ölçme yöntemlerine bağlıdır. Bu kapsamda birden fazla kestirim yöntemi kullanılabilir. Büyüklük ölçme yöntemini seçmede rol oynayan kriterler aşağıda belirtilmiştir [12]:

- *Kişilerden bağımsız olma:* Ölçmenin kim tarafından yapılacağı önemli olmamalıdır.
- *Tekrarlanabilirlik:* Aynı ölçüm tekrar yapıldığında, sonuç her zaman aynı kalmalıdır.
- *Karşılaştırılabilirlik:* Farklı sistemlerin büyüklükleri karşılaştırılabilmelidir.
- *Teknolojiden bağımsız olma:* Sistemin hangi teknolojileri kullandığı önemli olmamalıdır. Teknik özellikler ölçüm sonucunu etkilememelidir.
- *Erken aşamada uygulanabilirlik:* Ölçüm sadece sistemin teslim aşamasında yapılmamalıdır. Sistem geliştirme sürecinde ne kadar ölçüm yapılırsa o kadar iyi verimi ve bütçeyi gözlemleyebilirsiniz.
- *Sadelik:* Birimler kolaylıkla anlaşılabilmelidir.
- *Geleceğe uyumlu:* Veriler yeni bir teknolojiye geçildiğinde dahi anlamlı ve doğrudan uygulanılabilir olmalıdır.
- *Kullanıcı değerinin ifadesi:* Birim, kullanıcıya sunulan işlevsellik hakkında bilgi verir.

- *İfade karmaşıklığı:* Birim, sahip olduğu veriyi işleyerek gizli karmaşıklığı gösterir.
- *ISO standardı.*

Ölçme metotlarının yukarıdaki özellikler kapsamında karşılaştırılması, Çizelge 2.6'da ifade edilmiştir.

Çizelge 2.6. Ölçme Metotlarının Karşılaştırılması [46]

	İşlev Puan Analizi	COSMIC	Hikaye Puanları	Kod Satır Sayısı	Kullanım Durumu Puanları
Kişilerden bağımsız olma	✓	✓	✗	✓	✗
Tekrarlanabilirlik	✓	✓	✗	✓	0
Karşılaştırılabilirlik	✓	✓	✗	✗	✗
Teknolojiden bağımsız olma	✓	✓	✗	✗	✓
Erken aşamada uygulanabilirlik	✓	0	✓	✗	✓
Sadelik	0	0	0	✓	0
Geleceğe uyumlu	✓	✓	0	✗	0
Kullanıcı değeri	✓	✓	✗	✗	✓
İfade karmaşıklığı	✗	✗	✓	✗	✗
ISO standardı	✓	✓	✗	✗	✗

✓ = karşılamakta

0 = makul değerde karşılamakta

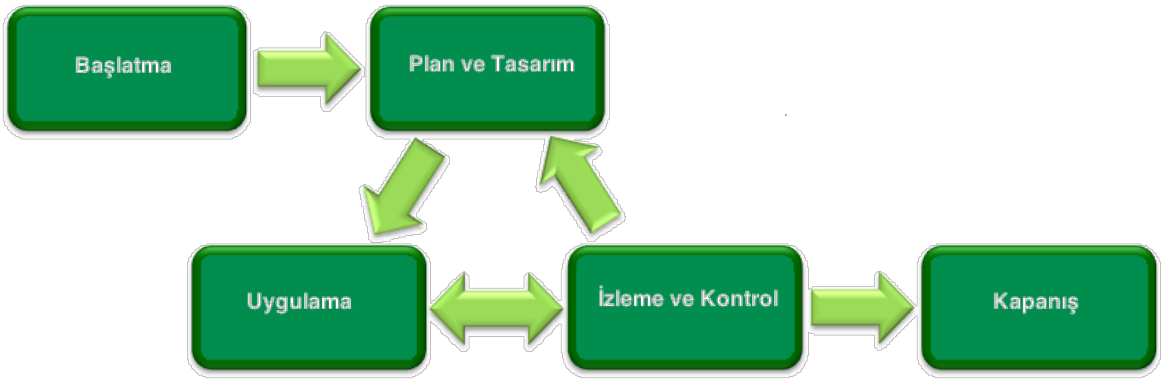
✗ = karşılamıyor

2.6. Proje Yönetimi ve İşlevsel Büyüklük

Fenton ve Bieman proje yönetimi tanımını, "paydaşların projeden beklentilerini karşılamak ya da aşmak için, proje etkinliklerine bilgi birikimi, beceri, araç ve tekniklerin uygulanması" olarak yapmaktadırlar [26].

Proje yönetimi genel olarak; Başlatma, Plan ve Tasarım, Uygulama, İzleme-Kontrol ve Kapanış adımlarını içermektedir. Şekil 2.5 proje yönetimi sürecinin adımlarını göstermektedir. Proje süresi boyunca devam eden ve kritik öneme

sahip olan “İzleme ve Kontrol” süreci, geriye dönük değerlendirme, geleceğe yönelik planlama yapma ve bu doğrultuda uygulamayı gerçekleştirme süreçlerini içerir.



Şekil 2.5. Proje Yönetimi Süreci [47]

Ölçme, proje yönetimi sürecinde “İzleme ve Kontrol” sürecinde ön plana çıkmaktadır. Proje yönetimi sürecinde ölçme, proje yönetimi planına göre planın neresinde olunduğuna, gerekli görülen adam-ay sayısına ve maliyet kestirimleri yapılmasına yardımcı olur.

İşlevsel büyüklüğün tekrarlamalı geliştirmede [7] [8] ya da çevik geliştirmede [9] [10] kullanılmasının faydalı olduğunun örnekleri literatürde yer almaktadır.

Önerilen model genel olarak izleme yaklaşımını benimsemektedir. Geliştirme dönemi bittikten sonra gerçekleştirilen gereksinimlerin işlevsel büyüklükleri otomatik olarak ölçülerek planlamanın ne kadarının gerçekleştirildiği anlaşılabilir. Böylece geliştirme dönemi planlananan işlev puanı sayısı, geliştirme dönemi sonunda gerçekleştirilen ile karşılaştırılarak, planlamanın ilerisinde veyahut gerisinde olunduğuna karar verilerek gerekli aksiyonlar alınabilir.

2.6.1. Proje Yönetimi Süreci

Ölçme, projelerde farklı şekilde uygulanabilir. Projenin başında analiz, tasarım ve gereksinimlerden yola çıkarak bir ölçme yapılabileceği gibi, proje süresince ve proje bitiminde de harcanan süre, maliyet, geliştirilen ürün vs. gibi verilerden yola çıkılarak da ölçme yapılması mümkündür.

Proje planlama, geliştirme ve geliştirme sonrasında işlev puanı sayısına bakılarak yapılabilecek ölçümler projenin takibi açısından önem taşıyabilir. Bu noktada,

yöneticilerin, ekibin ve varsa kalite ekibinin düzenli olarak ölçümleri yapması ve bu ölçüm sonuçlarına göre sağlıklı değerlendirme yapması sağlanabilir.

2.6.2. Kazanılan Değer

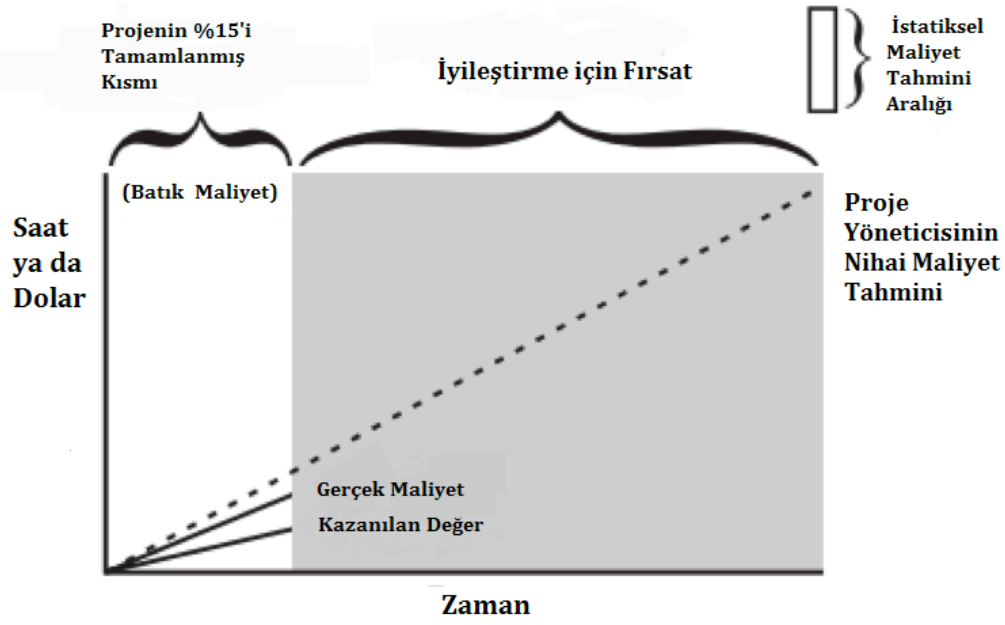
Kazanılan değer (“Earned Value – EV”) , son otuz yıldır kullanılan ve diğer önemli araçların yerini alan kanıtlanmış ancak yetersiz seviyede kullanılan proje yönetimi tekniğidir. Resmi uygulamada, Amerika Birleşik Devletleri’nin devlet kurumları tarafından satın alınan önemli yeni sistemleri yönetmek ve denetlemek amacıyla etkin bir araç olarak bulunmuştur. Daha basit haliyle, kazanılan değer herhangi bir yönetim projesinde kullanılan (yazılım projeleri de dahil olmak üzere) yararlı bir tekniktir [48].

Kazanılan değer proje yönetimi metodu, projenin kapsam, zaman ve maliyet yönetimini destekleyen etkili bir araçtır. Metot, maliyet hesaplanmasına, değişkenlerin ve performans endekslerinin planlanmasına, proje maliyetini tahmin etmeye ve proje tamamlanmasında program yapmaya olanak sağlar [6].

Kazanılan değer planlanan proje sonuçlarını proje performansıyla karşılaştırarak ileriye dönük planlama yapmaya, proje yöneticisinin ve proje ekibinin bu doğrultuda strateji belirlemelerine yardımcı olur.

Metot, maliyet ve planlama parametreleri için ölçüm olarak proje performansının maliyet ve değerini yaygın olarak kullanır. Maliyet ölçümünü ve değerini dolar, saat, çalışılan gün ve benzer birimler cinsinden ifade eder.

Metot kullanılarak, projenin henüz %15 i tamamlanmış haliyle bile doğru ve güvenilir değerler elde edilebilir. Şekil 2.6’da gösterildiği gibi, yöneticiler performans sonuçlarını kullanarak belirlenen sınırlar dahilinde projenin tamamlanana kadar ne kadar maliyete mal olacağını tahmin edebilirler. Elde edilen bu tahmin değerleriyle de proje için gereken maliyet doğrultusunda adam-ay cinsinden değerler elde edilmiş olur. Bu değerler günümüzde projelerin başarı ile sonuçlanmasında gerekli olan değerlerin önemli bir kısmını da oluşturmaktadır.



Şekil 2.6. "Erken Uyarı Sinyalleri" ile Maliyet Riskleri Yönetilebilir [48]

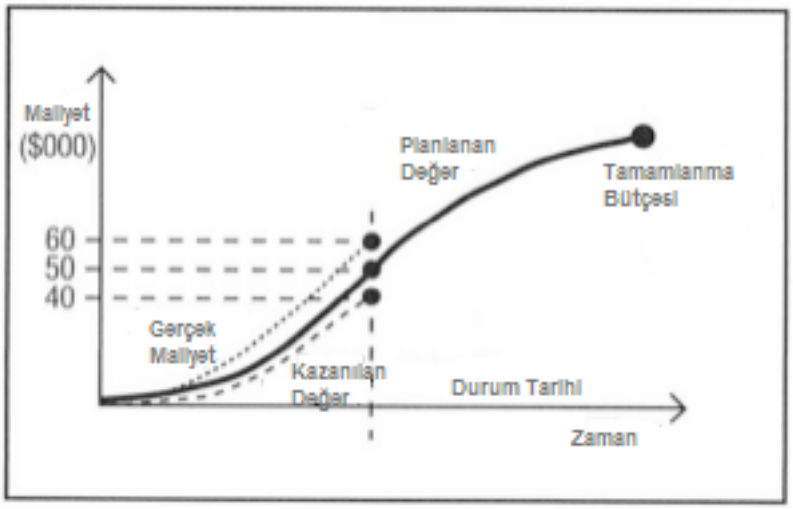
2.6.2.1. Kazanılan Değer Bileşenleri

Kazanılan değer, proje performansını değerlendirmek için aşağıda belirtilen proje parametrelerini kullanır [6]:

- **Planlanan Değer ("Planned Value – PV") :** Takvimle ilişkilendirilmiş bir aktiviteyi, iş paketini ya da ilgili projeyi tamamlamak için onaylanan bütçedir. Belirlenen zamana kadar, projede tamamlanacağı belirlenen fonksiyonların kazanılan değeri olarak görülebilir. Eskiden işin bütçelenen maliyeti ("budgeted cost of work – BCWS") olarak adlandırılmaktaydı.
- **Tamamlanma Bütçesi ("Budget at completion – BAC") :** Aktivite, iş paketi veya projenin tamamlanması için belirlenen toplam bütçedir. Planlanan değerin en yüksek değerine denk gelir. Kümülatif planlanan değer eğrisinde son noktadır.
- **Gerçek Maliyet ("Actual Cost – AC") :** Belirlenen zamanda, ilişkili değeri kazanmak için aktiviteyi, iş paketini ya da projeyi tamamlamak için harcanan kümülatif gerçek maliyettir. Önceleri yapılan işin gerçek maliyeti ("actual cost of work performed – ACWP") olarak isimlendirilmekteydi.
- **Kazanılan Değer ("Earned Value – EV") :** Belirlenen zamana kadar tamamlanan işin kümülatif kazanılan değeridir. Hedeflenen işin herhangi bir

zamana kadar yapılması için gerekli bütçe miktarını ifade eder. Önceleri yapılan işin bütçelenen maliyeti (“budgeted cost of work performed – BCWP") olarak isimlendirilmekteydi. Kazanılan değeri elde etmek için, işin toplam bütçesi ile tamamlanan yüzdesi çarpılır.

Şekil 2.7 yukarıda tanımlanan değerler arasındaki ilişkiyi göstermektedir.



Şekil 2.7. Planlanan Değer, Tamamlanma Bütçesi ve Gerçek Maliyet

Somut bir örnek vermek gerekirse, bir projeye 40 haftalık bir zaman zarfında ürün teslimi yapması için 100.000 TL bütçe harcandığını düşünelim. Bu değerlere göre 20 hafta sonunda projenin %50 sinin tamamlanması beklenmektedir. Ancak 20 hafta sonundaki raporda projenin %40’ının tamamlandığı ve 60.000 TL bütçe harcandığı belirtilmiş olsun. Bu durumda kazanılan değer yönetimi metodu kullanılarak,

BAC = 100.000 TL

SAC = 40 hafta

PV = 50% * 100.000 TL = 50.000 TL

AC = 60.000 TL

EV = 40% * 100.000 TL = 40.000 TL

AT = 20 hafta

Sonuç olarak :

Tamamlanan Yüzde = EV / BAC = 40.000 TL / 100.000 TL = %40

Harcanan Yüzde = AC / BAC = 60.000 TL / 100.000 TL = %60

CV = EV – AC = 40.000 TL – 60.000 TL = -20.000 TL

SV = EV – PV = 40.000 TL – 50.000 TL = -10.000 TL

PV Oranı = BAC / SAC = 100.000 TL / 40 hafta = haftada 2.500 TL

TV = SV / PV Oranı = -10.000 TL / haftada 2.500 TL = -4 hafta

CPI = EV / AC = 40.000 TL / 60.000 TL = 0.67

SPI = EV / PV = 40.000 TL / 50.000 = 0.80

CR = CPI * SPI = 0.67 * 0.8* = 0.53 olmaktadır.

Çizelge 2.7 kazanılan değerle ilgili formüllerin özetini vermektedir.

Çizelge 2.7. Kazanılan Değer İle İlişkili Formüller

Maliyet-İlişkili Hesaplamalar	
Maliyet Değişkeni (“Cost Variance” – “CV”)	CV = BCWP - ACWP
CV %	CV% = (CV/BCWP)*100
Maliyet Performans Endeksi (“Cost Performance Index” – “CPI”)	CPI = BCWP / ACWP
Zamanlama-İlişkili Hesaplamalar	
Zamanlama Sapması (“Schedule Variance” – “SV”)	SV = BCWP - BCWS
SV%	SV% = (SV/BCWS) * 100
Zamanlama Performans Endeksi (“Schedule Performance Index” – “SPI”)	SPI = BCWP / BCWS
Diğer Hesaplamalar	
Harcanan Yüzde	Harcanan Yüzde = (ACWP/BAC) *100
Tamamlanan Yüzde	Tamamlanan Yüzde = (BCWP / BAC) * 100
Tamamlanacak Performans Endeksi (“To-Complete Performance Index” - “TCPI”)	TCPI = (BAC - BCWP) / (BAC - ACWP)
Tanımlar	
ACWP	Actual Cost of Work Performed
BAC	Budget at Completion
BCWP	Budgeted Cost of Work Performed
BCWS	Budgeted Cost of Work Scheduled
EAC	Estimate at Completion

2.6.2.2. Önerilen Modelin Kazanılan Değer Analizi İçin Kullanılması

Kazanılan değer, diğer alanlarda olduğu gibi yazılım projelerinde de planlama aşamasında ve proje süresi boyunca mevcut durumu değerlendirmek, planlanan değerin dışına çıkıldığında gerekli aksiyonları almak amacıyla yöneticiler tarafından sıklıkla kullanılmaktadır.

Giriş kısmında bahsedilen model, işlevsel yazılım büyüklüğü ölçümümün otomatikleştirilmesini hedeflediğinden bu ölçüm, projenin geliştirilmesinin ilk safhalarından, proje bakım süresi boyunca kullanılabilir. Önerilen modelin, tekrarlı yazılım geliştirme sürecinde Sprint başlarında ve sonlarında hedeflenen kullanım durumlarının büyüklüğü ölçülerek bir tahmin yapılması ve durum değerlendirilmesinde bulunulmasına katkısı olacaktır. İlgili projenin planlanan değer (“PV”), kazanılan değer (“EV”) ve gerçek maliyet (“AC”) değerleri hesaplanarak bu kapsamda bir çıkarım elde edilebilir. Proje tamamlandığında hedeflenen maliyet (tamamlama maliyeti – “BAC”) ile gerçek maliyet arasında karşılaştırma yapılabilir.

Örneğin, çevik yazılım geliştirme yapan bir yazılım projesinin planlanan bütçesinin 100.000 TL olduğunu ve proje süresinin 40 hafta olduğunu, planlanan işlevsel büyüklüğün ise 2000 olduğunu varsayalım. İşlerin her bir Sprint başında planlandığını düşünelim. Sprintlerin 2 hafta olacağı düşünülürse, proje kapsamında bu 2 haftalık zamanın sahip olduğu bir bütçe miktarı ve bitirilmesi gereken iş miktarı bulunmaktadır. Sprint bitiminde yapılan işler önerilen otomatik işlevsel büyüklük ölçme modeli ile yapılırsa, takvimin ne kadar ilerisinde veya gerisinde olunduğu somut bir şekilde anlaşılabilir. Eğer 2 haftalık bir Sprint için 100 işlev puanına sahip bir iş planlandıysa, ancak Sprint sonunda ölçülen işlevsel büyüklük 80 ise, takvimin gerisinde kalındığını gösterir. Bu noktada proje yöneticisi/takım liderleri, ilgili sapmanın sebebine odaklanabilecek ve gerekli aksiyonları alabilecektir. Böylece gecikme riski olan durumlar engellenecektir.

İşlevsel büyüklüğü hesaba katmadan, sadece proje bütçesi üzerinden değerlendirme yapıldığı durumda ise, Sprint sonunda 80 işlev puanlık işin tamanlandığı durumda proje bütçesi üzerinden 10.000 TL harcandığını düşünelim. Eğer maliyet üzerinden değerlendirme yapıyor olsaydık, harcanan bütçenin toplam bütçe üzerindeki değerine bakarak projenin hangi oranda tamamlandığı bilgisini tahmin edebilirdik. Bu durumda projenin 10.000/100.000 = %10 oranında

tamanlandığı hesaplanacaktı. Ancak işlev puan üzerinden gidildiğinde, iki hafa sonunda uygulamadan beklenen işlevselliğin 80/2000 = %4 oranında karşılanabildiği hesaplanmaktadır. Bu durum, 2 haftalık Sprint sonunda proje maliyetinin aşıldığını göstermektedir. Bu sebeple, işlevsel büyüklüğün otomatik olarak hesaplanabilmesi, projenin cari durumunun değerlendirilebilmesi ve gerekli aksiyonların alınabilmesi için oldukça önemlidir.

Yukarıda örneklerle anlatılan senaryolar, işlevsel büyüklüğün kazanılan değer ve proje yönetimi açısından önemini bir kez daha vurgulamaktadır.

3. COSMIC İŞLEVSEL BÜYÜKLÜK ÖLÇÜMÜ

COSMIC, yazılımların büyüklüğünü ölçmede yaygın olarak kullanılan ve ISO tarafından onaylanmış uluslararası bir standarttır. COSMIC işlevsel büyüklük ölçme yöntemi aşağıdaki alanlarda yazılımın işlevselliğinin belirlenebilmesi için tasarlanmıştır [15]:

- *Veri-zengin yazılımlar:* Bankacılık, sigortacılık, muhasebe, satın alma, insan kaynakları, dağıtım ya da üretim gibi iş yönetiminin desteklenmesi için gereken iş uygulaması yazılımları. Bu tip yazılımlar gerçek dünyadaki olaylara ilişkin çok miktardaki verinin yönetimini gerektirdiği için genellikle veri-zengin yazılımlar olarak nitelendirilmektedir.
- *Gerçek-zamanlı yazılımlar:* Gerçek dünyada gerçekleşen olaylara ayak uydurmayı ya da onları kontrol etmeyi gerektiren gerçek-zamanlı yazılımlar. Telefon santrali ve mesaj anahtarlama yazılımları; ev aletleri, asansörler, araba motorları ve uçak gibi makineleri kontrol eden cihazlara gömülü yazılımlar; süreç kontrolü ve otomatik veri toplama için kullanılan yazılımlar ve bilgisayarların işletim sistemlerinin içerisinde bulunan yazılımlar bu tür yazılımlara örnek olarak verilebilir.
- *Melez yazılımlar:* Yukarıda yer alan yazılımların karışımı olan yazılımlardır. Havayolları ya da oteller için geliştirilen gerçek zamanlı rezervasyon sistemleri bu tür yazılımlara örnek olarak verilebilir.

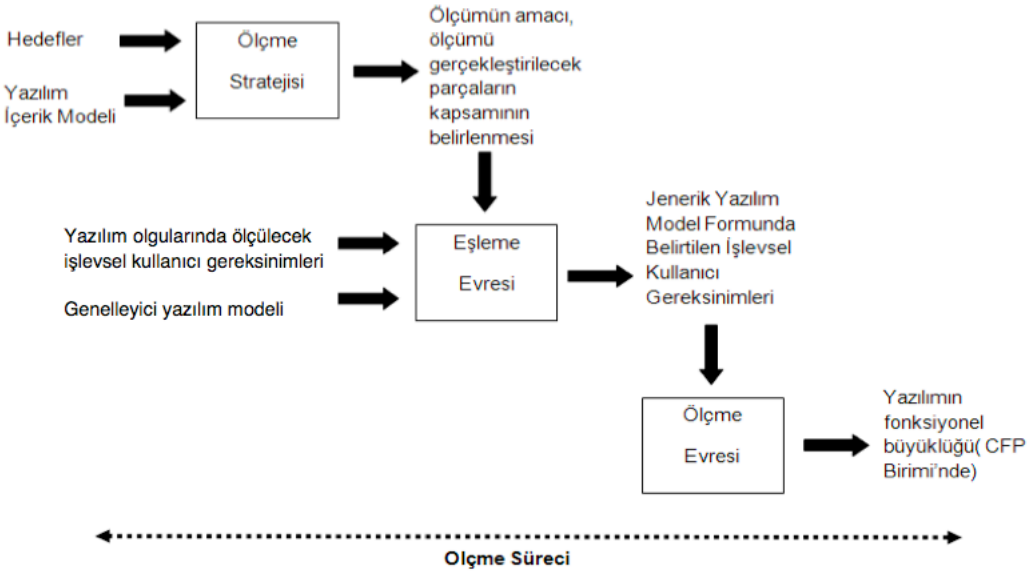
Tez çalışmasında COSMIC işlevsel büyüklük metodu kullanılmıştır. Bu metodun seçilmesinde, ISO tarafından tanınan uluslararası bir standart olması, kolay uygulanabilirliği, çeşitli yazılım uygulamalarında kullanılabilirliği, yazılımın büyüklüğünü birden fazla bakış açısından ölçebilmesi ve COSMIC kavramlarının modern yazılım mühendisliği kavramlarıyla örtüşmesi gibi özellikler etkili olmuştur. COSMIC ölçme yöntemi, ölçümü yapılacak yazılım parçasının İşlevsel Kullanıcı Gereksinimlerine bir dizi modelin, ilkenin, kuralın ve sürecin uygulanmasını içerir. Elde edilen sonuç, COSMIC yöntemine göre yazılımın işlevsel büyüklüğünün miktarının sayısal değerini ifade eder.

İşlevsel Kullanıcı Gereksinimleri, yazılım geliştirilmeden önce yazılım mühendisliği ürünlerinden elde edilebilir. Bu nedenle yazılımın işlevsel büyüklüğü, yazılım geliştirilmeden önce de ölçülebilir [15].

COSMIC ölçme süreci üç evreden oluşmaktadır:

- Ölçme Stratejisi Evresi: Yazılım Bağlam Modelinin ölçülecek yazılıma uygulandığı evredir.
- Eşleme Evresi: Genel Yazılım Modelinin ölçülecek yazılıma uygulandığı evredir.
- Ölçme Evresi: Gerçek büyüklük ölçüm sonuçlarının elde edildiği evredir.

COSMIC yönteminin bu üç evresi arasındaki ilişkiler Şekil 3.1'de gösterilmiştir.



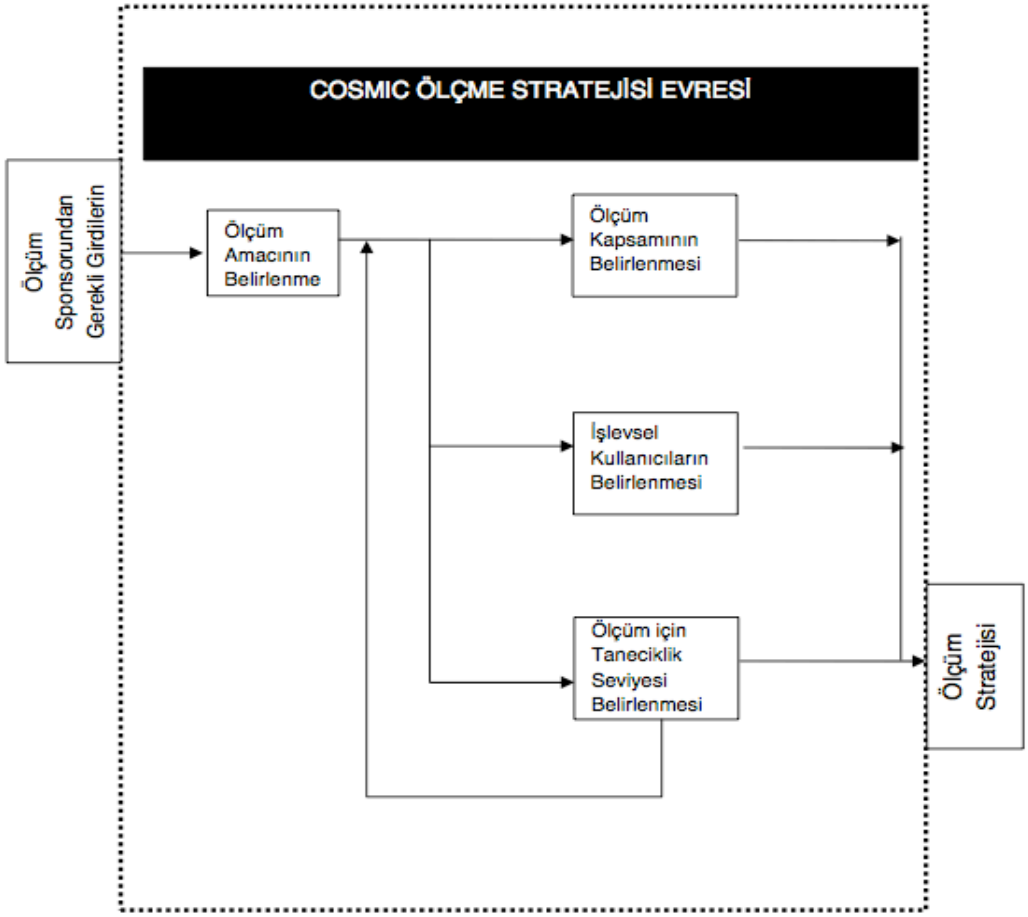
Şekil 3.1. COSMIC Ölçme Süreci Evreleri Arasındaki İlişkiler [15]

3.1. Ölçme Stratejisi Evresi ("Measurement Strategy Phase")

Yapılacak ölçme işleminin amacı, kapsamı, ölçme işleminin gerçekleştirileceği sistemin işlevsel kullanıcıları ve ölçmenin gerçekleştirileceği taneciklik seviyesi bu evrede belirlenir.

Herhangi bir ölçüm sonucunu kaydederken, Ölçme Stratejisi evresinden gelen verinin kaydedilmesi önemlidir. Bu parametrelerin tanımlanması ve kaydedilmesindeki başarısızlık, güvenilir şekilde anlaşılmayan ve

karşılaştırılamayan ya da proje işgücü kestirimi gibi süreçlere girdi olarak kullanılamayan ölçümlere, sürekli olarak sebebiyet verecektir [15]. Şekil 3.2 Ölçme Stratejisi Evresini göstermektedir.



Şekil 3.2. Ölçme Stratejisi Evresi [15]

Bu evreyle ilgili ilişkili tanımlar aşağıda açıklanmıştır.

Ölçme Amacı: Ölçmeye neden ihtiyaç duyulduğunu, sonucun ne için kullanılacağını tanımlayan ifadedir. Yazılımın işlevsel büyüklüğünün geliştirmeden önce ölçülmesi, yazılımın planlarından yani geliştirmeden önce oluşturulan ürünlerinden türetilen, yazılımın İşlevsel Kullanıcı Gereksinimlerine dayanır. İşlevsel kullanıcı gereksinimleri bu ürünlerden, uygun kurallar kullanılarak türetilir ve büyüklüğün ölçülebilmesi için gerekli boyutlar (veri hareketi sayısı) belirlenir.

Ölçme Kapsamı: Bir işlevsel büyüklük ölçmede içerilecek İşlevsel Kullanıcı Gereksinimleri kümesidir.

İşlevsel Kullanıcı Gereksinimleri: Kullanıcı gereksinimlerinin alt kümesidir. Yazılımın iş ve hizmet cinsinden ne yapması gerektiğini tanımlayan gereksinimleri

ifade eder. İşlevsel Kullanıcı Gereksinimleri, veri transferi, veri dönüşümü, veri saklama, veri erişimi kavramlarını içerir.

Ayrıştırma Seviyesi: Bir yazılım parçasının bileşenlerine bölünmesi (‘Seviye 1’ gibi), bileşenlerin alt bileşenlere bölünmesi (‘Seviye 2’), daha sonra alt bileşenlerin alt-alt bileşenlere bölünmesi (‘Seviye 3’) vb. ile elde edilen her seviyedir.

Eş: İki yazılım parçası aynı katmanda bulunuyorsa birbirlerinin eşidir.

Eş Bileşen: Karşılıklı beraber çalışan, aynı ayrıştırma seviyesinde bulunan, bir yazılım parçasının bir katman içinde bölünmesinden oluşan ve yazılım parçasına ait İKG’nin belirli bir kısmını karşılayan bileşenlerden her biridir.

İşlevsel Kullanıcı: Bir yazılım parçasının İşlevsel Kullanıcı Gereksinimleri verisinin göndereni yada alıcısı olan kullanıcıdır. Bir iş uygulaması düşünüldüğünde işlevsel kullanıcılar, normalde insan ve arayüzü bulunan diğer eş bileşen uygulamalarından oluşacaktır. Bu şekildeki bir yazılımın işlevsel kullanıcı gereksinimleri, normalde işlevsel kullanıcılarının yazılıma veri göndereceği ya da yazılımdan veri alacağı şekilde ifade edilir. Unutmamak gerekir ki konu yazılım olduğunda, farklı işlevsel kullanıcılar (kullanıcı tipleri) farklı işlevselliğe ihtiyaç duyabileceği için, işlevsel büyüklükler işlevsel kullanıcı seçimine göre değişecektir.

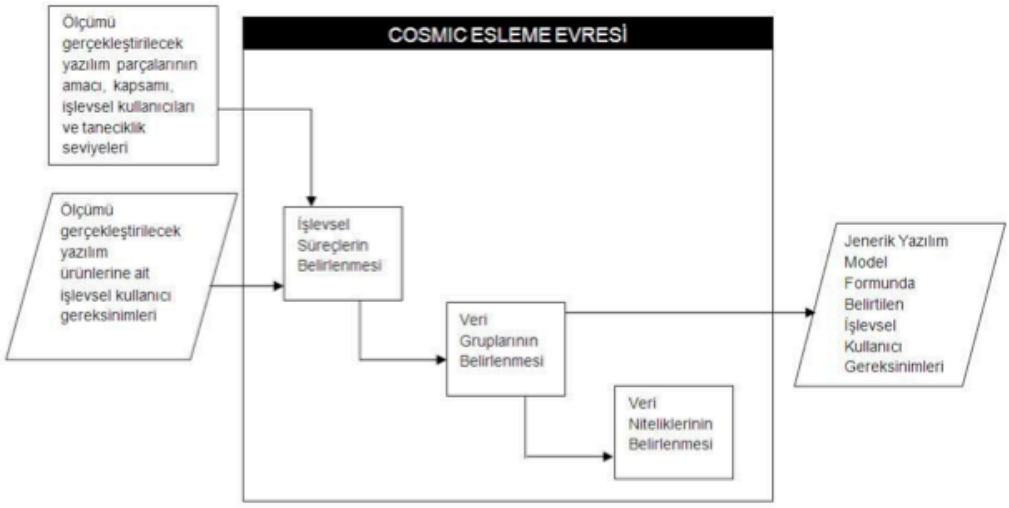
Sınır: Ölçülen yazılım ile işlevsel kullanıcılarının arasında, kavramsal bir arayüz olarak tanımlanır.

3.2. Eşleme Evresi (“Mapping Phase”)

Bu evrede işlevsel süreçler saptanır ve bu süreçlerle ilgili veri grupları ve veri özellikleri belirlenir. Şekil 3.3 Eşleme Evresini göstermektedir.

Bu evreyle ilgili ilişkili tanımlar aşağıda açıklanmıştır.

İşlevsel Süreç: Bir işlevsel süreç benzersiz, kohesif ve bağımsız olarak çalıştırılabilen bir set, veri hareketini içeren bir set, İşlevsel Kullanıcı Gereksiniminin temel bileşenidir. Yazılım parçasının işlevsel olay belirlediği bilgisini veren işlevsel kullanıcılardan gelen veri hareketi (bir Giriş) ile tetiklenir. İşlevsel olaya karşılık olarak gereken işlemler yapıldığında tamamlanır. Bir işlevsel süreç en azından iki veri hareketi içermelidir (bir Giriş ve bir Çıkış veya Yazma.)



Şekil 3.3. Eşleme Evresi Süreci [15]

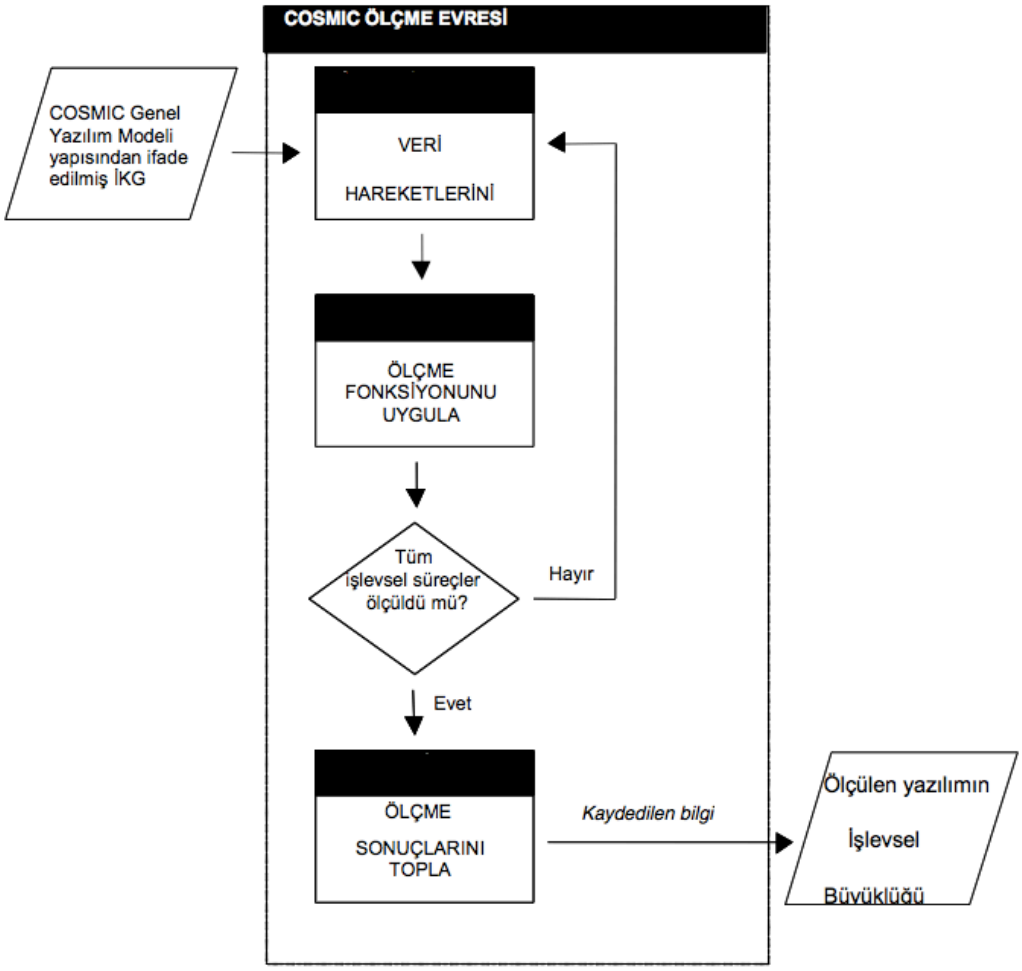
Tetikleyici Olay: Bir yazılım parçasının bir işlevsel kullanıcısının bir veya birden fazla işlevsel süreç başlatmasına (‘tetikleme’) yol açan bir olay (gerçekleşen bir şey). Bir olay bir işlevsel kullanıcı tarafından farkedilir ve işlevsel kullanıcı işlevsel bir süreci tetikler. Bir tetikleyici olay, işlevsel kullanıcı ve bir işlevsel süreci tetikleyen Giriş verisi arasındaki ilişki Şekil 3.4’te verilmektedir.

Şekil 3.4. Tetikleyici Olay, İşlevsel Kullanıcı ve İşlevsel Süreç Arasındaki İlişki [15]

Veri Grubu: Bir veri grubu her bir veri özniteliğinin aynı ilgi nesnesinin tamamlayıcı özelliklerini tanımladığı farklı, boş olmayan, sıralı olmayan ve tekrarlamayan bir veri öznitelikleri setidir.

3.3. Ölçme Evresi (“Measurement Phase”)

Bu evrede işlevsel süreçler için veri hareketleri belirlenmektedir. Ölçme fonksiyonu uygulandıktan sonra sonuçlar toplanarak yazılımın işlevsel büyüklüğü elde edilmiş olur. Bir yazılım parçasının işlevsel büyüklüğünün nasıl ölçüleceği Şekil 3.5’te özetlenmektedir.



Şekil 3.5. Ölçme Evresi Süreci [15]

Veri hareketi: Tek bir veri grubu tipini hareket ettiren temel işlevsel bileşendir. Veri hareketi tipinin dört çeşit alt tipi vardır:

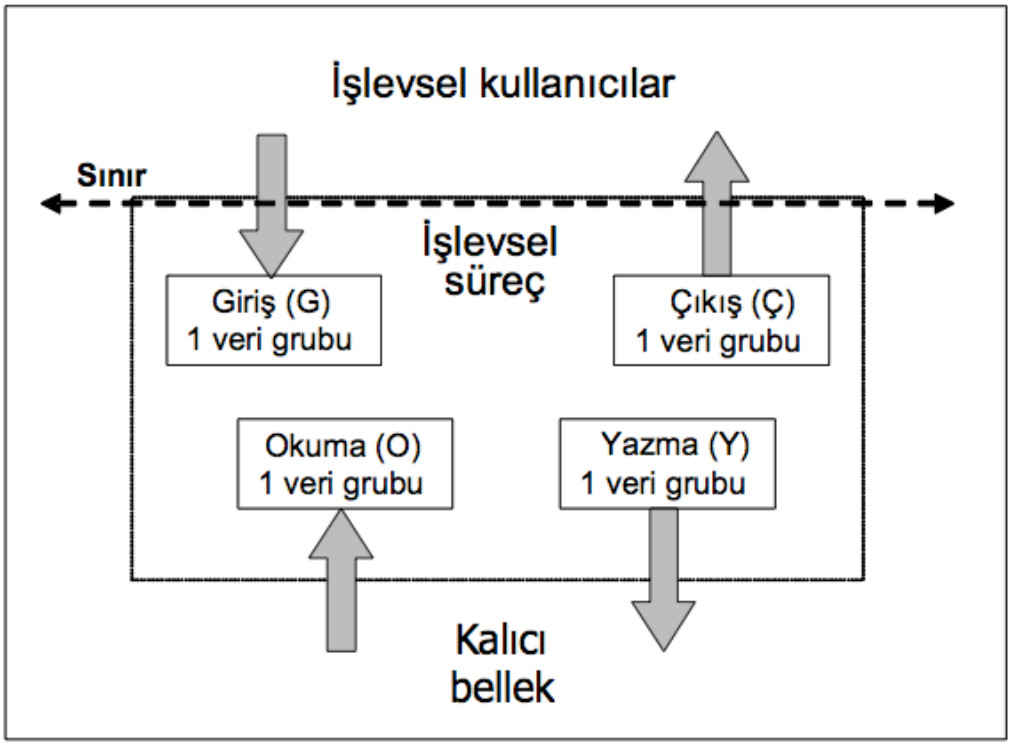
Giriş (“Entry”), Okuma (“Read”), Yazma (“Write”) ve Çıkış (“Exit”)

Giriş, bir veri grubunu bir işlevsel kullanıcıdan uygulama sınırından içeriye ihtiyaç duyulan bir işlevsel sürece doğru hareket ettiren veri hareketidir. Bir işlevsel sürecin girdisi birden çok veri grubu içeriyor ise, girdideki her bir benzersiz veri grubu için bir Giriş tanımlanır.

Okuma, bir veri grubunu kalıcı bellekten ihtiyaç duyulan işlevsel sürece doğru hareket ettiren bir veri hareketidir. Bir işlevsel süreç kalıcı bellekten birden çok veri grubu getirmek durumunda ise, getirilen her bir benzersiz veri grubu için bir Okuma tanımlanır.

Yazma, bir işlevsel süreçte yer alan veri grubunu kalıcı belleğe doğru hareket ettiren veri hareketidir. Bir işlevsel süreç, kalıcı belleğe birden çok veri grubu hareket ettirmek durumunda ise, kalıcı belleğe hareket ettirilen her bir benzersiz veri grubu için bir Yazma tanımlanır.

Çıkış, bir veri grubunu bir işlevsel süreçten uygulama sınırından dışarıya ihtiyaç duyan bir işlevsel kullanıcıya doğru hareket ettiren veri hareketidir. Eğer bir işlevsel süreçten çıktılar birden fazla veri grubundan oluşuyorsa, çıktıdaki benzersiz her bir veri grubu için bir Çıkış belirlenir.



Şekil 3.6. Dört Veri Hareketi Tipi ve Bunların İşlevsel Süreç ve Veri Grupları ile İlişkileri [15]

Şekil 3.6 dört veri hareketi tipi arasındaki genel ilişkiyi, ait oldukları işlevsel süreci ve ölçülen yazılımın sınırını göstermektedir.

Burada dikkat edilmesi nokta, tekrarlanan veri hareketi tipi oluşumları (yani aynı veri işlemelerine sahip aynı veri grubunun hareketi) herhangi bir işlevsel süreçte birden çok kez belirlenmemeli ve sayılmamalıdır.

Ölçüm fonksiyonu: COSMIC ölçüm fonksiyonu, COSMIC ölçüm standardına dayanarak fonksiyon değişkenlerine bir değer atayan matematiksel bir fonksiyondur. COSMIC ölçüm fonksiyonunun değişkeni veri hareketidir.

Ölçüm standartı: COSMIC ölçüm standardı, 1 CFP (COSMIC İşlev Puan), bir veri hareketinin büyüklüğü olarak tanımlanmıştır.

Ölçüm sonuçlarının toplanması: Herhangi bir işlevsel süreç için, ayrı veri hareketlerinin işlevsel büyüklükleri aritmetik olarak eklenerek CFP birimi olarak toplanmalıdır.

Büyüklük (işlevsel süreç) =
Σ büyüklük(Giriş$_i$) + Σ büyüklük(Çıkış$_i$) + Σ büyüklük(Okuma$_i$)+Σ büyüklük(Yazma$_i$)

4. İLİŞKİLİ ÇALIŞMALAR

Literatürde, işlevsel büyüklük kavramıyla ilgili birçok çalışma yer almaktadır. COSMIC işlevsel büyüklük hesaplamaya değinen çalışmaların sayısının fazla olmasıyla birlikte, ölçümümün otomatikleştirilmesi hakkında yeterli seviyede çalışma bulunmamaktadır. Bu bölümde, incelenen çalışmalardan, COSMIC işlevsel büyüklük hesaplama alanındaki çalışmalara değinilecektir.

Bevo ve ark. [19] COSMIC işlevsel büyüklük ölçme metotu ve UML (Unified Modeling Language) arasında eşleme yapmaya çalışmışlardır. Çalışma işlevsel büyüklük ölçümünü, kullanım durumu diyagramlarından yola çıkarak hesaplamayı amaçlamıştır. Bu varsayımda, her bir kullanım durumu diyagramı bir işlevsel sürece denk gelmekle birlikte, kullanım durumu diyagramlarındaki her bir etkileşim bir veri hareketine denk gelmektedir. Kullanım durumu diyagramlarındaki her bir aktör de işlevsel kullanıcıyı ifade etmektedir. Eşleme sonuçlarının başarı yüzdesini araştırmak için bazı çalışmalar yapmışlardır. Bu çalışmalarda Metric Xpert aracı kullanılmış ve ölçme sonuçları uzmanlar tarafından elde edilerek, sonuçlar karşılaştırılıp değerlendirilmiştir.

Poels çalışmasında [49] kurumsal bilgi sistemleri geliştirmesinde (örn: olay tabanlı yaklaşım) COSMIC metotunu uygulamak için bazı kurallar tanımlamıştır. Tetikleyici olay fikri, olay tabanlı yaklaşım ve COSMIC metodunun doğal eşlemesiyle geldiğinden, çalışma UML ve olay tabanlı olan MERODE metodolojisini kullanmaktadır.

Diab ve ark. [50] yaptıkları çalışmada, Rational Rose RealTime (RRRT) modelleri için otomatik olarak COSMIC-FFP işlevsel büyüklüğü ölçen aracı (µcROSE) anlatmaktadır. µcROSE, COSMIC-FFP ölçüm sürecini iki yönde geliştirmektedir. İlk olarak, araç ölçüm maliyetini ve yüksek ölçüm eğitimini (“advanced measurement training”) ayrıştırmaktadır. İkinci olarak da, µcROSE ölçüm varyansını kaldırmakta ve mükemmel tekrarlanabilirliği sağlamaktadır. Amaç, RRRT notasyonunda yazılan herhangi bir büyüklükteki yazılım gereksinimlerini alarak işlevsel büyüklüğü hesaplamaktadır.

Levesque ve ark. [22] kullanım durumu diyagramlarından ve dizi diyagramlarından sistemin işlevsel büyüklüğünü ölçmek için COSMIC metotunu kullanmışlardır. İşlevsel süreçleri, veri hareketi ve manipülasyon türleri olmak üzere ikiye

ayırmışlardır. UML mesaj değişimini düşünerek ve aktör-obje akış diyagramlarını kullanarak COSMIC işlevsel büyüklüğün tümdengelimine yoğunlaşmışlardır.

Ferrucci ve ark. [51] Web-COBRA'nın COSMIC ile birleşerek web uygulamaları geliştirme maliyeti tahmininde bulunabileceğini savunmaktadırlar. Web-COBRA, COBRA metodunun uyarlanmasıyla oluşturulmuştur. COBRA ise, Model ve Non-Model tabanlı metotların birleşmesini öneren tipik bir karma (composite) metottur. Web-COBRA ise web uygulamalarında gereksinimleri göz önüne alarak geliştirme maliyetini tahmin etmeyi öneren bir metottur. Çalışmada kullanılan veriler 15 web uygulamasından alınmıştır. Web-COBRA yı uygulamak için maliyet faktörleri ve miktarları belirlenmiş, daha sonra da web uygulamalarından veriler toplanmıştır.

Lavazza ve ark. [21] COSMIC tabanlı işlevsel büyüklük ölçümü için, UML modellerini COSMIC ile birlikte kullanmışlardır. Örnek uygulama olarak 'Rice Cooker Revisited' kullananan yazarlar, UML bağlamı ile ölçüm kurallarını uygulanabilir yaparak, UML kullanmanın nasıl COSMIC ölçüm pratiğini geliştirdiğini göstermektedirler.

Bir diğer çalışmada [52] Hussain ve ark. çevik süreçlerde kullanılabilecek, metinsel gereksinimlerden yola çıkarak COSMIC işlevsel büyüklük kestirim yaklaşımını önermektedirler.

Lind ve Heldal [53] manuel olarak hesaplanan kod büyüklüğü tahmininin nasıl minimize edileceğini sorusuna cevap aramaktadırlar. Çalışmada kod büyüklüğü tahmini için UML kullanarak gerekli bilgileri yakalayan UML Profile anlatılmaktadır. Makalede ayrıca COSMIC ana kavramları ve UML Profile arasındaki eşleşme ("mapping") kuralları da anlatılmaktadır.

Fehlmann ve ark. [20] UML diyagramlarını kullanan çevik geliştirme ekiplerinden etkilenmişlerdir. Önerilerinde COSMIC işlev puanını hızlı ve kolay yoldan hesaplamak için, UML sıra diyagramı ("sequence diagram") kullandıklarını ifade etmişlerdir. Bu çalışmada ilginç olan ise, sıra diyagramı bileşenlerinin COSMIC ölçüm yapısı perspektifinden tanımlanmış olmasıdır.

Bianco ve Lavazza [54] yaptıkları çalışmada COSMIC ölçüm metodunun, kullanıcı gereksinimlerine uygulanma olasılığını araştırmaktadır. Örnek olarak tehlikeli mallar taşıma izleme sistemi ölçülmüştür. Sonuç olarak, Problem Frame tekniği

ve COSMIC FSM metodunun ölçümde yararlı olduğu belirtilmiştir. UML kullanımının ise ölçüm maliyetini düşürdüğü ifade edilmiştir.

Buglione ve ark. [55] ürün büyüklüğünün erken aşamada tahmin edilebilme olasılığını incelemişlerdir. Yazarlar COSMIC işlevsel büyüklük birimleri ile ölçülen ürün (FUR ve NFR) büyüklüğü tahmini için Proje Büyüklük Birimi ("Project Size Unit – PSU") tekniğini önermişlerdir. Örnek uygulama olarak sınav yönetim sistemi kullanılmıştır. Sonuç olarak, FUR ve NFR işlevsel büyüklük ölçümünün yazılım planlama aşamasının öncesinde tahmin edilebileceği belirtilmiştir.

Akça ve Tarhan [23] ölçüm kütüphanesi geliştirmiş ve kaynak koda ilgili kütüphanenin metotlarını ekleyerek yarı-otomatik olarak COSMIC işlev puanını elde edebildiklerini ifade etmişlerdir. Ölçme koduna eğitimli bir yazılımcı tarafından manuel olarak eklemeler yapıldıktan sonra, hesaplama işlemi çalışma zamanında kullanıcı senaryolarına bağlı olarak yapılmaktadır. Sonraki çalışmalarında [17] ise, yazarlar geliştirdikleri metodun manuel olarak yapılan ölçmeye nazaran %84 oranında bir maliyet kazancı sağladığını belirtmişlerdir.

Sağ ve Tarhan [24] AspectJ teknolojisi ile çalışma zamanı sırasında yakalanan UML sequence diagramlarından yola çıkılarak COSMIC işlevsel büyüklüğü otomatik olarak hesaplamaktadırlar. Hesaplama işleminin iki ana bileşeni bulunmaktadır. Bunlar Tracer ve Cosmic Calculator dır. Tracer bileşeni, işlevsel süreçler gerçekleşirken sınıflar arasındaki etkileşimleri yakalamaktadır. Cosmic Calculator bileşeni ise UML sequence diagramlar üzerine COSMIC ölçüm kurallarını uygulayarak işlevsel büyüklüğü hesaplamaktadır. 'Cosmic Solver' diye adlandırılan araç, Java Swing, JPA ve JDBC teknolojilerinde çalışmaktadır. Otomatik gerçekleşen hesaplamanın, manuel ölçümle büyük ölçüde uyuştuğu belirtilmiştir.

Bu çalışma kapsamında CFP'nin otomatik ölçülmesine yoğunlaşılmıştır. Önerilen yöntemde kaynak koddaki ayrıştırma işlemi [17] deki yaklaşıma benzer iken, kullanıcı senaryolarına bağlı olarak çalışma zamanında işlevsel büyüklük hesaplama işlemi [21] [23] [24] lerdeki çalışmalara benzemektedir. Çalışma, [23]'deki koda manuel olarak ekleme kısıtını ortadan kaldırarak koda aşina olmayan biri tarafından bile kolay bir şekilde otomatik olarak COSMIC işlevsel büyüklüğün hesaplanmasına olanak sağlamaktadır.

5. OTOMATİK ÖLÇME YÖNTEMİ

Tez çalışması kapsamında önerilen yöntemin temel amacı COSMIC büyüklük hesaplama işleminin Java iş uygulamaları için otomatik hale getirilmesidir. Bu bölümde, önerilen yöntem ve ölçme işleminin nasıl gerçekleştirileceğine dair bilgiler verilecektir.

Otomatik ölçme yöntemini geliştirmeye başlamadan önce COSMIC işlevsel büyüklük ölçümü kullanımını anlamak amacıyla bir anket hazırlanmıştır. Hazırlanan anket genel olarak farkındalık, ölçüm maliyeti ve otomasyon konularına yoğunlaşmıştır. Örnek uygulamayı geliştiren ekip üzerinde uygulanan ankette, katılımcıların büyük çoğunluğu COSMIC kavramı hakkında çok fazla bir bilgisi olmayan yazılımcılardan oluşmaktadır. Katılımcılar arasında ayrıca yöneticiler, test mühendisleri, veri tabanı sistemi mühendisleri, iş analistleri ve kalite güvence mühendisleri bulunmaktadır.

Ankette esas olarak şu araştırma sorularına (AS) cevap aranmıştır:

AS-1 : Kurumunuzdaki yazılım büyüklüğü ölçümü kullanımının ve farkındalığının seviyesi ne ölçüdedir?

AS-2 : Kurumunuzda yazılım büyüklüğü ölçümünde maliyet ve otomasyon etkili faktörler midir?

Anketi 24 kişiden 21'i cevaplamıştır. EK 2 anket sorularını ve cevaplarını açıklamaktadır. Sonuçlar, COSMIC işlevsel büyüklük metotudun, yazımların büyüklüğünü ölçmede uygun bir metot olduğunu göstermektedir. Ayrıca COSMIC metotudunun kullanılmasında ölçme işleminin otomatikleştirilmesinin yarar sağlacağı sonucu, bir diğer elde edilen neticedir.

Geliştirme aşamasına gelindiğinde, COSMIC işlevsel büyüklüğün otomatik olarak hesaplanması için "Measurement" Kütüphanesi ve Ayrıştırma İşlemi tasarlanmıştır. "Measurement" Kütüphanesi, işlevsel süreçleri tespit ettikten sonra toplam COSMIC İşlev Puanını hesaplamak için dizayn edilmiştir. Ayrıştırma İşlemi ise kaynak kodu belirli kurallar dahilinde ayrıştırarak, ilgili yerlere oluşturulan kütüphane metotlarını otomatik olarak eklemeyi sağlar. Ayrıştırma işleminden sonra Java uygulaması çalıştırılır ve kullanıcı senaryoları koşulurken COSMIC İşlev Puanı otomatik olarak hesaplanmış olur.

Önerilen modelde işlevsel kullanıcı; kullanıcı senaryolarını grafik kullanıcı arayüzünden tetikleyen bir kişi, makine ya da başka bir uygulama olabilir. Ölçmenin kapsamı, seçilen bir modül ya da uygulamanın tamamı olabilir.

Önerilen modelde, uygulamanın kalite isterlerinin büyüklüğe olan etkisi hesaplamada dikkate alınmamıştır.

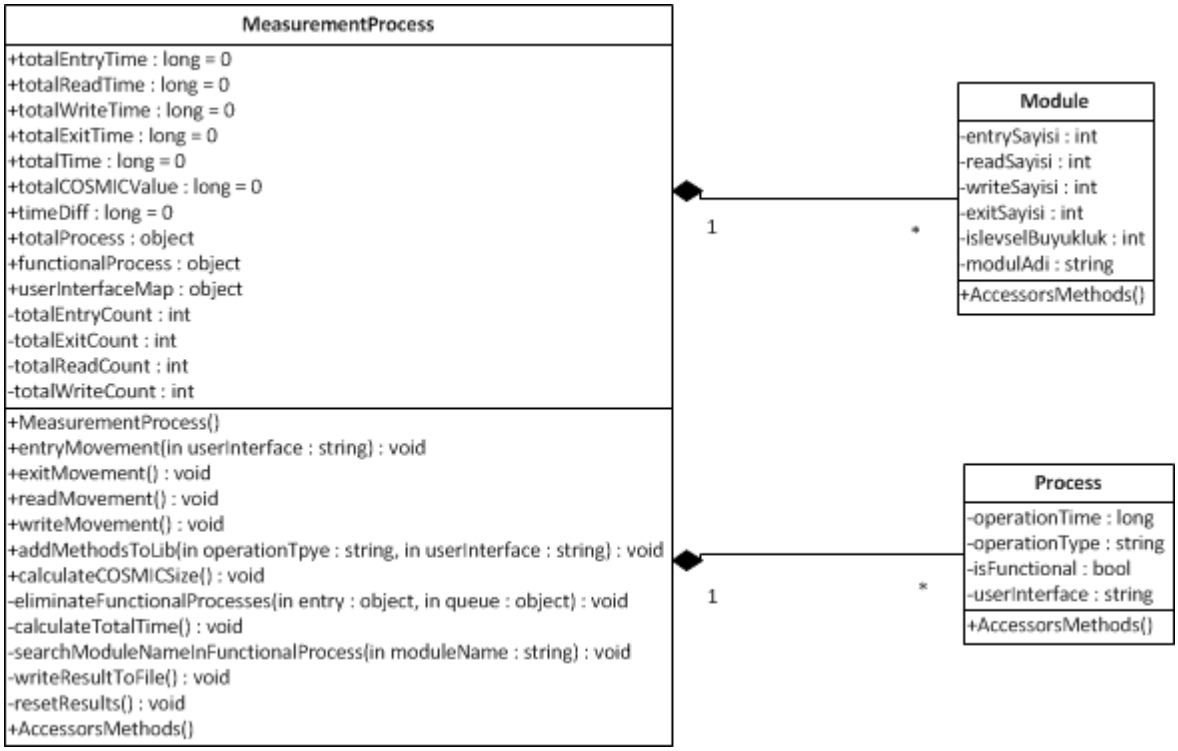
5.1. Ölçme Yöntemi

Önerilen modelde otomatik ölçme yöntemi, Ölçme Kütüphanesi ve Ayrıştırma İşlemi olarak iki bileşenden oluşmaktadır. Ölçüm yapanın yapması gereken işlem, .jar şeklinde oluşturulan ölçüm kütüphanesini ilk sefer için proje ortamına tanıtmak ve sonrasında Ayrıştırma İşlemi kodunu, projenin bulunduğu dizini ve proje jar adlarını vererek çalıştırmaktır. Kütüphanenin projeye tanıtılması işlemi tek sefere mahsustur. Bundan sonra yapılacak ölçme işlemlerinde sadece yeni eklenen modül/kod vs. için Ayrıştırma İşlemi kodunun çalıştırılması ve çalışma zamanında ölçülmek istenilen kullanım durumlarının koşulması yeterlidir. Ölçme işlemini yapan kullanıcının takip etmesi gereken süreç adımları Şekil 5.1'de gösterilmektedir.

Şekil 5.1. Kullanıcı Bakış Açısıyla Ölçme İşleminin Adımları

5.2. Ölçme Kütüphanesi

Measurement Kütüphanesi, COSMIC veri hareketi alt tipleri için bildirim oluşturmak için metotlar içermektedir. Kütüphane, işlevsel süreçleri ayırt etmek ve işlevsel süreçler üzerinden COSMIC İşlev Puanını hesaplamak için veri hareketi bildirimlerini kayıt altına alır. Measurement Kütüphanesi, COSMIC Ölçüm Kılavuzunda [15] verilen bilgiler doğrultusunda oluşturulmuştur. Şekil 5.2 kütüphanenin UML gösterimini ifade etmektedir.



Şekil 5.2. Measurement Kütüphanesinin UML İle Gösterimi

Hesaplama için kritik olan bu metotlardan bazıları aşağıda açıklanmıştır:

entryMovement (String userInterface): GUI üzerinde kullanıcı senaryolarının tetiklenmesi sonucu Giriş (“Entry”) hareketlerinin algılanması ve kaydedilmesi için tasarlanmıştır. Modül bazında hesaplama yapabilmek için metot, parametre olarak kaynak koddaki ilgili sınıfın bulunduğu paket adının son değerini alır. Çağırma zamanı (sistemin o anki saati), veri hareketi tipi ve GUI bilgisi, bu metot içerisinde saklanır.

readMovement (): Veri tabanı sorguları tarafından tetiklenen Okuma (“Read”) hareketlerinin algılanması ve kaydedilmesi için tasarlanmıştır. Çağırma zamanı (sistemin o anki saati) ve veri hareketi tipi, bu metot içerisinde saklanır.

writeMovement (): Veri tabanı güncellemeleri tarafından tetiklenen Yazma (“Write”) hareketlerinin algılanması ve kaydedilmesi için tasarlanmıştır. Çağırma zamanı (sistemin o anki saati) ve veri hareketi tipi, bu metot içerisinde saklanır.

exitMovement (): Arayüzde olan herhangi bir değişikliğin olması sonucu tetiklenen Çıkış (“Exit”) hareketlerinin algılanması ve kaydedilmesi için tasarlanmıştır. Çağırma zamanı (sistemin o anki saati) ve veri hareketi tipi, bu metot içerisinde saklanır. Bilgi/Hata mesajları, sorgu sonucu meydana gelen veri tablosu verileri,

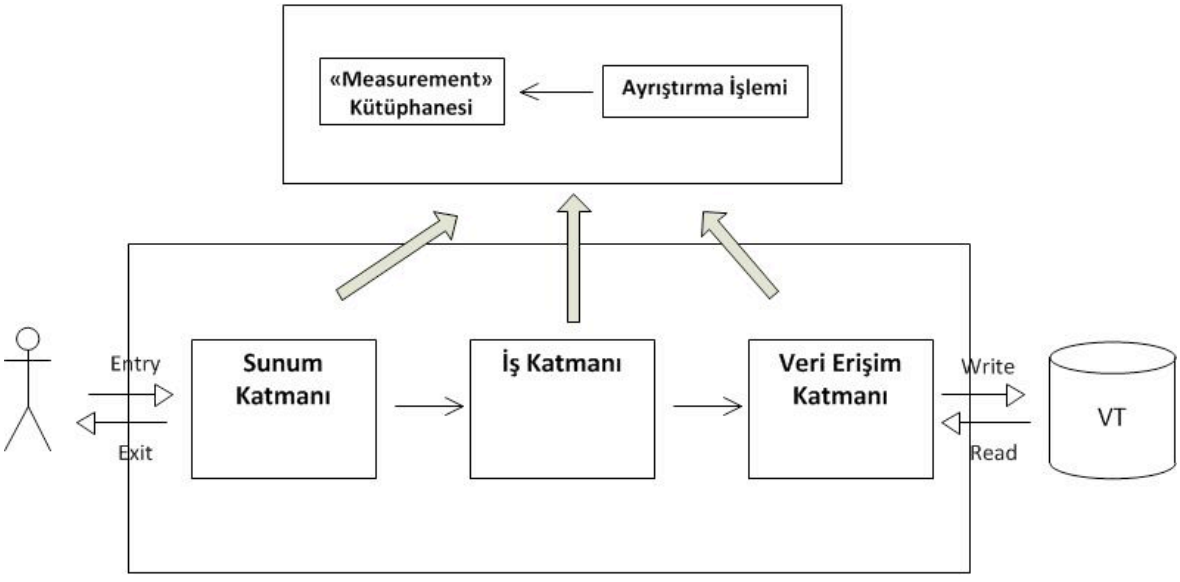
kullanıcıya dönen veri tabanı sorgu sonuçları örneklerden bazılarıdır.

calculateCOSMICSize (): Yukarıda belirtilen metotlarda kaydedilen veri hareketlerinden, işlevsel süreç olanları ayırt ederek COSMIC İşlev Puanını hesaplar. Uygulama sona erdiğinde (çıkış yapıldığında) bu metot çağrılır. İşlevsel süreçler hesaplanırken, veri hareketlerinin sırası dikkate alınır. En az bir "Giriş" hareketi beraberinde "Çıkış" ya da "Yazma" veri hareketini içeriyorsa [15] bu süreç işlevsel süreçtir diye işaretlenerek hesaplamaya katılır. Her bir veri hareketi alt türü 1 CFP olarak sayılır.

addMethodsToLib(String operationType, String userInterface): Kütüphane metotlarına erişmek için, uygulama kaynak kodunun ilgili yerlerinde çağrılan ortak metottur. Kaynak koda eklenen bu kütüphane metotu, ayrıştırma işlemine göre operasyon türünde "Entry", "Read", "Write", "Exit" gibi farklı parametreler alır. Diğer bir parametre olarak ise, modül bazında hesaplama yapabilmek için kaynak kodda çağrılan sınıfın bulunduğu paket adının son hanesini almaktadır.

writeResultToFile(): Hesaplama detaylarının Word dokümanına kaydedilerek kullanıcıya gösterilmesi işleminden sorumludur. calculateCOSMICSize() metotu tarafından çağrılır. Toplam COSMIC büyüklük sonucu, modül bazında sonuçlar, veri hareketlerinin sayıları Word dokümanında listelenir.

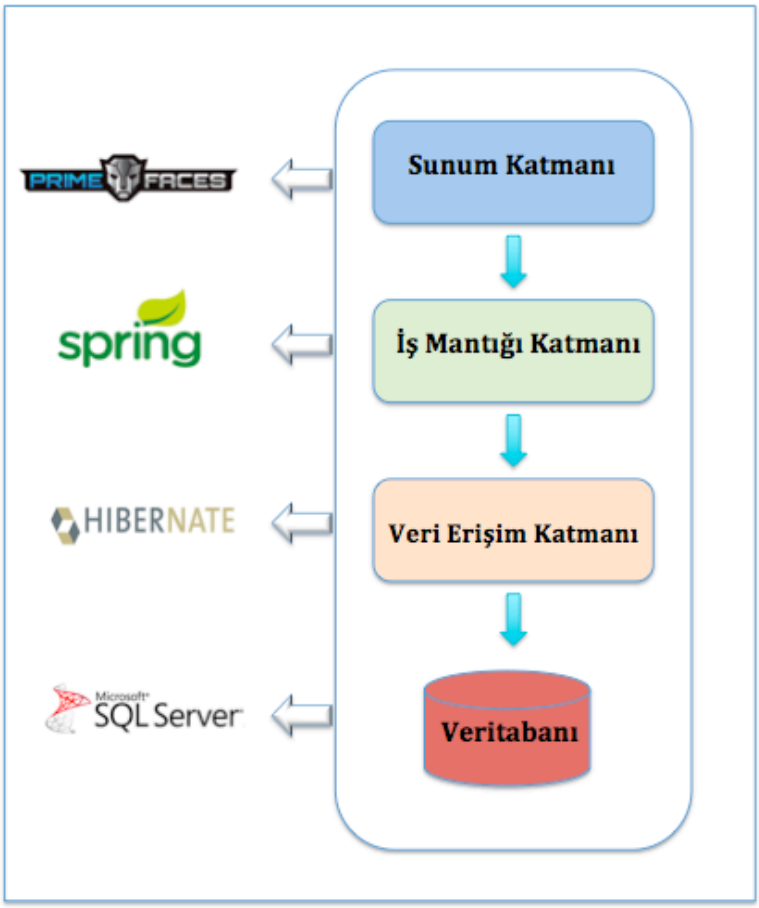
Ölçme işleminin uygulanacağı üç katmanlı mimari ve bu mimarinin önerilen model ile olan ilişkisi, Şekil 5.3'te verilmiştir.



Şekil 5.3. Otomatik Ölçme Yönteminin Mimarisi

5.3. Hedef Uygulamaların Mimarisi

Önerilen model, JSF framework yapısına sahip uygulamaların otomatik olarak COSMIC işlevsel büyüklüklerini hesaplamak için tasarlanmıştır. Spesifik mimariye sahip olan uygulamalar için geçerli olan modelde, uygulamanın sahip olduğu mimari yapı, ölçme metotu için ön koşuldur. Hedef uygulamaların mimari yapısı Şekil 5.4'te verilmiştir. Uygulamaların JSF, Spring ve Hibernate teknolojilerini kullandıkları öngörülmüştür.



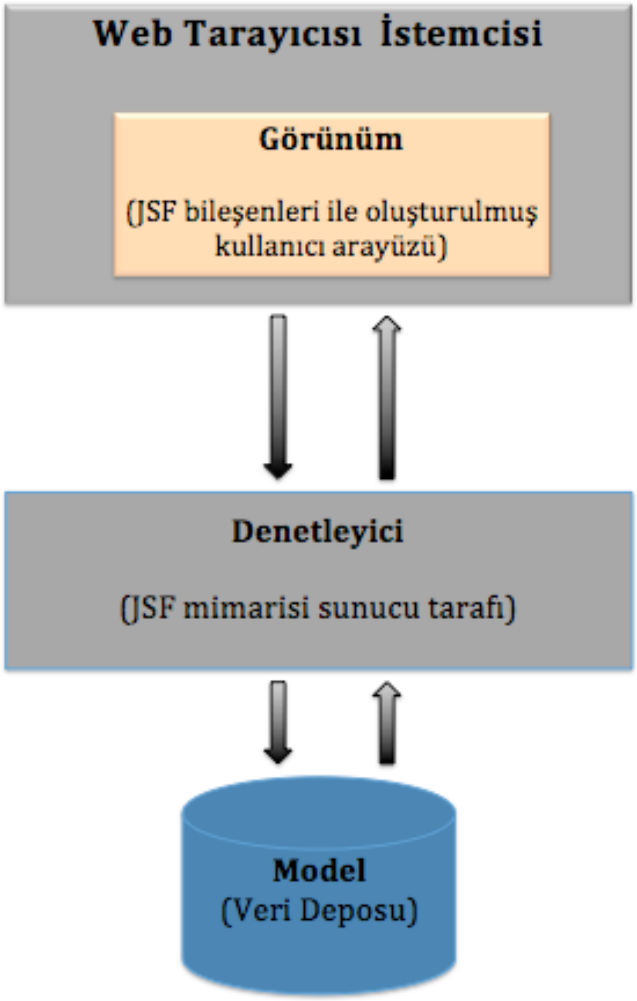
Şekil 5.4. Üç Katmanlı Hedef Uygulamaların Mimarisi

JSF (Java Server Faces), JCP (Java Community Process) tarafından geliştirilen java teknolojisi üzerine kurulmuş web uygulamaları geliştirilen MVC ("Model-View-Controller – Model-Görünüm-Denetleyici") framework üdür [56]. Şekil 5.5 JSF MVC mimarisini açıklamaktadır. JSF'nin kendi standart çatısı ("framework") ile web uygulamaları geliştirilebilir. Bunun yanında JSF standartlarını uygulayan açık kaynak kütüphaneler (PrimeFaces, RichFaces, Ajax4JSF, MyFaces gibi) vardır.

Bunları ihtiyaçlarımıza göre kullanabiliriz. JSF Event Driven Framework (EDF) sağlar. Yani herhangi tetikleme işleminden sonra bir metot çalışır. Fare tıklanması ve klavye tuşuna basılması gibi işlemler tetikleme işlemlerine örnektir. JSF'de kullanılabileceğimiz birçok bileşen mevcuttur. Böylece yazılımcılar Http'den verileri alıp göndermek yerine olaylara ("event") yoğunlaşır.

Managed Beans: JSF framework ünde "Managed Bean" kavramı yer alır. Managed Bean (Backing Bean) her sayfanın iş mantığını ve model nesnelerini içerir. Bir diğer tanımla, yaşam evresi JSF IOC (Inversion of Control) tarafından yönetilen java sınıflarıdır. Managed Bean, getter ve setter metotlarını, iş mantığını ya da backing bean leri içeren normal Java bean idir. @ManagedBean ek açıklaması ("annotation") kullanılarak istemci tarafından bu sınıflara erişim sağlamak mümkündür. Kullanıcı arayüzü bileşeni için Model olarak çalışır. Managed Bean ler üzerinden erişilecek Model sınıfları da oluşturulması mümkündür. Model sınıfları ve Managed Bean ler ön yüzde form mantığıyla aldığımız değerler, Managed Bean içerisinde saklandığından, önerilen modelde bu sınıflar üzerinde ilgili yerlere kütüphane metotları yerleştirilmiştir.

JSF sayfaları .xhtml uzantılı dosyaları ifade eder. JSF sayfasında kullanıcıdan aldığımız veya kullanıcıya bilgi amaçlı gösterdiğimiz form değerleri Managed Bean tarafında saklanır. Örnek vermek gerekirse, Şekil 5.6'daki gibi bir JSF sayfamız olsun. jsf sayfasında kullanıcıdan 'username' bilgisi girilmesi istenmiş. Bu bilgi Managed Bean tarafında Şekil 5.7'de gösterildiği gibi tutulur.



Şekil 5.5. JSF MVC Mimarisi [57]

```
<h:form>
    <table>
 <tr>
    <td><h:outputLabel value="Username" /></td>
    <td><h:inputText value="#{userManagedBean.username}" /></td>
 </tr>
    </table>
</h:form>
```

Şekil 5.6. Örnek JSF sayfası [58]

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.*;

@ManagedBean(name="userManagedBean")
@SessionScoped
public class UserManagedBean{

 private String username;

 public String getUsername() {
  return username;
 }

 public void setUsername(String username) {
  this.username = username;
 }
}
```

Şekil 5.7. Örnek Managed Bean [58]

Hedef uygulamalarda Servis/İş katmanında iş kurallarının yönetildiği Servis sınıfları olduğu kabul edilmiştir. Data Access Object (DAO) sınıflarında ise veri tabanına ulaşmak ve Oluşturma/Okuma/Güncelle/Silme (“Create/Read/Update/ Delete - CRUD”) işlemlerinin yapıldığı öngörülmüştür.

5.4. Ayrıştırma İşlemi

Ayrıştırma İşlemi bileşeni, uygulamanın kaynak kodunu ayrıştırarak tetikleyici olyları belirledikten sonra uygun yerlere kütüphanenin *addMethodsToLib()* metot çağrısını otomatik olarak ekler.

Kaynak kodun ayrıştırma işlemi yapılırken Java API altında, java.util paketinde bulunan “Pattern” ve “Matcher” sınıfları kullanılmıştır. Kaynak koddaki metotların içeriği hakkındaki (metot adı, başlangıç-bitiş satır numarası, metot return type, parametreleri, içeriği vb.) bilgiler için “javaparser” kütüphanesi kullanılmıştır.

Ayrıştırma İşlemi sırasında baz alınan bazı metotlar ve ilişkili oldukları anahtar

kelimeler vardır. Çizelge 5.1 ayrıştırma işlemi sırasında veri hareketlerini yakalamak için dikkate alınan anahtar kelimeleri özetlemektedir.

Çizelge 5.1. Veri Hareketlerine Karşılık Gelen Anahtar Kelimeler

Anahtar Kelimeler	**Karşılık Gelen Veri Hareketi**
action, actionListener, fileUploadListener	Entry
save, persist, update, delete, merge, create	Write
Classname.class, from, join	Read
.show, (DataTable), FacesMessage	Exit

“Giriş” veri hareketi türü için, kaynak koddaki xhtml sayfaları taranır ve tetikleyici metotlar bulunur. Tipik bir JSF sayfasında, Managed Bean lere erişim için kullanılan ve kullanıcı tetiklenmesiyle (buton tıklanması gibi) sürecin başlamasını sağlayan “action”, “actionListener” ve “fileUploadListener” anahtar kelimeleri, ayrıştırma işleminde baz alınmışlardır. Bu anahtar kelimelerin tetiklendiği metot adları kayıt altına alınarak, Managed Bean lerde bu metotlar bulunmuş ve *addMethodsToLib()* kütüphane metotu, “Entry” parametresi ve ilgili paketin son hanesini parametre alarak bu yerlere eklenmiştir.

“Okuma” ve “Yazma” veri hareketleri için, Servis ve DAO sınıfları taranarak Çizelge 5.1’de listenen kelimelerin geçtiği metotlara *addMethodsToLib()* kütüphane metotu ilişkili olduğu veri hareketi türüne göre (“Read” ya da “Write”) eklenmiştir. “Okuma” hareketleri tespit edilirken, HQL (Hibernate Query Language) ve Criteria türündeki sorgu türleri düşünülmüştür.

Tablolar, diyaloglar, bilgi/hata mesajları “Çıkış” veri hareketi olarak düşünülmüştür. Bu bilgi Managed Bean sınıflarında aranarak diğer hareketlerde olduğu gibi *addMethodsToLib()* kütüphane metotu, “Exit” parametresi geçilerek bu yerlere eklenmiştir.

Yukarıda belirtilen durumlara ek olarak, COSMIC Ölçüm Kılavuzunda [15] belirtilen, “bir işlevsel sürecin girdisi birden çok veri grubu içeriyor ise, girdideki her bir benzersiz veri grubu için bir Giriş tanımlanması kuralı için”, Managed Bean lerin ilişkili oldukları Model sınıflarının *setter()* metotlarında nesne türünde parametre alan metotlara *addMethodsToLib()* kütüphane metotu “Entry” parametresi

geçilerek eklenmiştir. Böylece arayüzde kullanıcının girdiği değerler veri tabanındaki birden fazla tabloya dokunuyor ise, bu yöntemle bu tablo sayıları elde edilmeye çalışılmıştır.

Yine aynı şekilde "kalıcı belleğe birden çok veri grubu hareket ettirmek durumunda, kalıcı belleğe hareket ettirilen her bir benzersiz veri grubu için bir Yazma tanımlanır" kuralını sağlamak için oldukları Model sınıflarının *setter()* metotlarına *addMethodsToLib()* kütüphane metotu "Write" parametresi geçilerek eklenmiştir. Böylece Hibernate in tek bir "Write" cümlesi ile birden fazla tabloya yazma işleminin kod taraması sırasında anlaşılamaması problemi aşılmaya çalışılmıştır.

"Eğer bir işlevsel süreçten çıktılar birden fazla veri grubundan oluşuyorsa, çıktıdaki benzersiz her bir veri grubu için bir Çıkış belirlenir" kuralı için de yine Model sınıflarının *setter()* metotlarına *addMethodsToLib()* kütüphane metotu "Exit" parametresi geçilerek eklenmiştir. Bu sayede, birden fazla tablodan veri çekilerek getirilen sonuçlar için ne kadar tablodan veri çekildi ise o kadar "Çıkış" sayısı elde edilmesi amaçlanmıştır.

Son olarak, Java sınıflarında en az bir kez kütüphane metotu eklendiğinde kütüphane "import" edilerek ilgili sınıfa tanıtılmış olunur.

Ayrıştırma İşlemi bir kez yapıldıktan sonra, uygulama otomatik ölçüm için hazır hale gelmektedir.

5.5. Varsayım ve Kısıtlar

Yukarıdaki bölümlerde detayları belirtilen, uygulamanın çalışması sırasında COSMIC işlevsel büyüklüğü otomatik olarak hesaplama yönteminin gerçekleştirimi ve kullanımıyla ilgili bazı kısıtlar bulunmaktadır.

İlk olarak, önerilen model JSF tabanlı web uygulamaları için uygundur. MVC yapısını baz alarak Ayrıştırma İşlemi bileşeni ilgili işlemleri gerçekleştirdiğinden, başka teknolojileri kullanan uygulamalarda bu model ile istenilen seviyede sonuç alınamayacaktır.

İkinci olarak, otomatik ölçüm çalışma zamanında kullanıcı senaryoları üzerinden gerçekleştiğinden, herhangi bir işlevsel kullanıcının (insan, makine vb.) event leri tetikleyerek işlem yapması gerekmektedir.

Üçüncü olarak, hesaplama sonuçlarının kullanıcıya gösterilmesi için sistemden bir ‘Çıkış’ işleminin yapılması gerekir. Ayrıştırma İşleminde bu durum ‘logout()’ fonksiyonu ile baz alındığından, sistemden başka türlü bir çıkış gerçekleşirse bu durum Ayrıştırma İşlemi bileşenine tanıtılmalıdır.

Dördüncü olarak, modül bazında hesaplama işlemini ayırt etmek için, Measurement Library de gelen “Entry” bilgisi içerisindeki ilgili sınıfın bulunduğu paket adının son kelimesini alınmaktadır. Eğer, ilgili uygulamada aynı modülü ilgilendiren kodlar farklı paketlerde farklı adlarla oluşturulduysa, modül bazında hesaplama işleminin başarı seviyesi istenilen düzeyde olmayacaktır.

Son olarak, Ayrıştırma İşlemi bileşeni metot çağırma hiyerarşisine bakmadığından, birbirini çağıran metotlar için ekstra hesaplamalar yapılabilmektedir. Uygulamada veri tabanı sorgu işlemleri ortak bir altyapı vs. gibi durumları kullandığında ölçme sonucu yine beklenen seviyede olmayacaktır.

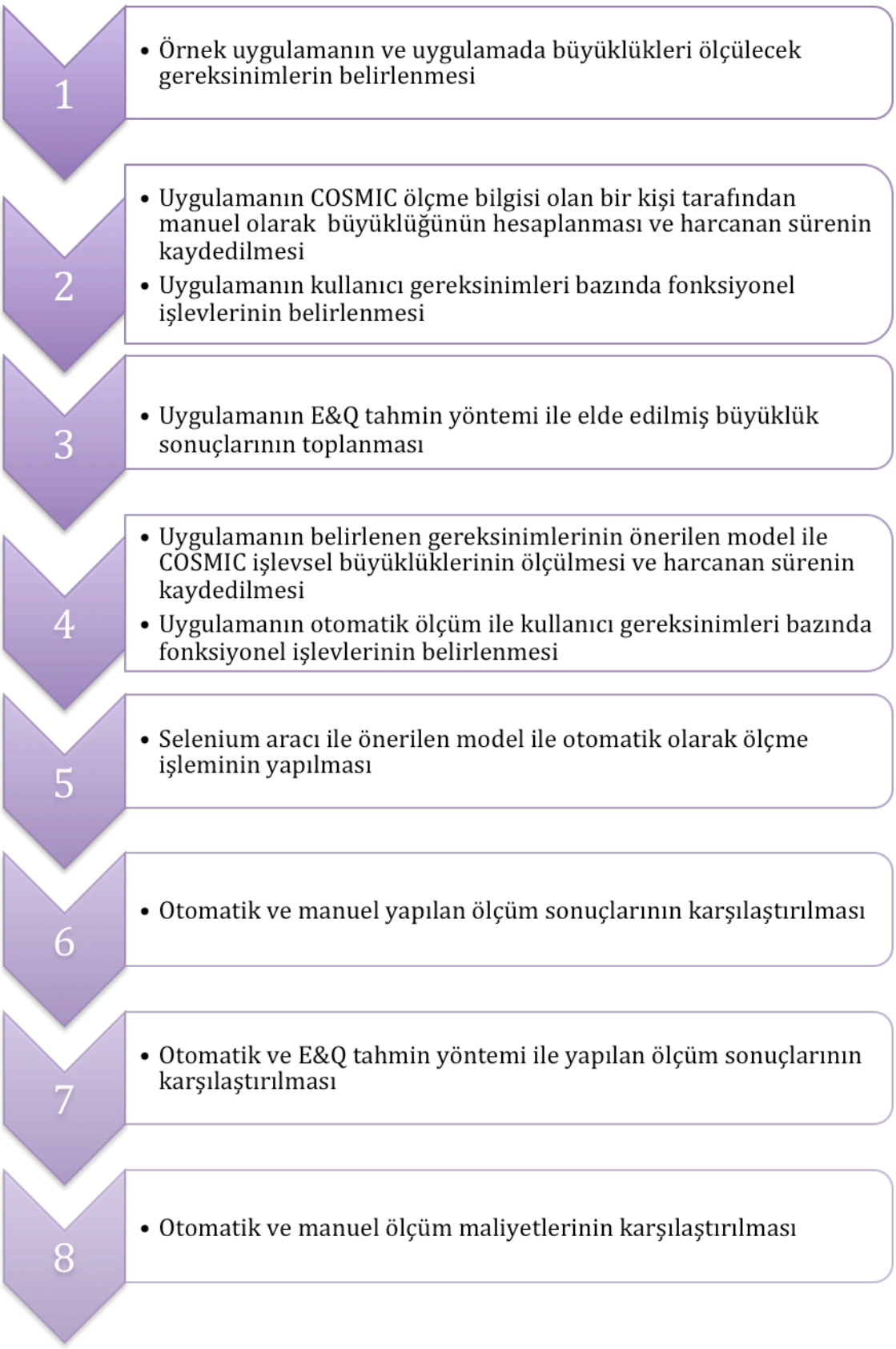
6. ÖRNEK UYGULAMA

Bu bölümde, önerilen yöntemin uygulanabilirliğini ve maliyet-etkinliğini değerlendirmek için yapılan çalışmaya yer verilmiştir. Önerilen yöntemi test etmek amacıyla seçilen Java iş uygulaması, 12 yazılım mühendisi tarafından geliştirilmiş ve bir kamu kurumunda aktif olarak kullanılan bir bilişim sistemi geliştirme projesidir. Uygulama yaklaşık 370 bin satır kod içermekte ve mimari yapısı Bölüm 5.3'te anlatılan mimariye çok yakındır. Uygulama PrimeFaces (açık kaynak kodlu Ajax temelli JSF bileşeni), Spring Framework (Java geliştirme platformu), Maven (Java Deployment Tool - Java Dağıtım Aracı) ve Hibernate (ORM aracı) teknolojilerini kullanmaktadır. Örnek uygulamada genel olarak Ekle / Okuma / Güncelle / Silme / Listeleme (CRUDL) işlemleri yapılmaktadır. Uygulamanın web-servisler aracılığı ile başka uygulamalar ile de entegrasyonu mevcuttur.

Uygulama çok büyük ve geniş kapsamlı olduğundan ölçme işlemini gerçekleştirmek için uygulamanın bir bölümü pilot olarak seçilmiştir. Seçilen kısım, uygulamanın bir modülündeki yaygın olarak kullanılan senaryolardan bazılarını kapsamaktadır. Yaklaşık 3 bin satır koda denk gelen bu kısım 9 kullanım durumunu içermektedir. Kurumun güvenlik politikalarından dolayı seçilen kullanım durumları hakkında detaylı bilgi verilemeyecektir, ancak seçilen kullanım durumlarının temel olarak CRUDL ("Create/Read/Update/Delete/List") işlemlerinden oluştuğu bilinmelidir. Seçilen kullanım durumları tipik kullanım durumlarıdır ve işlevleri ve karakteristikleri sistemdeki diğer kullanım durumlarına benzemektedir.

6.1. Eylem Planı

Durum çalışması yapılacak örnek uygulama belirlendikten sonra eylem planına göre durum çalışmasında izlenecek adımlar Şekil 6.1'de gösterilmektedir.



Şekil 6.1. Eylem Planı

6.2. Manuel Ölçüm

Örnek uygulamada manuel olarak hesaplanan COSMIC işlevsel büyüklük, arayüze bakarak ilgili arayüzlere karşılık gelen kaynak kod ve Varlık-İlişki diyagramlarına göre COSMIC ölçüm kılavuzunda [15] belirtilen kurallara göre hesaplanmıştır. COSMIC bilgisi orta düzeyde olan bir yazılımcı tarafından 9 kullanım durumunu içeren kısım manuel olarak yaklaşık 4.5 saat sürede hesaplanmıştır. Burada kritik olan nokta, ölçerin COSMIC FSM'ye ve uygulamaya aşina olmasıdır. Bu ölçümün uygulamayı bilmeyen biri tarafından yapıldığında, manuel ölçme maliyetinin artması kaçınılmaz olacaktır.

Çizelge 6.1 manuel ölçüm sonuçlarını özetlemektedir.

Çizelge 6.1. Manuel Ölçüm Sonuçları

Kullanım Durumu Sayısı	9
Toplam İşlevsel Büyüklük	952
Harcanan Süre	~ 4.5 saat
Toplam İşlevsel Süreç Sayısı	25

6.3. Otomatik Ölçüm

Ayrıştırma İşlemi, bu çalışma kapsamında seçilen uygulamanın belli kısımları için uygulanmış ve koddaki ilgili yerlere oluşturulan kütüphanedeki *addMethodsToLib()* metotu otomatik olarak eklenmiştir. 9 kullanım durumu birer birer koşulmuş ve bu kullanım durumlarına denk gelen işlevsel büyüklük çalışma zamanında otomatik olarak hesaplanmıştır. Sistemden çıkış yapıldıktan sonra ölçüm yapan kişiye sonuçlar Word belgesinde gösterilmiştir. Kullanım durumları üzerinden geçerek otomatik olarak hesaplanan ölçüm sonucu yaklaşık 10 dakika sürmektedir.

Yazılımcı-1 tarafından gerçekleştirilen otomatik ölçüm sonuçları Çizelge 6.2 özetlemektedir. Yazılımcı-2 tarafından yapılan otomatik ölçüm sonuçları ise Çizelge 6.3'te belirtilmiştir.

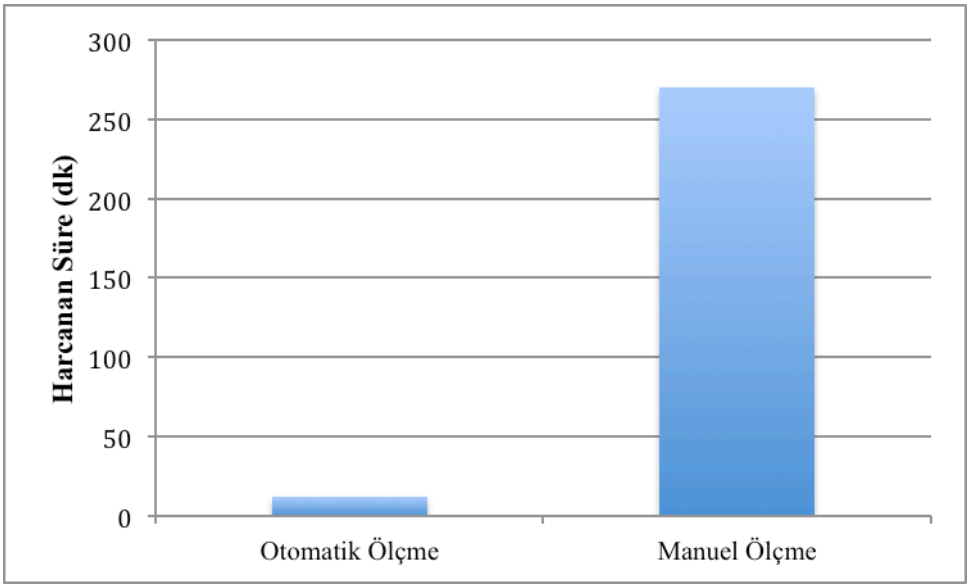
Çizelge 6.2. Otomatik Ölçüm Sonuçları (Yazılımcı-1)

Kullanım Durumu Sayısı	9
Toplam İşlevsel Büyüklük	975
Harcanan Süre	~ 10 dk
Toplam İşlevsel Süreç Sayısı	25

Çizelge 6.3. Otomatik Ölçüm Sonuçları (Yazılımcı-2)

Kullanım Durumu Sayısı	9
Toplam İşlevsel Büyüklük	975
Harcanan Süre	~ 14 dk
Toplam İşlevsel Süreç Sayısı	25

Şekil 6.2 otomatik ve manuel ölçüm için harcanan süre miktarlarını dakika cinsinden göstermektedir. Otomatik ölçümün, manuel ölçüme nazaran maliyet-etkin olduğu bilgisi görülmektedir.



Şekil 6.2. Otomatik ve Manuel Ölçüm Süreleri

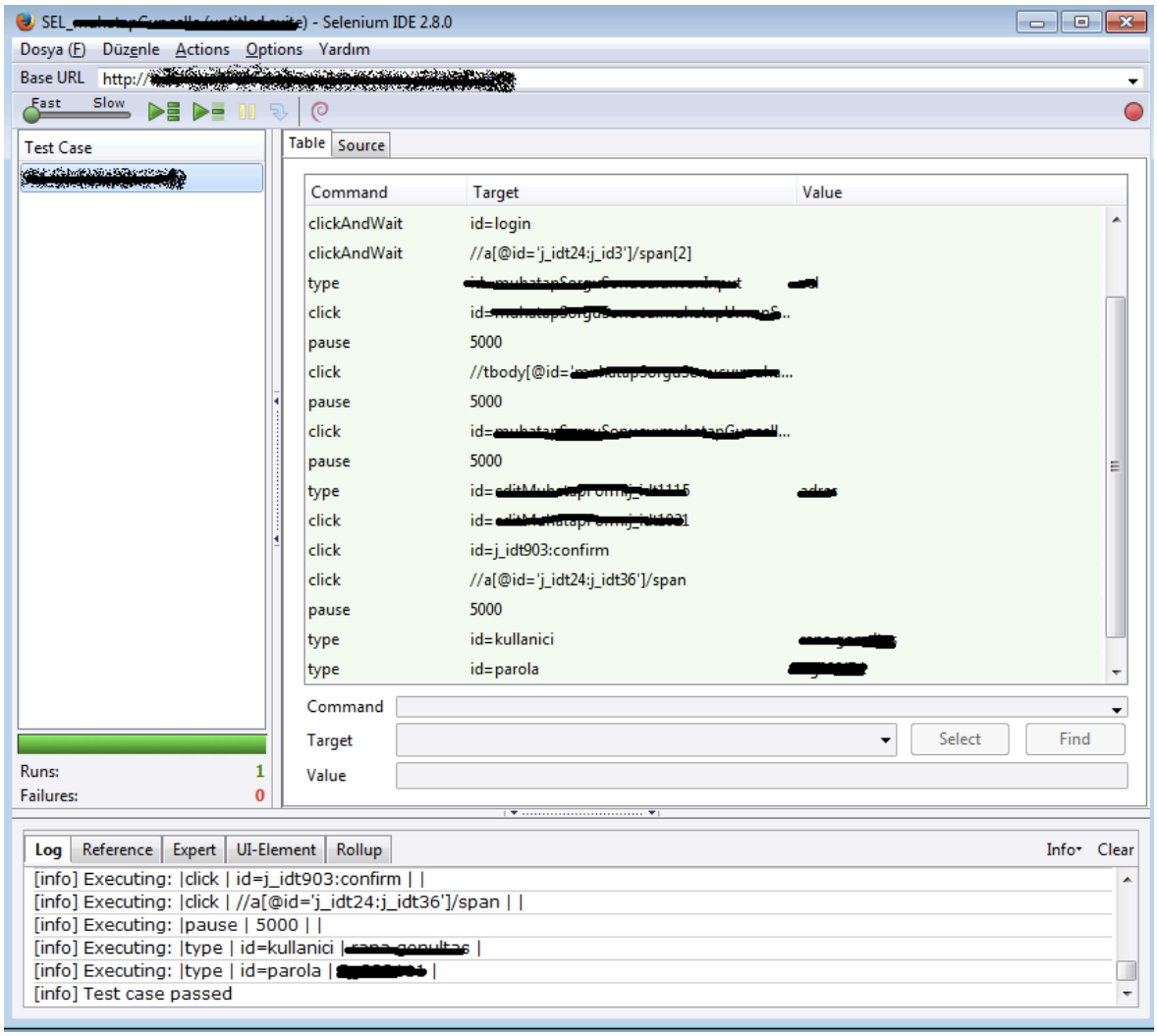
6.4. Selenium ile Otomatik Ölçüm

Selenium, web tabanlı uygulamaların testlerinin tarayıcı üzerinden yapılmasını sağlayan bir araçtır. Test adımlarını web üzerinden görebilmek ve tanımlayabilmek için selenium IDE kullanılır. Selenium IDE mevcut durumda Firefox web tarayıcısı için geliştirilmiş bir eklentidir. Selenium ile web uygulamalarındaki

hareketler kaydedilebilir. Bu özellik ile kullanıcı senaryolarına bağlı olarak tanımlanan testlerin her seferinde tanımlanmasına gerek duyulmadan bir kez kaydedilen test senaryosu gelecek durumlar için otomatik olarak tekrar koşulabilir.

Önerilen modelde otomatik ölçme işlemi çalışma zamanında kullanıcı senaryolarına bağlı olarak ölçüldüğünden otomatik test aracının ileriki ölçümler için kullanılmasının faydalı olacağı düşünülmüştür. Bu sebeple Selenium'da otomatik ölçüm kapsamında büyüklükleri ölçülecek kullanım durumları tanımlanarak ölçümün bir makine tarafından da otomatik olarak yapılabilmesi sağlanmıştır.

Şekil 6.3 Selenium IDE'nin arayüzünü göstermektedir.



Şekil 6.3. Selenium Aracı ile Otomatik Ölçüm İşlemi Arayüzü

Selenium aracına tanımlanan kullanıcı senaryoları ardı ardına koşularak otomatik ölçüm sonuçları elde edilmiştir. Selenium ile elde edilen sonuçlar, Bölüm 6.3'te

belirtilen uzman tarafından ölçülen sonuçlarla aynıdır. Çizelge 6.4 Selenium ile yapılan otomatik ölçüm sonuçlarını göstermektedir.

Çizelge 6.4. Selenium ile Yapılan Otomatik Ölçüm Sonuçları

Kullanım Durumu Sayısı	9
Toplam İşlevsel Büyüklük	975
Harcanan Süre	~ 8 dk
Toplam İşlevsel Süreç Sayısı	25

6.5. Erken ve Hızlı Tahminleme Yöntemi ile Ölçüm

Örnek uygulamanın geliştirildiği kurumda var olan Kalite ve Süreç Yönetim Birimi, belli başlı yöntemler ile kurumda yürütülen projelerin büyüklüklerini hesaplamaktadır. Proje büyüklüklerinin hesaplanması görevine ek olarak birim, kurumda yürütülen faaliyetlerle ilgili süreçleri uluslararası standartlara uygun olarak ve en iyi pratikler doğrultusunda oluşturmak, bu süreçlerin uygulamalarını izlemek ve sürekli iyileşmesini sağlamak, faaliyetlerin ve iş ürünlerinin kalite hedeflerini karşılamasını güvence altına almaya çalışmaktadır.

Kurumda bulunan Kalite ve Süreç Yönetim Birimi tarafından tez kapsamında COSMIC İşlevsel Büyüklük Ölçümü ile otomatik olarak büyüklüğü hesaplanan uygulamanın büyüklüğü, COSMIC Tam İşlev Puanı biriminde, Erken ve Hızlı Tahminleme yönteminin (“E&Q”) [40] kuruma özgü olarak uyarlanmış haliyle gerçekleştirilmektedir.

Ölçüm için ilk olarak projedeki gereksinimler listelendikten sonra bu gereksinimlere karşılık gelen İşlevsel Eşlenikler belirlenmektedir. Gereksinim büyük bir işlevselliği ifade eder şekilde ise ve bu ifade bir şekilde gereksinim büyüklüğünü yönlendirmek üzere konulmuş olan kolonlarla eşleşiyorsa, ilgili gereksinim parçalara bölünmez, bunun yerine kendisini ifade eden ya da en yakın olan işlevsellik seçilmektedir. Şekil 6.4 E&Q Yöntemi ile gereksinim ve işlevsel eşleniklerin eşleştirilmesi göstermektedir.

Gereksinim														İşlevsel Eşlenikleri
Gereksinim	Gereksinim Detayları / Açıklamalar	Varlık Ekle	Varlık Güncelle	Varlık Sil	Varlık Görüntüleme	Varlık Listeleme	Varlık Yönetimi	Basit Varlık CRUD	Kompleks Varlık CRUD	Basit Varlık Sorgulama	Kompleks Varlık Sorgulama	Onaylama	Reddetme	Raporlama (Varsayımsal Rapor Sayısı Belirtilmelidir. Maks: 20)
Örnek Gereksinim	Örnek gereksinim açıklamasıdır.													

Gereksinim	Gereksinim Detayları / Açıklamalar	Varlık Ekle	Varlık Güncelle	Varlık Sil	Varlık Görüntüleme	Varlık Listeleme
Proje Ekle / Güncelle	Projeler eklenir ve güncellenir ancak silinemez.	1	1			

Gereksinim	Gereksinim Detayları / Açıklamalar	Varlık Ekle	Varlık Güncelle	Varlık Sil	Varlık Görüntüleme	Varlık Listeleme	Varlık Yönetimi	Basit Varlık CRUD	Kompleks Varlık CRUD
Proje CRUD	Projeler karmaşık bir yapıdır, kendisine bağlı Proje İzleme ve Gerçekleşme varlıkları da bulunmaktadır.								1

Şekil 6.4. E&Q Yöntemi İle Gereksinim ve İşlevsel Eşleniklerin Listelenmesi

Daha sonra işlevsel eşlenikleri bulunan gereksinimlere karşılık gelen E&Q Estimation yöntemine dayanan işlevsel büyüklük değerleri Şekil 6.5'te gösterildiği hesaplanmaktadır.

Gereksinim	Gereksinim Detayları / Açıklamalar	FS	FM	FL	FV	TS	TM	TL	GS	GM	GL	MS	MM	ML	Minimum	Ortalama	Maksimum
Proje CRUD	Projeler karmaşık varlıklardır,	0	0	0	0	0	0	1							42	48,5	63
Kullanıcı Ekle		0	0	0	1	0	0	0							8	10,5	14
Kullanıcı Güncelle		0	0	0	1	0	0	0							8	10,5	14
Kullanıcı Sorgula		0	0	0	1	0	0	0							8	10,5	14

Şekil 6.5. E&Q Yöntemi İle İşlevsel Büyüklük Hesaplama

İşlevsel Eşlenik'i bulunamamış olan gereksinimler Şekil 6.6'da gösterildiği gibi listenin sonunda yer alır ve bunlara denk gelen E&Q tahmin işlevsel eşlenikleri seçilir. Bunun içinse E&Q-Referans Tablosu (İşlevsel Eşlenik'i bulunamamış olan gereksinimlere ait işlev puanını belirlemek ve puanlar sekmesinde uygun alana işlemek için kullanılır) kullanılabilir. Böylelikle tüm gereksinimlere ait işlevsel büyüklükler belirlenmiş olur.

Gereksinim	Gereksinim Detayları / Açıklamalar	FS	FM	FL	FV	TS	TM	TL	GS	GM	GL	MS	MM	ML	Minimum	Ortalama	Maksimum
Proje CRUD	Projeler karmaşık varlıklardır,	0	0	0	0	0	0	1							42	48,5	63
Kullanıcı Ekle		0	0	0	1	0	0	0							8	10,5	14
Kullanıcı Güncelle		0	0	0	1	0	0	0							8	10,5	14
Kullanıcı Sorgula		0	0	0	1	0	0	0							8	10,5	14
Sınıflandırılamamış Gereksinim X									1						20	60	110

Şekil 6.6. İşlevsel Eşlenik'i Bulunamamış Gereksinimler

Sonuç olarak tüm işlevsel büyüklük değeri toplam değerden elde edilerek Şekil 6.7'de gösterildiği gibi kestirim tamamlanmış olur.

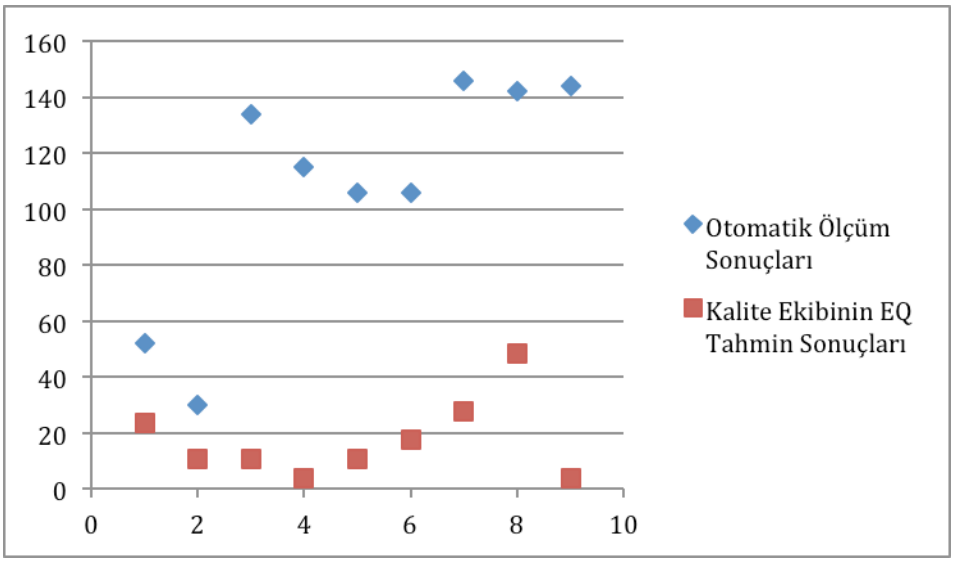
Minimum	Ortalama	Maksimum
42	48,5	63
8	10,5	14
8	10,5	14
8	10,5	14
20	60	110
	140	

Şekil 6.7. E&Q Yöntemi İle Hesaplanan Toplam Sonucun Listelenmesi

Çizelge 6.5 E&Q tahminleme yöntemi ile ölçülen sonuçları özetlemektedir. Şekil 6.8'de ise otomatik ölçüm sonuçları E&Q tahminleme yöntemi ile karşılaştırılmaktadır. Otomatik ölçme yöntemi sonuçlarının E&Q yöntemine göre, kullanım durumlarındaki işlevsel büyüklük puanları için farkı daha iyi ayırt ettiği söylenebilir. Ancak, daha sağlıklı bir değerlendirme yapabilmek için işlevsel büyüklük ölçüm sonuç verilerinin sayısı arttırılmalıdır.

Çizelge 6.5. E&Q Yöntemi ile Manuel Olarak Hesaplanan Sonuçlar

Kullanım Durumu Sayısı	9
Toplam İşlevsel Büyüklük	128.9
Harcanan Süre	~ 5 dk



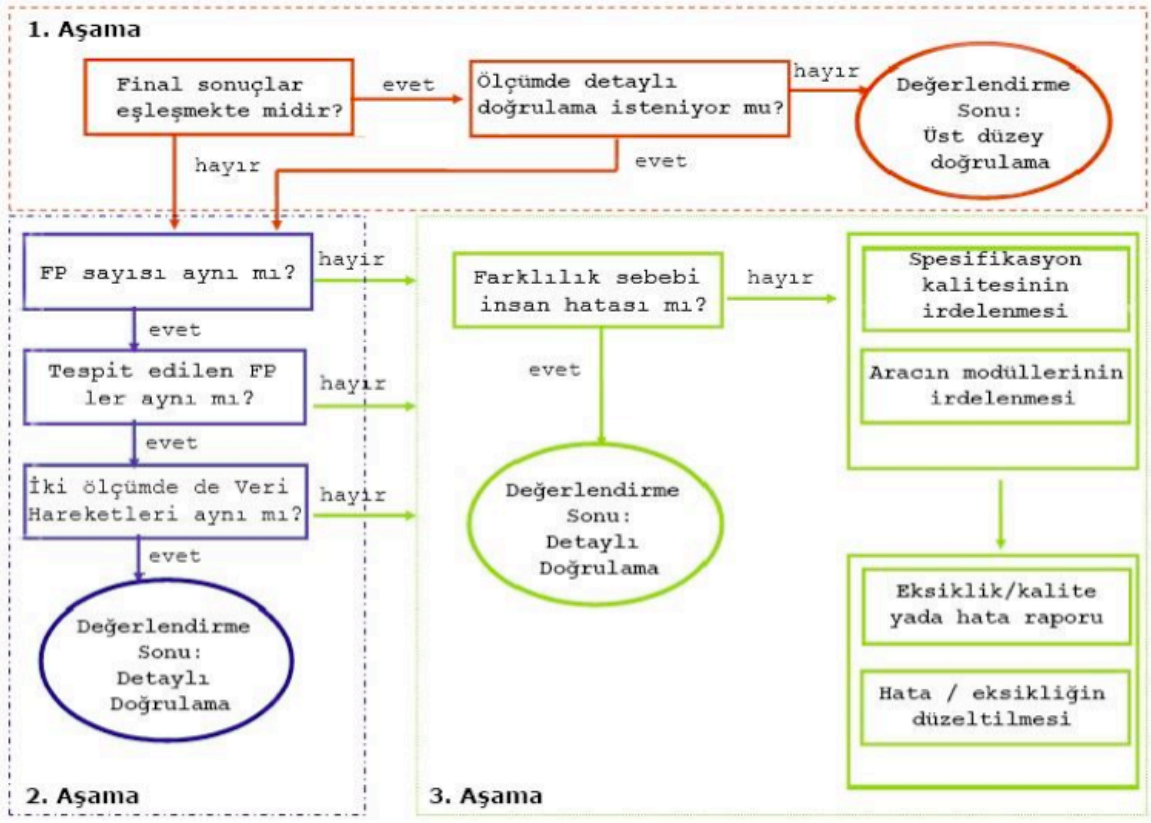
Şekil 6.8. Otomatik Ölçüm Sonuçlarının E&Q Tahminleme Yöntemi İle Karşılaştırılması

E&Q yöntemi ve otomatik ölçüm sonuçlarının farklı çıkmasına yol açan bazı sebepler bulunmaktadır. Bunlardan ilki E&Q yönteminin, var olan kullanım durumlarında ifade edilen değerlere göre bir tahminleme işlemi yapmasıdır. Tahminleme yaparken daha önceden belirlenen sınırlar dışına çıkılamaması (Çizelge 2.5) ve bu sebeple sonuçların genel olarak belli bir sayı aralığında olması farklılığa neden olan sebeplerden biridir. Otomatik ölçümün, koddan yola çıkarak bir hesaplama işlemi yapması sonucunda E&Q yöntemine göre daha büyük ve farklı sayıda sonuçlarının çıkması kaçınılmazdır.

7. YÖNTEME İLİŞKİN DEĞERLENDİRME

7.1. Değerlendirme

COSMIC ISO 19761 ölçme metotunu otomatikleştiren araçlar için tasarlanan üç aşamalı doğrulama protokolü [59] ile önerilen yöntem ile elde edilen sonuçların doğruluğu değerlendirilmiştir. Protokolün amacı, tüm ölçme zincirinin doğru ölçüm sonuçlarını ürettiğini garanti etmektir. Protokol, Şekil 7.1'de belirtilen adımları izleyerek sonuçları karşılaştırmayı öngörür.



Şekil 7.1. FSM otomasyon araçları doğruluğu için 3 aşamalı doğrulama protokolü [59]

- 1. Aşama: Sayısal sonuçların karşılaştırılması. Bu aşamada, ölçülen yazılımın toplam büyüklüğü (COSMIC İşlev Puanı cinsinden) manuel ve otomatik ölçüm sonuçlarından elde edilen değerler ile karşılaştırılır. Eğer sonuçlar aynı çıkar ise, otomatik ve manuel ölçme süreçlerinde bir farklılık olmadığı anlaşılır. Ancak, sonuçların aynı çıkması, her bir COSMIC eleman değerlerinin ve süreçlerin aynı olacağını göstermez. Eğer doğrulama bu aşamada biter ise, ancak son sonuçlar doğrulanmış olur.

Tez çalışması kapsamında önerilen yöntem ile otomatik olarak hesaplanan sonuçlar, orta seviyede ölçme bilgisi olan uzman tarafından manuel olarak yapılan ölçme işlemi sonuçları ile karşılaştırılmış ve Çizelge 7.1'de gösterilen değerler elde edilmiştir. Toplam işlevsel puana göre değerlendirilen sonuçların yakın olduğu (%96 doğruluk ile) ancak eşit olmadığı gözlenmiştir. Bu sebeple 2. Aşama doğrulama protokolüne geçilmiştir.

Çizelge 7.1. Doğrulama Protokolü 1. Aşama Sonuçları

Manuel Yapılan Ölçüm (CFP)	**Otomatik Yapılan Ölçüm (CFP)**
Toplam Puan : 936	Toplam Puan: 975

- 2. Aşama: Detaylı karşılaştırma. Eğer 1. Aşamada sayısal sonuçlar eşleşmezse ve bu farklılığın sebebi anlaşılmak isteniyorsa, sonuçlar daha detaylı olarak karşılaştırılır. Otomatik ve manuel ölçme sonuçları daha detaylı karşılaştırılarak (örn: işlevsel süreçler ve veri hareket alt türü seviyesinde) farklılığın sebebi bulunur. Değerlendirme yapılırken sonuçlar üzerinde insan kaynaklı bir hata olup olmadığı da detaylı ölçme sonuçlarının değerlendirilmesi sonucu ortaya çıkabilir.

  Bu aşama için, otomatik ve manuel ölçme ile elde edilen işlevsel süreçlerin sayısı ve veri hareketlerinin sayısı karşılaştırılmıştır. Sonuç olarak işlevsel süreçlerin sayılarının aynı, veri hareketlerinin sayılarının farklı olduğu ortaya çıkmıştır. Çizelge 7.2 otomatik ve manuel ölçüm sonuçlarına göre kullanım durumlarının listesini ve bunlara karşılık gelen işlevsel süreçlerin sayısı ile her bir kullanım durumua ait CFP'yi göstermektedir.

Çizelge 7.2. Doğrulama Protokolü 2. Aşama Sonuçları

Kullanım Durumu No	İşlevsel Süreç Sayısı		COSMIC İşlev Puan Sayısı	
	Otomatik Ölçüm	Manuel Ölçüm	Otomatik Ölçüm	Manuel Ölçüm
KD#1	1	1	52	46
KD#2	1	1	30	24
KD#3	3	3	134	128
KD#4	3	3	115	109
KD#5	3	3	106	112
KD#6	3	3	106	114
KD#7	3	3	146	138
KD#8	3	3	142	135
KD#9	5	5	144	130
Toplam	**25**	**25**	**975**	**936**

- 3. Aşama: Otomasyon aracı ve girdi doğrulaması. Bu aşama, otomasyon aracının hangi kısmının hatadan sorumlu olduğuna karar verme ve hataya sebep olan kusurları bulma adımlarını içerir. Hata düzeltikten sonra, otomasyon aracı tekrar çalıştırılır ve yeniden doğrulama işlemi yapılır.

  Önerilen model doğrultusunda yapılan ölçme sonuçlarındaki eksiklik ve hatalar bu aşamada değerlendirilmiş ve geliştirici kaynaklı birkaç hatanın düzeltilmesi gereği ortaya çıkmıştır. Düzeltmeler sonrasında ölçme işlemi tekrarlanmıştır. Bölüm 7.2'de düzeltme işleminden sonra ortaya çıkan ölçme sonuçlarındaki farklılıklar detaylı bir şekilde anlatılmaktadır.

7.2. Sapma Nedenleri

Önerilen model ile yapılan otomatik ölçme sonuçları, manuel olarak yapılan ölçme sonuçlarından biraz farklı çıkmıştır. Veri hareketleri sayılarının farklı çıkmasındaki sapma nedenleri araştırılmış ve üç ana problem belirlenmiştir: Uygulama mimarisinde katmanlar arasındaki "Okuma" ya da "Yazma" veri hareketlerine ilişkin tekrarlamalı metot çağırmalarının yakalanması, Yazılımcıların farklı kodlama standartlarını benimsemeleri ve Sunum katmanında gereksiz "Giriş" veri hareketlerinin sayılmasıdır.

Problemlere ilişkin detaylı açıklamalar şöyledir:

- *Uygulama mimarisinde katmanlar arasındaki "Okuma" ya da "Yazma" veri hareketlerine ilişkin tekrarlamalı metot çağırmalarının yakalanması:* Önerilen modelde ölçme işlemi, uygulamayı kapalı kutu yaklaşımını baz alarak bütün olarak ölçülmesini amaçlanmaktadır. Ölçüm, uygulamada bulunan katmanları ayrı ayrı baz alıp buna göre ölçüm yapmayı desteklemez. Kaynak kodun ayrıştırılma işlemi sırasında metot çağrılarını yakalamak zor olduğundan kod dosya dosya okunarak sınıf bazında bir tarama işlemi yapılmaktadır. Ugulama mimarisinin ikiden fazla katmanı bulunmaktadır ve servis ile veri erişim katmanları arasında birbirini çağıran aynı kelimeleri içeren metot çağrıları bulunabilmektedir. Ayrıştırma İşlemi bileşeni, bu çağrılarda bulunan Okuma ya da Yazma veri hareketlerini ayrı ayrı dikkate aldığından bu tür durumlarda birden fazla Okuma ya da Yazma veri hareketi sayılmaktadır.
- *Yazılımcıların farklı kodlama standartlarını benimsemeleri:* Her ekibin, hatta her yazılımcının kendine ait bir kodlama standartı bulunmaktadır. Eğitim geçmişi, tecrübe, alışkanlıklar vb. özellikler kişilerin aynı problemi farklı şekilde çözmelerini, ortaklaştırma altyapısını sağlamalarını, kimi durumlarda ise gerekli görülen yerlerde özelleştirmeye gitmelerine yol açar. Bu sebeple, geliştirme ekibi Java programlama dilinin ve kullanılan teknolojinin belli başlı standartlarına uysa bile, yazılımcı seviyesinde kodlama stilleri farklılık gösterecektir. Örneğin, veri tabanı sorgulama alt yapısı takımın ihtiyaç duyduğu bir altyapıda değil ise, ekip kendi çözümlerini üretecek ve bu yeni standart sonucu ortaya çıkan yapı önerilen modeldeki Ayrıştırma İşlemi bileşeni tarafından yakalanamayacaktır.
- *Sunum katmanında gereksiz "Giriş" veri hareketlerinin sayılması:* Ayrıştırma İşlemi, Bean classlarına arayüzden tetiklenerek geçilen hareketler için Giriş hareketlerini saymaktadır. Bu işlemi gerçekleştirmek için .xhtml dosyaları taranıp JSF de tetikleme işlemini gerçekleştiren yerler bulunup Bean class larındaki metot adları işaretlenmektedir. Uygulamada .xhtml dosyalarında taranarak bulunan action metotlarıyla aynı isme sahip olan metotlar Giriş hareketi olarak sayıldığından, eğer her hangi bir metot tarafından da çağrılıyor ise fazladan Giriş hareketinin sayılmasına sebep olmaktadır.

7.3. Geçerlilik Tehditleri

Bu bölümde deneysel sosyal araştırmaların kalitesini belirlemek için yaygın olarak kullanılan dört test yöntemi kapsamında örnek uygulamanın geçerliliği sorgulanmaktadır. Yin [60] tarafından özetlenen testler; yapısal geçerlilik ("construct validity"), dahili geçerlilik ("internal validity"), harici geçerlilik ("external validity") ve güvenilirliktir ("reliability").

- **Yapısal Geçerlilik**

  Yapısal Geçerlilik, çalışılan kavramlarda doğru metriklerin belirlenmesi ile ilgilidir [60]. Veri toplama aşamasında sübjektif değerlendirmelerden kaçınmayı ve yeterli seviyede operasyonel ölçüm setlerini geliştirmeyi öngörür [60].

  Önerilen yöntem ölçme işleminin otomatikleştirilmesini amaçlamaktadır. Örnek uygulama üzerinde önerilen yöntem uygulanarak hesaplamadaki doğruluk oranı ve etkinliği test edilmiştir.

  Önerilen yöntem sonucu elde edilen sonuçların manuel hesaplama sonucuna ne kadar yakın olduğu, netlik kavramında en önemli yere sahiptir. Doğruluk değerlendirilmesi manuel ve otomatik ölçüm sonuçlarından çıkan, toplam işlevsel büyüklük puanı ve toplam işlevsel süreçlere bakılarak yapılmaktadır. Sonuçların birbirine yakın çıkması, doğru metriklerin seçildiği ve yapısal geçerliliğin sağlandığını göstermektedir.

  Ayrıca Soubra ve ark. [59] önerdiği doğrulama protokolünün kullanılması yapısal geçerliliği artırmaktadır.

  Otomatik ölçüm COSMIC Kılavuzunda [15] belirtilen kurallar çerçevesinde yapılmaktadır. Manuel olarak yapılan ölçüm ise, uygulama kodu üzerinden geçerek varlık-ilişki diyagramlarını da göz önünde bulundurarak yine COSMIC Kılavuzunda [15] belirtilen kurallara bakılarak yapılmaktadır. Büyüklük hesaplaması yapılırken harcanan efor, zaman maliyeti olarak baz alınmaktadır.

- **Dahili Geçerlilik**

  Dahili Geçerlilik sadece açıklayıcı ya da nedensel çalışmalar için geçerli olmakla birlikte betimsel ve araştırma çalışmaları için geçerli değildir.

Deneme sonucu olarak bağımlı değişkende meydana geldiği görülen gelişim, değişme ve farkı etkileyen faktörün deneysel değişken veya değişkenler olup olmadığı konusudur [60].

Örnek uygulama üç katmanlı bir Java web uygulamasıdır. Sunduğu işlevselliğe bakıldığında büyük bir uygulama olduğu söylenebilir. Seçilen modüldeki kullanım durumlarının geliştirilen modelin kriterlerini sağlaması, dahili geçerlilik oranını olumlu yönde etkilemektedir.

Dahili geçerliliğe tehdit olabilecek unsurlardan biri, ölçümü yapan ve otomatik ölçme modelini geliştiren kişinin sertifikalı bir uzman olmamasından kaynaklı ortaya çıkabilecek hata riskleridir. Bir diğer tehdit unsuru ise ölçümün örnek uygulamanın belli bir kısmı için test edilmesi ve tek bir uygulamada denenmiş olmasıdır. Ayrıca, otomatik ölçme yöntemini geliştiren kişinin örnek uygulama hakkında bilgisi olması bir başka tehdit olarak görülebilir.

- **Harici Geçerlilik**

  Harici Geçerlilik, çalışma bulgularının genelleştirilmesi için tanımlama yapılabilmesi ile ilgilidir [60]. Ölçülen sonuç, gelişme ya da farkın gerçekte bir anlamı olup olmaması, varsa bunun seviyesi ve diğer durumlar için de genelleyebilme olasılığı dış geçerliliğin derecesini gösterir [60].

  Harici geçerliliğe tehdit olabilecek unsurlardan biri, gerçekleştirilen ölçümlerin belli bir kapsam dahilinde yapılması sonucu ölçme yönteminin diğer tüm JSF tabanlı Java web uygulamalarında aynı başarı oranınına sahip olabileceğinin tam anlamıyla kanıtlanamasıdır. Ayrıca mimari yönden bir farklılık olmasa bile, Ayrıştırma İşlemi sırasında aranan anahtar sözcüklerin yetersiz kalması, kodlama standartları vb. durumlardan dolayı öngörülemeyen durumlarda, sonuçların doğruluk seviyesi istenilen düzeyde olmayabilir.

  Harici geçerliliğin sınanması için önerilen yöntem, farklı kullanıcılarla farklı uygulamaların büyüklüklerinin ölçülmesi işleminde kullanılmalıdır.

- **Güvenilirlik**

  Güvenilirlik, çalışmanın tekrar edilebilir ve aynı koşullarda her seferinde aynı sonuçları üreteceğinin ispat edilmesi ile ilgilidir [60]. Güvenilirliğin amacı, hataları minimize etmek ve çalışmanın doğruluğuna güvenmektir [60].

  Yöntemin tekrar edilebilir oluşu güvenilirliği sağlayan en önemli unsurdur. Farklı kişiler ve araçlar (Selenium vb.) ile önerilen yöntem test edilebilir. Açıklayıcı bilgiler ve kullanım kılavuzu ile ölçümler üçüncü kişiler tarafından da kolaylıkla yapılabilir.

8. SONUÇLAR

Yazılım büyüklüklerinin işlevsel olarak ölçülmesinin proje yönetimi açısından önemi gün geçtikçe artmaktadır. Ancak işlevsel büyüklüklerin manuel olarak ölçülmesi oldukça maliyetli bir süreçtir. Bu noktada, ölçümün otomatikleştirilmesi fikri önem kazanmaktadır. İşlevsel büyüklük ölçümün otomatikleştirilmesi işlemi, tekrarlamalı ya da çevik yazılım geliştirme yaklaşımını benimseyen yazılım projelerinde, Sprint ler arasında geliştirilen koda bakıp işlevsellik miktarı belirleyerek proje planlamasını gözden geçirmede yararlı olabilir. Ayrıca yazılım kurumları için, tarihsel büyüklük veritabanları ("histrorical size databases") oluşturularak gelecek projelerin planlanması işleminde faydalı olabilir.

Bu tez çalışması kapsamında, üç katmanlı mimariye sahip olan Java iş uygulamaları için COSMIC işlevsel büyüklüğün otomatik olarak hesaplanması için geliştirilen model ve yazılım bileşenleri anlatılmakta ve örnek bir uygulamada önerilen modelin kullanımı anlatılmaktadır. Önerilen model, Measurement Kütüphanesi'nin geliştirilmesi ve Ayrıştırma İşleminin uygulanması olarak iki kısımdan oluşmaktadır. Önerilen model sonucu gerçekleştirilen otomatik ölçüm sonuçları ilk olarak manuel olarak yapılan ölçümle karşılaştırılmıştır. Otomatik ölçülen ve manuel olarak hesaplanan COSMIC işlevsel büyüklüklerinin %96 oranında yakınsadığı görülmüştür. Ayrıca, otomatik olarak yapılan ölçüm süresi manuel olarak yapılan ölçüm süresinin 1/27 si olarak yaklaşık 10 dakika sürmüştür. Bu sonuçlar, geliştirilen modelin doğruluk oranının yüksek olduğunu ve maliyet-etkin bir ölçme modeli olduğunu göstermektedir.

Otomatik ölçümün ayrıca Erken ve Hızlı yöntemiyle de karşılaştırılması yapılmıştır. Bu kapsamda, seçilen modüldeki gereksinimlerin işlevsel büyüklüklerinin otomatik olarak hesaplanmış hali, Erken ve Hızlı yöntemiyle elde edilen sonuçlarla karşılaştırılmıştır. Erken ve Hızlı yöntemi kullanım durumları üzerinden giderek bir tahminleme yapmakta iken, önerilen model direkt olarak kod üzerinden yola çıkarak bir tahminlemede bulunmaktadır. Kodun her zaman güncel olduğunu ve gereksinimlerin kodla birlikte paralelde her zaman güncel tutulamayacağını göz önüne alırsak, geliştirme aşamasında önerilen model ile daha doğru işlevsel büyüklük tahmininde bulunabileceği söylenebilir. Bu noktada önerilen modelin sunduğu yaklaşım, daha doğru sonuçlar alınmasında etkili olacaktır. Bu fikir, örnek

uygulamanın gerçekleştirildiği kurumdaki Kalite Birimi ile paylaşılmış ve olumlu geri dönüşler alınmıştır. Örnek uygulamayı geliştiren ekibe yapılan anket sonucunda da büyüklük hesaplama işleminin COSMIC metotu kullanılarak otomatikleştirilmesi fikrinin iyi olabileceği sonucu çıkmıştır. Tüm bu değerlendirme ve yorumlar önerilen modelin etkinliğini ve kullanılabilirliğini desteklemektedir.

Çalışmanın ana amacı olan otomatikleştirme işleminin yanı sıra, önerilen yöntemin birçok avantajı bulunmaktadır. Bunlar, ölçme işleminde ölçerin etkisini azaltmak, manuel ölçümün maliyetini yok etmek, seçilen kullanım durumlarından yola çıkarak (senaryolar doğrultusunda ilgili arayüzlerde gezinerek) işlevsel büyüklüğü hesaplamak vb. dir. Yapılan ölçüm sonuçları sayesinde, COSMIC işlevsel büyüklüğün yüksek çıktığı yerlerdeki kısımların yeniden gözden geçirilerek düzeltilmeye ihtiyaç olduğu ortaya çıkmıştır. Gereksiz veritabanı sorgulamaları ve benzer işleve sahip kod parçaları bu yöntem sayesiyle farkedilmiştir. Ayrıca, Erken ve Hızlı Yöntemi vb. yöntemler ile sadece kullanım durumlarından yola çıkarak hesaplanan sonuçların bir süre sonra güncelliklerini yitirdiği ve kullanım durumları eksik olan durumlarda da işlevsel büyüklüklüklerin hesaplanmadığı görülmüştür. Bu noktada, önerilen yöntemin bu sorunlara bir çözüm olduğu görülmektedir.

Gelecek çalışmalarda, doğrulama bulgularında bulunan değerlere göre Ayrıştırma İşleminin geliştirilmesi ve farklı uygulamalarda da ölçümün yapılarak doğruluğunun değerlendirilmesi amaçlanmaktadır. Ayrıca, farklı teknolojiler kullanılarak geliştirilen yazılımlar da baz alınarak Ayrıştırma İşlemi genişletilerek iyileştirilebilir. Ölçüm sonuçları daha basit metotlar (kullanım durumu puanlama sayma araçlar, verilen ortamdaki işlev puan bazına ortalama kod satır sayısı miktarı vb.) ile karşılaştırılabilir. Ek olarak, E&Q yöntemiyle tahminde bulunan sonuçlar ile otomatik ölçme yöntemi ile elde edilen sonuçlar arasında bir ilişki bulunabilir ve veri hareketleri bazında bir değerlendirme yapılabilir.

KAYNAKLAR

[1] Big, T., Small, A. CHAOS MANIFESTO 2013: Think Big, Act Small. The Standish Group International, 1–52, 2013. Retrieved from http://www.standishgroup.com (Mayıs, **2015**).

[2] Albrecht A. J., "Measuring Application Development Productivity," in *Proceedings of the Joint SHAREGUIDEIBM Application Development Symposium*, vol. 83, no. 408, pp. 83–92, **1979**.

[3] Brandon Jr, D. M. Implementing Earned Value Easily And Effectively. *Project Management Journal*, 29(2), 11–18, June **1998**.

[4] Garcia, C. A. L., & Hirata, C. M. Integrating functional metrics, COCOMO II and earned value analysis for software projects using PMBoK. In *Proceedings of the 2008 ACM symposium on Applied* computing - SAC '08 (p. 820). ACM Press, **2008**.

[5] Li, J. H., Wei, C. J., Li, J., & Li, Q. Earned value project management of model-centric software development. In 2008 International Conference on Wireless Communications, Networking and Mobile Computing, WiCOM, **2008**.

[6] Anbari, F. T. Earned value project management method and extensions. *Project Management Journal*, 34(4), 12–23, **2003**.

[7] Pow-Sang, J. A., & Jolay-Vasquez, E. An approach of a technique for effort estimation of iterations in software projects. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC* (pp. 367–374), **2006**.

[8] Pow-Sang, J. A., & Imbert, R. Effort estimation in incremental software development projects using function points. In *Communications in Computer and Information Science* (Vol. 340 CCIS, pp. 458–465), **2012**.

[9] Hussain, I., Kosseim, L., & Ormandjieva, O. Approximation of COSMIC functional size to support early effort estimation in Agile. *Data and Knowledge Engineering*, *85*, 2–14, **2013**.

[10] Editors, S. "Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)," **2011**.

[11] International Standarts Organization and International Electrotechnical Commission, "Mk II function point analysis," Counting Practices Manual, ISO/IEC 20968:2002, **2002**.

[12] International Standarts Organization and International Electrotechnical Commission, "NESMA functional size measurement method version 2.1," Definitions and counting guidelines for the application of Function Point Analysis, ISO/IEC 24570:2005, **2005**.

[13] International Standarts Organization and International Electrotechnical Commission, "IFPUG functional size measurement method," ISO/IEC20926:2009, **2009**.

[14] International Standarts Organization and International Electrotechnical Commission, “COSMIC: a functional size measurement method,” ISO/IEC 19761:2011, **2011**.

[15] Common Software Measurement International Consortium, “COSMIC Method Version 4.0, Measurement Manual,” **2014.**

[16] Bajwa S. S., Gencel C., “What are the significant cost drivers for COSMIC functional size based effort estimation?,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5891 LNCS, pp. 62–75, **2009**.

[17] Akca, A. A., Tarhan, A., “Run-time measurement of COSMIC functional size for Java business applications: Is it worth the cost?,” *Proc. - Jt. Conf. 23rd Int. Work. Softw. Meas. 8th Int. Conf. Softw. Process Prod. Meas. IWSM-MENSURA 2013*, pp. 54–59, **2013**.

[18] Robiolo, G., “How Simple is It to Measure Software Size and Complexity for an IT Practitioner?,” *2011 Int. Symp. Empir. Softw. Eng. Meas.*, pp. 40–48, **2011**.

[19] Bévo, V., Lévesque, V., and Abran, A., “Application de la méthode FFP à partir d’une spécification selon la notation UML : Compte Rendu Des Premıers Essaıs D'applıcatıon Et Questıons,” In: Proc. 9th Int. Workshop Soft. Measurement. Lac Supérieur, Canada pp. 230-242, **1999**.

[20] Fehlmann, T. M., Kranich, E., “COSMIC Functional Sizing based on UML Sequence Diagrams,” p. 16, **2011**.

[21] Lavazza, L., Del Bianco, V., “A case study in COSMIC functional size measurement: The rice cooker revisited,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5891 LNCS, pp. 101–121, **2009**.

[22] Levesque, G., Bevo, V., and Cao, D. T., “Estimating software size with UML models,” *Proc. 2008 C3S2E Conf. - C3S2E ’08*, p. 7, **2008**.

[23] Akca, A. A., Tarhan, A., “Run-time Measurement of COSMIC Functional Size for Java Business Applications: Initial Results,” *2012 Jt. Conf. 22nd Int. Work. Softw. Meas. 2012 Seventh Int. Conf. Softw. Process Prod. Meas.*, pp. 226–231, Oct. **2012**.

[24] Sag, M. A., Tarhan, A., “Measuring COSMIC Software Size from Functional Execution Traces of Java Business Applications,” *2014 Jt. Conf. Int. Work. Softw. Meas. Int. Conf. Softw. Process Prod. Meas.*, pp. 272–281, **2014**.

[25] CMMI Product Team, “CMMI® for Development, Version 1.3 CMMI-DEV, V1.3 - Improving processes for developing better products and services,” **2010**.

[26] Fenton, N., & Bieman, J. *Software metrics: a rigorous and practical approach*. CRC Press, **2014**.

[27] IEEE, “IEEE Standart Glossary of Software Engineering Terminology”, ISBN 1- 55937-067, September 28, **1990**.

[28] S. Deeds-rubin, “A SLOC Counting Standard,” pp. 1–16. (Mayıs, **2015**).

[29] Rosenberg, J. Some misconceptions about lines of code. Proceedings Fourth International Software Metrics Symposium, 137–142. **1997.**

[30] Galorath, D. D., & Evans, M. W. Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves, **2006.**

[31] Nassif, A. B., Capretz, L. F., & Ho, D. Software estimation in the early stages of the software life cycle. International Conference on Emerging Trends in Computer Science, Communication and Information Technology, January, 5–13. **2010.**

[32] Kurmanadham V.V.G.B. Gollapudi, “Function Points or Lines of Code? – An Insight”, Global Microsoft Business Unit, Wipro Technologies, **2005**.

[33] Ross, M., Size Does Matter : Continuous Size Estimating and Tracking.Quantitative Software Management. Retrieved from http://www.qsm.com/ (Mart, **2015**).

[34] Park, R. E. Software Size Measurement: A Framework for Counting Source Statements (CMU/SEI-92-TR-20, ADA 258 304). *Pittsburgh PA: Software Engineering Institute, Camegie Mellon University,* Sept. **1992**.

[35] Çevik Yazılım Geliştirme “AGILE”, ACM, **2015.**

[36] Cohn, M. *User Stories Applied: For Agile Software Development*. *Writing* (Vol. 1). Addison-Wesley Professional, **2004**.

[37] R. K. Clemmons, “Project Estimation With Use Case Points,” *J. Devense Softw. Eng.*, **2006**.

[38] Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. *HarlowEssex England Addison* (Vol. 2640). Addison-Wesley, **1992**.

[39] Karner, G. Resource estimation for objectory projects. *Objective Systems SF AB*, 1–9, **1993**.

[40] Tommaso, I., Roberto, M., Franco, P. Early & Quick Function Points® v3.0: enhancements for a Publicly Available Method, **2008**.

[41] Conte, M., Iorio, T., Meli, R., Dpo, L. S., & Santillo, L. E&Q: An Early & Quick Approach to Functional Size Measurement Methods. *Software Measurement European Forum (SMEF)*, **2004**.

[42] Santillo, L., Conte, M., & Meli, R. Early & Quick Function Point: Sizing More with Less. *11th IEEE International Software Metrics Symposium (METRICS’05)*, 5–10, **2005**.

[43] Gencel, C., & Buglione, L. Do base functional component types affect the relationship between software functional size and effort? *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4895 LNCS*, 72–85, **2008**.

[44] Albrecht, A. J. Software Function , Source Lines of Code , and Development Effort Prediction : A Software Science Validation, (6), **1983**.

[45] Symons, C. R. Function Point Analysis: Difficulties and Improvements. *IEEE Transactions on Software Engineering*, *14*(1), 2–11, **1998**.

[46] Nesma Official Site, Sizing, http://nesma.org/themes/sizing/ (Nisan, **2015**).

[47] Wikipedia, http://en.wikipedia.org/wiki/Project_management (Nisan, **2015**).

[48] Fleming, Q., & Koppelman, J. Earned value project management, (July), 19–23, **2000**.

[49] Poels, G., *Functional Size Measurement Method for Event-Based Object-Oriented Enterprise Models.* Proceedings of the 4th International Conference on Enterprise Information Systems–ICEIS, pp. 667–675, Ciudad Real, Spain, **2002**.

[50] Diab, H., Koukane, F., Frappier, M., & St-Denis, R. µ cROSE: Automated measurement of COSMIC-FFP for Rational Rose RealTime. *Information and Software Technology*, *47*(June 2004), 151–166, **2005**.

[51] Ferrucci, F., Gravino, C., & Di Martino, S. Estimating web application development effort using Web-COBRA and COSMIC: An empirical study. *Conference Proceedings of the EUROMICRO*, 306–312. **2009.**

[52] Hussain, I., Kosseim, L., & Ormandjieva, O. Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining. *Proceedings of the Natural Language Processing and Information Systems, and 15th International Conference on Applications of Natural Language to Information Systems*, 80–91, **2010**.

[53] Lind, K., & Heldal, R., "A practical approach to size estimation of embedded software components," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1–15, **2012**.

[54] Del Bianco, V., & Lavazza, L. Applying the COSMIC functional size measurement method to problem frames. *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, 282–290, **2009**.

[55] Buglione, L., Ormandjieva, O., & Daneva, M. Using PSU for early prediction of COSMIC size of functional and non-functional requirements. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5338 LNCS*, 352–361, **2008**.

[56] Oracle Official Site, JavaServer Faces Technology, http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html (Nisan, **2015**).

[57] Overview of JavaServer Faces (JSF) Technology, https://techcommunity.softwareag.com/pwiki/-/wiki/Main/JavaServer+Faces+and+webMethods+CAF+Applications/pop_up

;jsessionid=AE517E45F544D61E24A704A69F9FC182?controlPanelCategory=portlet_36 (Nisan, **2015**).

[58] Serkan Sakınmaz Blog, JSF - 04 - Managed Bean ve Scope Kavramı, http://serkansakinmaz.blogspot.com.tr/2013/11/jsf-managed-bean-ve-scope-kavram.html (Nisan, **2015**).

[59] Soubra, H., Abran, A., and Ramdane-Cherif, A., "Verifying the Accuracy of Automation Tools for the Measurement of Software with COSMIC -- ISO 19761 Including an AUTOSAR-Based Example and a Case Study," *2014 Jt. Conf. Int. Work. Softw. Meas. Int. Conf. Softw. Process Prod. Meas.*, pp. 23–31, **2014**.

[60] Yin, R. K., Case Study Research: Design and Methods, Applied Social Research Methods Series, vol. 5. **2009**.

EK 1 : MEASUREMENT LIBRARY VE AYRIŞTIRMA İŞLEMİ KULLANIM KILAVUZU

Önerilen model ile COSMIC büyüklüğün çalışma zamanında otomatik olarak hesaplanması için bazı adımlar bulunmaktadır. Bu kısım, Measurement Library ve Ayrıştırma İşlemi kullanımını detaylı olarak anlatmayı hedeflemektedir.

1) Measurement Kütüphanesinin Uygulamaya Tanıtılması

Uygulama eğer Apache Maven 'Java Deployment Tool' u kullanıyor ise, projenin pom.xml dosyalarına aşağıda belirtildiği gibi Measurement Library kütüphanesini tanıtmak gerekmektedir.

```
<dependency>
        <groupId>measurementLibrary</groupId>
        <artifactId>measurementLibrary</artifactId>
        <version>1.0</version>
</dependency>
```

Eğer uygulama, Maven i kullanmıyor ise de, 'Referenced Library' olarak Measurement Kütüphanesini eklemek yeterli olacaktır.

2) Ayrıştırma İşlemi Kodunun Çalıştırılması

Ayrıştırma İşlemi bir .java dosyasıdır. Ayrıştırma İşlemi kodu çalıştırılırken, girdi olarak proje ile ilgili bazı bilgiler gereklidir. Bunlar:

a) Projenin bulunduğu dizin
b) Servis ve Dao Sınıflarını içeren jar dosyası
c) İstemci Sınıflarını içeren jar dosyası

Yukarıda istenilen veriler Ayrıştırma İşlemi aşamasında Managed Bean, Model, Servis ve Dao Sınıflarına erişmek amacıyla gereklidir.

3) Arayüzde Tetikleme İşlemlerinin Yapılması

Önerilen modelde otomatik hesaplama işlemi çalışma zamanında kullanıcı senaryolarına bağlı olarak gerçekleşir. Bir butonun tıklanması buna en iyi örnektir. Böylece Ayrıştırma İşlemi Kodu ile uygulamanın kaynak koduna otomatik olarak eklenen kütüphane metotu çağrılmış olunur ve süreçlerin kaydedilmesi işlemi

başlar. Ölçümün doğruluğu açısından, aynı anda birden fazla kişi ölçüm yapmamalıdır.

4) Sonuçların Elde Edilmesi

Sonuçlar, sistemden 'Çıkış' yapıldıktan sonra Word dokümanında result.docx adı altında oluşturularak kullanıcıya gösterilir. Örnek olarak, aşağıda verildiği gibi bir sonuç dosyası alınır.

COSMIC HESAPLAMA SONUÇLARI

Toplam COSMIC Sayısı: X
Toplam Entry(E) Sayısı: X1
Toplam Read(R) Sayısı: X2
Toplam Write(W) Sayısı: X3
Toplam Exit(X) Sayısı: X4

Toplam İşlevsel süreç zamanı: u dakika v saniye
Toplam Entry İşlevsel süreç zamanı: w saniye
Toplam Read İşlevsel süreç zamanı: x saniye
Toplam Write İşlevsel süreç zamanı: y saniye
Toplam Exit İşlevsel süreç zamanı: z saniye

Modül bazında COSMIC sayısı

{A Modülü= A, B Modülü=B}

Modül bazında Entry/Read/Write/Exit Sayıları

A Modülü için Entry/Read/Write/Exit Sayıları
Entry Sayısı: A1
Read Sayısı: A2
Write Sayısı: A3
Exit Sayısı: A4
B Modülü için Entry/Read/Write/Exit Sayıları
Entry Sayısı: B1
Read Sayısı: B2
Write Sayısı: B3
Exit Sayısı: B4

EK 2 : ANKET SORULARI VE CEVAPLAR

S#	Soru	Seçenekler ve Cevapların dağılım yüzdesi	Medyan
BÖLÜM-I (Yanıtlayıcı Profili)			
1	İşyerinizdeki göreviniz nedir?	(%61.9) Geliştirme (%4.8) Kalite Güvence (%14.3) Proje Yönetimi (%19) Diğer	
2	Anket konusunun mevcut ilgi alanlarınızla yakınlığı nedir?	(%4.8) Çok az (%23.8) Az (%28.6) Orta (%28.6) Yüksek (%14.3) Çok yüksek	Orta
3	Anketi yanıtlamak için gönüllülük düzeyiniz nedir?	(%0) Çok az (%4.8) Az (%38.1) Orta (%42.9) Yüksek (%14.3) Çok yüksek	Yüksek
4	Yazılım büyüklüğünün proje planlamada kullanımı hakkında bilgi düzeyiniz nedir?	(%4.8) Çok az (%33.3) Az (%47.6) Orta (%14.3) Yüksek (%0) Çok yüksek	Orta
5	Yazılım büyüklüğünün görev yönetiminde kullanımı hakkında bilgi düzeyiniz nedir?	(%9.5) Çok az (%33.3) Az (%47.6) Orta (%9.5) Yüksek (%0) Çok yüksek	Orta
6	Yazılım büyüklüğünün, yazılım hata değerlendirmede kullanımı hakkında bilgi düzeyiniz nedir?	(%4.8) Çok az (%42.9) Az (%33.3) Orta (%19) Yüksek (%0) Çok yüksek	Orta
BÖLÜM-II (Mevcut Durum)			
7	İşyerinizde geliştirdiğiniz yazılımların satır sayısı ölçülüyor mu?	(%14.3) Fikrim Yok (%28.6) Hayır (%57.1) Evet	
7.1	Ölçülen satır sayısı herhangi bir değerlendirmede kullanılıyor mu?	(%4.8) Fikrim Yok (%28.6) Hayır (%52.4) Evet	
7.1.1	Neleri değerlendirmede kullanılıyor? -- uygun seçenek(ler)i işaretleyin	(%57.1) Ürün büyüklüğü (%33.3) Kodun kalitesini (%33.3) Üründeki	

		değişiklik miktarını (%19) Ekip üretkenliğini (%4.8) Yazılımcı üretkenliğini	
8	Otomatik ölçmenin satır sayısı kullanımına etkisini nasıl değerlendiriyorsunuz?	(%4.8) Çok az (%28.6) Az (%52.4) Orta (%4.8) Yüksek (%0) Çok yüksek	Orta
9	Şimdiye kadar geliştirdiğiniz kodun satır sayısını merak edip ölçtünüz mü?	(%28.6) Evet (%61.9) Hayır	
10	Size göre satır sayısı kullanımını etkileyen faktörler nelerdir? -- uygun seçenek(ler)i işaretleyin	(%9.5) Fikrim yok (%52.4) Kurumun ölçme politikası (%42.9) Kullanım amacının bilinmesi (%19) Kullanım amacının proje yönetimini desteklemesi (%47.6) Kullanım amacının kalite yönetimini desteklemesi (%0) Diğer	
11	İşlevsel büyüklük kavramını daha önce duydunuz mu?	(%57.1) Hayır (%38.1) Evet	
11.1	İşlevsel büyüklük kavramına ilişkin bilgi düzeyiniz nedir?	(%23.8) Çok az (%28.6) Az (%14.3) Orta (%0) Yüksek (%0) Çok yüksek	Orta
11.2	COSMIC işlevsel büyüklük kavramını daha önce duydunuz mu?	(%38.1) Hayır (%33.3) Evet	
11.2.1	COSMIC işlevsel büyüklük kavramına ilişkin bilgi düzeyiniz nedir?	(%42.9) Çok az (%9.5) Az (%14.3) Orta (%0) Yüksek (%0) Çok yüksek	Çok az

BÖLÜM-III (Bilgilendirme)			
12	Bu Kısım için lütfen aşağıda verilen bilgilendirme notunu okuyunuz. Sonrasında soruları cevaplamaya devam edebilirsiniz.		
	Bilgilendirme Notu: İşlevsel büyüklük ölçme yöntemleri, yazılımın "İşlevsel Kullanıcı Gereksinimleri"ni kullanarak işlevsel büyüklüğünü hesaplamaya yarar. İşlevsel Kullanıcı Gereksinimleri (İKG), kullanıcı gereksinimlerinin işlevsel özellikler içeren (ekleme, silme, sorgu vb.) alt kümesidir. COSMIC yöntemi, işlevsel büyüklük ölçme yöntemlerinden biridir. Bu yöntem ile mevcut bir yazılımın işlevsel büyüklüğü geriye dönük olarak hesaplanabilir ya da yeni geliştirilecek bir yazılımın işlevsel büyüklüğü tahmin edilebilir. Ölçme yöntemi, işlevsel büyüklüğü ölçülecek yazılımın teknolojik bileşenlerinden ve geliştirilmesi için kullanılan yöntemlerden bağımsız olarak tasarlanmıştır. Veri güçlü sistemler (örn. yönetim bilgi sistemleri), kontrol güçlü sistemler (örn. gerçek zamanli sistemler) ve melez sistemler (örn. havayolu rezervasyon sistemi) için uygulanabilir. Ne var ki yöntemin ölçme kuralları, karmaşık algoritmaların ve video imgelerinin büyüklüğe etkisini hesaba katmaz. COSMIC yöntemi, yazılımın İşlevsel Kullanıcı Gereksinimlerini, bir dizi "İşlevsel Süreç"e ayrıştırır. Her İşlevsel Süreç, veri hareketi tipi (kullanıcı girdisi/çıktısı) veya veri işleme tipi (veritabanı okuma/yazma) icra eden tek ve eşsiz alt-süreçtir. Ugulama sınırının dışında bir aktör tarafından tetiklenir ve tamamlandığında uygulamayı tutarlı durumda bırakır. Tanımı gereği bir İşlevsel Süreç en azından, Giriş+Çıkış ya da Giriş+Yazma tiplerinden oluşmalıdır. İşlevsel Süreç icinde yer alan her tip (Giriş, Çıkış, Okuma, Yazma), 1 COSMIC işlev puan (CIP) olarak sayılır. Büyüklüğü ölçülecek yazılımın varlık-ilişki diyagraminin çıkarılmış olması, COSMIC yönteminin sağlıklı uygulanması için önemlidir. COSMIC yöntemi ile önce ayrıştırılan İşlevsel Süreçlerin büyüklüğü hesaplanır. Daha sonra İKG'lerin büyüklüğü, içerdiği İşlevsel Süreçlerin büyüklüğü toplanarak elde edilir. Tüm yazılımın işlevsel büyüklüğü ise içerdigi İKG'lerin büyüklükleri toplanarak hesaplanır.		

	Örnek İKG: “Bir Çalışanın kayıtları güncellenebilmelidir. Kullanıcının Çalışan’ın adını bilmekte ancak Çalışan numarasını bilmemektedir. Bunun için önce Çalışan bütün kişileri görüntüleyip sonra ilgili kişiyi seçebilmelidir. Seçilen kişinin güncellenebilecek kayıtları görüntülenmelidir.” Bu İKG, ardışık gerçekleşen üç İşlevsel Süreç (IS) içerir: IS-1: Kullanıcı önce isme göre Çalışanlar listesini görüntüler. IS-2: Kullanıcı belirli bir Çalışan’ı seçer ve onunla ilgili bilgileri görüntüler. IS-3: Kullanıcı Çalışan’ın bilgilerini günceller. COSMIC yöntemi ile İşlevsel Sureçlerin ve İKG’nin büyüklüğü aşağıdaki gibi hesaplanir. IS-1: Kullanıcı önce isme göre Çalışanlar listesini görüntüler. Giriş : 1 (‘Listele’ isteği) Yazma : 0 Okuma: 1 (Çalışan tablosu) Çıkış : 1 (Çalışan listesi) IS-1 CIP = 3 IS-2: Kullanıcı belirli bir Çalışan’ı seçer ve onunla ilgili bilgileri görüntüler. Giriş : 1 (Çalısan No) Yazma : 0 Okuma: 1 (Çalışan tablosu) Çıkış : 1 (Çalışana ait veriler) IS-2 CIP = 3 IS-3: Kullanıcı Çalışan’ın bilgilerini günceller. Giriş : 1 (Güncellenmiş Çalışan verileri) Yazma : 1 (Çalışan tablosu) Okuma: 0 Çıkış : 1 (Onay ya da hata mesajı) IS-3 CIP = 3 İKG’nin COSMIC işlevsel büyüklüğü = 3 + 3 + 3 = 9 CIP

BÖLÜM-IV (Keşif)			
13	İşyerinizde geliştirdiğiniz yazılımların işlevsel büyüklük ölçülüyor mu?	(%42.9) Fikrim Yok (%28.6) Hayır (%23.8) Evet	
13.1	İşlevsel büyüklük ne sıklıkla ölçülüyor?	(%57.1) Fikrim Yok (%0) Bir projede (%14.3) Bazı projelerde (%0) Birçok projede	
13.2	İşlevsel büyüklük ölçme için hangi yöntem(ler) temel alınıyor? -- uygun seçenek(ler)i işaretleyin	(%42.9) Fikrim Yok (%19) COSMIC (%14.3) IFPUG (%0) Mark II (%0) Diğer	
13.3	İşlevsel büyüklük manuel olarak mı ölçülüyor?	(%52.4) Fikrim Yok (%0) Hayır (%9.5) Evet	
13.3.1	İşlevsel büyüklüğü manuel ölçmenin maliyetini nasıl değerlendiriyorsunuz?	(%23.8) Fikrim Yok (%0) Çok az (%9.5) Az (%4.8) Orta (%19) Yüksek (%4.8) Çok yüksek	Yüksek
13.4	Ölçülen işlevsel büyüklük herhangi bir değerlendirmede kullanılıyor mu?	(%47.6) Fikrim Yok (%9.5) Hayır (%9.5) Evet	
13.4.1	Neleri değerlendirmede kullanılıyor? -- uygun seçenek(ler)i işaretleyin	(%23.8) Ürün büyüklüğünü (%9.5) Kodun kalitesini (%0) Üründeki değişiklik miktarını (%0) Ekip üretkenliğini (%0) Yazılımcı üretkenliğini (%9.5) Diğer	
14	Size göre geliştirdiğiniz yazılımların işlevsel büyüklük ölçümü için COSMIC yöntemi kullanılabilir mi?	(%9.5) Hayır (%76.2) Evet	
15	Otomatik ölçmenin işlevsel büyüklük kullanımına etkisini nasıl değerlendiriyorsunuz?	(%9.5) Çok az (%9.5) Az (%38.1) Orta (%23.8) Yüksek (%4.8) Çok yüksek	Orta

BÖLÜM-V (Geribildirim)			
16	Bu anketi yanıtlamak hoşuma gitti.	(%23.8) Çok az (%9.5) Az (%33.3) Orta (%19) Yüksek (%4.8) Çok yüksek	Orta
17	Bu anketi yanıtlamak yazılım büyüklüğü ve kullanımları hakkındaki farkındalığımı arttırdı.	(%9.5) Çok az (%19) Az (%23.8) Orta (%23.8) Yüksek (%9.5) Çok yüksek	Orta

ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı : Rana GÖNÜLTAŞ
Doğum Yeri : Ankara
Medeni Hali : Bekar
E-posta : ranagonultas@gmail.com
Adresi : Dikmen Cad. No:318/24 Çankaya/ANKARA

Eğitim

Lisans : 2006-2011 Çankaya Üniversitesi Bilgisayar Mühendisliği (Tam Burslu)
Yüksek Lisans : 2012-2015 Hacettepe Üniversitesi Bilgisayar Mühendisliği

Yabancı Dil ve Düzeyi

İngilizce – İyi Seviyede

İş Deneyimi

2011 Aralık - Halen: Araştırmacı, TÜBİTAK-BİLGEM - Yazılım Teknolojileri Araştırma Enstitüsü (YTE)

Deneyim Alanları

Veritabanı (SQL) ve Web tabanlı yazılım geliştirme, yazılım test konularında bilgi sahibi.

Tezden Üretilmiş Projeler ve Bütçesi

Yok

Tezden Üretilmiş Yayınlar

R.Gönültaş ve A.Tarhan. Run-time Calculation of COSMIC Functional Size via Automatic Installment of Measurement Code into Java Business Applications, The 41th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015) (*Kabul edildi, basımda*).

Tezden Üretilmiş Tebliğ ve/veya Poster Sunumu ile Katıldığı Toplantılar

Yok