# JAVA İŞ UYGULAMALARI İÇİN YAZILIM İŞLEVSEL BÜYÜKLÜĞÜNÜN İŞLETİM İZLERİNDEN ÖLÇÜLMESİ

# MEASURING FUNCTIONAL SIZE OF JAVA BUSINESS APPLICATIONS FROM EXECUTION TRACES

**MUHAMMET ALİ SAĞ**

**YRD. DOÇ. DR. AYÇA TARHAN**

**Tez Danışmanı**

Hacettepe Üniversitesi

Lisansüstü Eğitim – Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Mühendisliği Anabilim Dalı İçin Öngördüğü

YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2014

Muhammet Ali SAĞ' ın hazırladığı **"Java İş Uygulamaları İçin Yazılım İşlevsel Büyüklüğünün İşletim İzlerinden Ölçülmesi"** adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**' nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. İlyas ÇİÇEKLİ ……
Başkan

Prof. Dr. Hayri SEVER ……
Üye

Doç. Dr. Altan KOÇYİĞİT ……
Üye

Yrd. Doç. Dr. Ahmet Burak Can ……
Üye

Yrd. Doç. Dr. Ayça TARHAN ……
Danışman

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

Prof. Dr. Fatma SEVİN DÜZ
Fen Bilimleri Enstitüsü Müdürü

# 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 üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

…/10/2014

MUHAMMET ALİ SAĞ

# ÖZET

## JAVA İŞ UYGULAMALARI İÇİN YAZILIM İŞLEVSEL BÜYÜKLÜĞÜNÜN İŞLETİM İZLERİNDEN ÖLÇÜLMESİ

**Muhammet Ali SAĞ**

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Ayça TARHAN

Ekim 2014, 113 sayfa

Yazılım büyüklüğünün ölçülmesi, geliştirme sürecinin tüm evrelerinde, artan yazılım büyüklükleri sebebiyle giderek karmaşıklaşan proje yönetim süreçleri için büyük önem taşımaktadır. Manuel olarak gerçekleştirilen büyüklük ölçme işlemlerinde ölçüm yapan uzmanın bireysel etkisi ölçümlerde farklı sonuçlara yol açabilmektedir. Diğer yandan ölçümlerin gerektirdiği maliyet ve zaman gibi problemlerin yanı sıra ölçmeye girdi olabilecek dokümantasyon eksikliği göz önüne alındığında, ölçme işleminin otomatikleştirilmesi konusu ön plana çıkmıştır.

Çalışma kapsamında; COSMIC işlevsel büyüklük yöntemi kuralları çerçevesinde, Java iş uygulamalarının işlevsel süreçlerinin tespiti ve yazılım işlevsel büyüklüğünün otomatik olarak hesaplanması hedeflenmiştir. Yöntem işlevsel büyüklüğü, çalışma zamanında yapılan dinamik analiz yöntemiyle yazılımdan elde edilecek UML Dizge Diyagramları üzerinden hesaplamaktadır. Dinamik analiz için bağlam yönelimli programlama yönteminin Java platformundaki gerçekleştirimi olan AspectJ kullanılmaktadır. Yöntem, yazılım kodunda herhangi bir değişiklik gerektirmeden uygulanabilmektedir. Büyüklüğü hesaplanacak yazılımın işlevsel süreçleri arayüzden tetiklenen olaylar aracılığıyla tespit edilmektedir. Tetiklenen olayların işletim izleri AspectJ ile metin biçeminde dizge diyagramlarına

dönüştürülmektedir. Daha sonra bu dizge diyagramları üzerinden veri hareketleri tespit edilerek COSMIC işlevsel büyüklük hesaplanmaktadır.

Önerilen yöntemi desteklemek için ‘COSMIC Solver’ adlı bir prototip araç geliştirilmiştir. Hedefler çerçevesinde yöntemin işe yararlılığı örnek bir ön uygulama ve bir durum çalışması üzerinden doğrulanmıştır. Durum çalışmasında çerçevesi prototipe uygun olarak seçilen Java uygulamasının işlevsel büyüklüğü manuel olarak ve önerilen yöntem/araç kullanarak ölçülmüş ve elde edilen sonuçlar değerlendirilmiştir. Ölçümlerde elde edilen sonuçların birbirine yakınsadığı ve hesaplamada otomasyonla tezde hedeflenen oranın (%80+) üstünde netlik elde edilebildiği görülmüştür. Ayrıca otomatikleştirmenin doğal bir sonucu olarak zaman maliyetinde büyük kazanç (1/10 oranında) sağlanabildiği gözlemlenmiştir.

**Anahtar Kelimeler:** İşlevsel büyüklük ölçümü, COSMIC yöntemi, AspectJ, UML, Dizge Diyagramları, otomatik işlevsel büyüklük ölçümü, dinamik analiz

# ABSTRACT

## MEASURING FUNCTIONAL SIZE OF THE JAVA BUSINESS APPLICATIONS FROM EXECUTION TRACES

**Muhammet Ali SAĞ**

Master of Science, Computer Science

Supervisor: Yrd. Doç. Dr. Ayça TARHAN

October 2014, 113 pages

During all phases of software development, software size measurement has great importance for project management processes that are gradually becoming more complex because of growing sizes of software. Due to individual effect of measurement specialist, manually operated size measurements might lead to different results in measured values. On the other hand, considering the cost and time problems and the lack of documentation to input to size measurement, the issues of automatizing process of measurement came to the front.

In this study, it is aimed to automatically compute software functional size within the scope of COSMIC functional size methodology in case of detecting functional processes and measuring the size of Java business applications. The method measures functional size using UML Sequence Diagrams that are produced from software with dynamic analysis method at runtime. For dynamic analysis, AspectJ as an implementation of aspect oriented programming methodology on Java platform is being used. The method could be applied without any changes in software code. Functional processes are detected by interactions that are produced by using graphical user interfaces of the software. With the help of AspectJ, execution traces are converted to sequence diagrams which are in text

format. Then, using that sequence diagrams, the COSMIC functional size is measured by using data movements.

In order to support proposed method, a prototype tool called ' COSMIC Solver', was developed. Considering the goals, the proposed method's usefulness was tested with an example application and a case study. In the case study, in addition to using manual measurement, proposed method/tool was used to measure the size of a Java application that is chosen from a public library as conformant to the prototype, and results are evaluated. It is observed that the results of measurements are converging to each other and that the accuracy of the value calculated by the prototype exceeds the targeted value in this study (80%+). Moreover, it is observed that a natural result of automatizing is a great time saving which is about 10 times.

**Keywords**: Functional Size Measurement, COSMIC method, AspectJ, UML, Sequence Diagrams, automatic functional size measurement, dynamic analysis

# TEŞEKKÜR

Tez konusunun belirlenmesini sağlayan, tez çalışmasını yönlendiren, tez metnini içerik ve biçim bakımından inceleyerek şekillendiren Sayın Yrd. Doç. Dr. Ayça TARHAN' a sonsuz teşekkürler…

# İÇİNDEKİLER

# ŞEKİLLER DİZİNİ

# ÇİZELGELER DİZİNİ

# SİMGELER ve KISALTMALAR

| | |
|---|---|
| AOP | Aspect Oriented Programming(Bağlam Yönelimli Programlama) |
| API | Application Programming Interface |
| Ark | Arkadaşları |
| COSMIC | Common Software Measurement International Consortium |
| E | Entry(Girdi) |
| FFP | Full Function Point (Tam İşlevsel puan) |
| FISMA | Finnish Software Measurement Association |
| FP | Functional Process (İşlevsel Süreç) |
| FPA | Function Point Analysis (İşlevsel Puan Analizi) |
| FSM | Functional Size Measurement (İşlevsel Büyüklük Ölçümü) |
| FUR | Functional User Requirements (İşlevsel Kullanıcı Gereksinimleri) |
| GUI | Graphical User Interface (Grafiksel Kullanıcı Arayüzü) |
| IEEE | Institute of Electrical and Electronics Engineers |
| IFPUG | International Function Point Users Group |
| İKG | İşlevsel Kullanıcı Gereksinimleri |
| ISO | International Standardization Organization |
| JPA | Java Persistence API |
| MIS | Management Information System (Bilgi Yönetim Sistemleri) |
| MOF | Meta Object Facility |
| MUB | Method Update Bulletin (Metod Güncelleme Bülteni) |

| | |
|---|---|
| NESMA | Netherlands Software Metrics Association |
| OMG | Object Management Group |
| OO | Object Oriented (Nesne Tabanlı) |
| R | Read (Okuma) |
| UML | Unified Modeling Language (Birleşik Modelleme Dili) |
| W | Write (Yazma) |
| X | Exit (Çıktı) |

# 1. GİRİŞ

## 1.1. Problem Tanımı

“İşlevsel Büyüklük Ölçümü” yazılım proje yönetimi süreçlerinde önem arz eden bir konudur. Bilgi sistemlerinin süreç yönetimlerinde spesifikasyonların belirlenmesi, geliştirimi ve gerçekleştirimlerinin etkin şekilde yapılabilmesi için, ilgili sistemlerin işlevsel büyüklük bilgilerinin elde edilmesi ve etkin şekilde kullanılabilmesi gereklidir.

ISO 14143-1 [12] standardında yer alan tanıma göre, işlevsel büyüklük ölçümü; bilgi sistemi gereksinimlerinin kullanıcı perspektifinde işlevsel eylemler olarak belirlenmesidir. Albrecht ve ark. [2] ’nın yaptıkları çalışma ve önerdikleri yöntem sonrasında, işlevsel büyüklük ölçümü İşlev Puan Analizi ile gerçekleştirilmektedir. İşlev puanı, paydaşlar için kavramsal olarak işlevsel terimleri kullanan bir yazılım metriğidir.

Yazılım geliştirme faaliyetlerinin ilk adımlarından itibaren işlevsel büyüklüğün ölçülebilir olması, ölçme işlemini, artan yazılım boyutları nedeniyle kritik önem kazanan yazılım proje yönetimi sürecinin en önemli bileşenlerinden biri haline getirmiştir. Diğer yandan geliştirici yeteneği, tasarım tekniği ya da programlama dili değişse dahi işlevsel büyüklük puanının değişmiyor olması işlevsel puanı diğer büyüklük ölçümlerinden pozitif ayrıştırmakta ve öne çıkarmaktadır.

Bütün bunlara rağmen, işlevsel büyüklük ölçümü yöntemlerinin henüz çözüme kavuşturulmamış problemleri bulunmaktadır. Bu problemlerden biri aynı organizasyonda ve aynı üründe olsa dahi farklı uzmanlar tarafından yapılan ölçümlerde elde edilen sonuçların insan faktörü nedeniyle birbirinden farklılık gösterebilmesidir [8]. Diğer yandan ölçme eylemi için gerekli uzmanlık ve eylemin oluşturduğu ekstra maliyet problemi, işlevsel büyüklük ölçümü yöntemlerinin endüstride yaygınlık kazanmasını engellemektedir. Belirtilen problemlere bir çözüm getirebilmek ve öznel yargılamaya neden olan insan faktörünü minimize ederek tutarlı sonuçlar elde edebilmek için ölçüm işleminin otomatikleştirilmesi bir zorunluluk olarak öne çıkmaktadır [2] [3] [4].

Bu çalışmanın hedefi Java iş uygulamaları için COSMIC işlevsel büyüklük ölçme eyleminin otomatikleştirilmesidir. Bu kapsamda, uygulama kodu üzerinde çalışarak

işlevsel süreçlerin tespit edilmesi ve bu işlevsel süreçlere ait veri hareketlerinin oluşturduğu toplam işlev puanın bir uzmana ihtiyaç duymadan gerçekleştirilmesi için gerekli çalışmalar yapılmıştır.

## 1.2. Tez Çalışmasının Hedefi ve Kapsamı

Tez çalışması kapsamında iki/üç katmanlı Java iş uygulamalarının dinamik analiz yöntemiyle çalışma zamanında elde edilen dizge diyagramlarını kullanılarak işlevsel süreçlerinin ve büyüklüğünün otomatik olarak hesaplanması hedeflenmiştir.

İşlevsel büyüklük hesaplama yaklaşımlarında “COSMIC” yöntem kullanılmıştır.

Dinamik analiz yöntemiyle çalışma zamanında işlevsel kullanıcı eylemleriyle belirlenen işlevsel süreçlerin iş akışlarıyla çalışılmıştır.

Yapılan çalışmada alttaki adımlar izlenmiştir;

Araştırma:

- Tez çalışması kapsamında kullanılacak olan COSMIC İşlevsel büyüklük ölçme yöntemi ile ilgili ölçme prensipleri, kuralları ve ilgili çalışmaların incelenmesi.
- Tez hedefi olan otomatik işlevsel büyüklük hesaplama konusunda yapılmış çalışmaların incelenmesi, uygulanabilirliğin araştırılması.
- Uygulama kod analizi ile işlevsel süreçlerin tespitinin yapılabilirliğinin araştırılması
- UML diyagramları üzerinden COSMIC işlevsel büyüklüğün hesaplanması konusunda yapılan çalışmaların araştırılması.
- Yazılımlardan UML dizge diyagramlarının elde edilebilmesinin yollarının araştırılması.

Gerçekleme:

- Otomatik ölçme işlemi için uygulamalardan UML dizge diyagramları formunun elde edilmesi için çalışmaların yapılması.
- UML dizge diyagramları üzerinde COSMIC kurallarını uygulayarak işlev puanı hesaplayacak yazılımın geliştirilmesi

Geçerleme:

- Önerilen yöntemi test etmek için daha önce geliştirilmiş ve uzman tarafından işlev puanı hesaplanmış bir Java iş uygulamasının, geliştirilen prototip ile test edilmesi.
- Örnek uygulama üzerinden manuel ve otomatik yöntem ölçme sonuçlarının karşılaştırılması.
- Durum çalışması ile geliştirilen prototipin daha geniş kapsamda test edilmesi ve sonuçların karşılaştırılması.
- Manuel ve otomatik yöntemin hesaplama sürelerinin kaydedilerek karşılaştırılması.

### 1.3. Tez Çalışmasının Uygulama Alanı

Tez çalışması kapsamında önerilen yöntem iş dünyası için birçok farklı şekilde faydalı olup, çeşitli kullanım alanları oluşturmaktadır. Bunlardan bazıları;

- Modernizasyon Projeleri: Koddan COSMIC büyüklüğün hesaplanması yazılım modernizasyon projelerinde oldukça ihtiyaç duyulan bir girdi üretmek için kullanılabilir. Halihazırda çalışmakta olan, farklı bir ekip tarafından ve farklı bir dilde geliştirilmiş yazılımın kurumsal ihtiyaçları karşılamaması nedeniyle başlatılan yazılım modernizasyon projesi iş gücü planlamalarında işlevsel büyüklük en iyi girdi olarak iş görecektir.
- Veri Kütüphaneleri: COSMIC büyüklük hesaplamasının kurumsal olarak kullanılmaya karar verilmesinin ardından geriye dönük olarak tamamlanmış projelere ait bilgi tabanı oluşturma hedefinden hareketle, COSMIC büyüklük ve kurumsal yeteneklerin eşleştirilerek bir *Benchmark* veri kütüphanesi oluşturmada kullanılabilir.
- Performans Değerlendirmeleri: Proje iş çıktılarının takibi ve geliştirilmesi hedefiyle, geliştirme aşamasında üretkenlik (büyüklük/işgücü) gibi performans ölçümlerini elde etmek için kullanılabilir [26].
- Proje Yönetim Takibi: Büyük yazılım projeleri geliştirilme sürecinde ulaşılan büyüklüğün sürekli takip edilebilmesi için gerekli iş gücünün minimize edilerek otomatik bir şekilde yapılabilmesine olanak sağlayacaktır. Bu da kestirim doğruluğu takibinin yanı sıra maliyet-işgücü-zaman üçlüsü

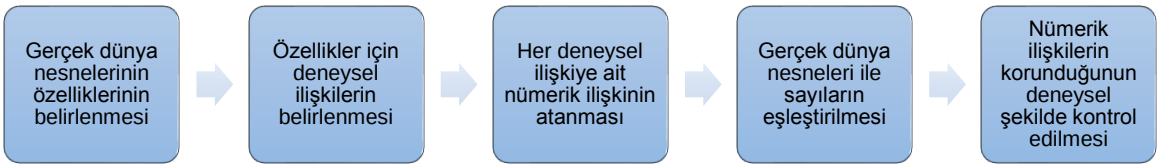
kapsamında sağlayacağı katkı nedeniyle proje yönetim süreçlerinde kullanılabilir.

# 2. ÖN BİLGİ

## 2.1. Ölçme Kavramı

Günlük hayatımızın bir parçası olan ölçme işlemi her ne kadar farkında olmasak da her an gerçekleştirdiğimiz bir eylemdir. Karşıdan karşıya geçerken gelip geçen arabaların hızından tutun, pazarda meyve alırken, evden işe, işten eve geçerken harcayacağımız zamana kadar hayatımızın her anında ölçme işleminden yararlanırız. Ölçme işlemi; çevremizi, dünyamızı anlamamız, iletişim kurmamız gibi temel yaşam faaliyetlerinin yanı sıra, yaşamımızı iyileştirme, hareketlerimizi planlama gibi konularda bizlere yardımcı olan bir eylemdir [27].

Ölçme, gerçek dünyadaki varlıklara ait nitelikleri açıkça belirlenmiş kurallara göre tanımlama amacıyla bu varlıklara ait niteliklere sayıların ve sembollerin atanması sürecidir [27]. Bu eylem sayesinde karşılaştırma ve değerlendirme yapabilmemiz mümkün hale gelmektedir. Bir cismin uzunluğunu ve kısalığını, bir evin büyüklüğünün ve küçüklüğünü, bir cismin uzaklığını ve yakınlığını, bir sürenin uzunluğunu ve kısalığını ölçme sayesinde bu kavramlara yüklenen anlamlar ile değerlendirmekte ve kategorize edebilmekteyiz.




Şekil 1 Standart ölçme eylemi temel aşamaları [27]

Birçok disiplinde yapılan çalışmalar ölçme işleminden bağımsız şekilde düşünülemez. Bir mimarın, fizikçinin veya ekonomistin yaptığı çalışmalar ilgili metriklerden olmaksızın gerçekleştirilemez [27]. Fakat yazılım mühendisliğinde metriklerin kullanımı diğer disiplinlerdeki kullanıma oranla çok daha yenidir. Halen yazılım mühendisliği alanında gerek yazılımcı, gerek proje yöneticisi olsun, tüm paydaşlar ölçme kavramıyla mesafelidir. Ölçme kavramına aşina olanlar ise özellikle zaman ve işgücü kısıtları sebebiyle bu konunun üzerine düşmemekte ve ikinci plana atmaktadır. Bu durum özellikle büyük yazılım projelerinin başarısız olmasının altında yatan sebeplerden biridir [27].

## 2.2. Yazılım Mühendisliği'nde Ölçme Kavramı

Yazılım mühendisliği (Software Engineering) yazılım geliştirme eylemiyle ilgilenen bilim dalıdır. IEEE Bilgisayar Topluluğu'nun yaptığı tanıma göre Yazılım Mühendisliği [28];

- Bilgisayar yazılımının geliştirilmesi, işletilmesi ve sürdürülmesi için sistematik, disiplinli, ölçülebilir bir yaklaşımın uygulanmasıdır. Yani, mühendislik disiplininin yazılıma uygulanmasıdır.

Tanımda belirtildiği üzere yazılım mühendisliği, sistematik, disiplinli ve ölçülebilir bir yaklaşımdır. Fakat çoğunlukla göz ardı edilen ölçülebilir olma niteliği, oluşturduğu zaman ve bütçe problemleriyle yazılım mühendisliğinde başarısızlık hikayeleri ortaya çıkarabilmektedir. Bu durumun aşılabilmesi için yazılım mühendisliği kavramlarının belirlenecek standartlar kapsamında ölçülebilir hale getirilmesi ve ölçme işleminin getirdiği avantajlardan tüm paydaşların, projenin her aşamasında faydalanması ile mümkündür.

Yazılım mühendisliğinin, diğer mühendislik disiplinlerine göre problem yaratan niteliklerinden biri, karar alma eyleminde genel olarak nicel verilerin kullanım eksikliğidir [36]. Bunun yanı sıra uluslararası standart olarak tanınabilecek, genel anlamda kabul edilmiş ve ihtiyaçlara cevap verebilecek olgunlukta yazılım ölçme birimlerinin kısıtlı olması ölçme eyleminin benimsenmesini olumsuz yönde etkileyen sebeplerden biri olarak öne çıkmaktadır [36]. Ortak bir standart oluşturulamaması, bu konuda farklı kişi ve gruplarca yapılan ölçme çalışmalarının karşılaştırılabilir olmaktan uzak kalması sonucunu doğurmaktadır. Kıyaslanamayan deneysel sonuçların ise daha iyiye erişme noktasında, yazılım mühendisliği ölçme disiplinine gerekli katkıyı sağlamalarını imkânsız kılmaktadır [36]. Bu tür problemleri aşabilmek için standartlaştırma konusunda farklı bakış açılarına sahip araştırmacıların ortak bir amaca ulaşmak için koordineli çalışmaları önem arz etmektedir.

Yazılım boyutlarının ölçülmesi eyleminde söz konusu boyutun ölçülebilir birçok yönü vardır. En önemli yönleri 'uzunluk', 'işlevsellik', 'karmaşıklık' ve 'tekrar kullanılabilirlik' olarak sayılabilir [27]. Uzunluk, ortaya çıkan ürünün fiziksel büyüklüğünü belirtir. İşlevsellik ise ürünün kullanıcıya sağladığı işlevlerin büyüklüğüdür. Karmaşıklık farklı şekillerde yorumlanabilir olsa da genellikle

‘algoritma karmaşıklığı’, ‘yapısal karmaşıklık’, ‘problem karmaşıklığı’ olarak düşünülür. Yeniden kullanılabilirlik ise ortaya çıkan ürünün ne kadarının yeni geliştirildiği, ne kadarının önceki çalışmalardan yararlanılarak tamamlandığı ile ilgili bir büyüklüktür.

Yazılım mühendisliğinde üç temel hedefle yazılım büyüklük ölçümleri gereklidir:

- Yazılım projesini anlamak ve modellemek,
- Yazılım projelerinin yönetilmesinde yol göstermek,
- Yazılım süreç geliştirme ve iyileştirme çalışmalarına yön vermek

Yazılımın ölçülebilmesi, harcanılan zaman, emek, proje büyüklüğü ve kalite gibi faktörlerin belirlenmesine olanak sağlamaktadır. Organizasyonlar, bu verilere dayanarak olası diğer projeler için kestirim yapabilme imkânı kavuşabilmektedirler. Ayrıca, planlama, izleme ve kontrol gibi yazılım proje faaliyetlerinin yürütülmesi büyük ölçüde yazılım büyüklüğü bilgisine bağlıdır.

Yazılım mühendisliğinin tüm faaliyetleri göz önüne alındığında, zamandan tutun üretkenliğe kadar birçok ölçme metriği barındırdığı açıktır. Fakat konuyu belirlenmiş bağlam üzerinde tutabilmek için çalışma kapsamında ilgilenilen ölçme metriklerinden “büyüklük-boyut: işlevsellik” üzerinde durulacaktır.

## 2.3. İşlevsel Büyüklük Ölçme

Yazılım büyüklük ölçme eylemlerinde ilk olarak İşlev Puanı (Function Points) ve İşlev Puan Analizi (Function Points Analysis-FPA) 1979 yılında IBM’in satır sayısına alternatif olarak sunulmuştur [26]. İşlevselliği temel alarak bir yazılım ürününün büyüklüğünü ölçme fikri ilk Allan Albrecht tarafından önerilmiştir. 1983’de ise, Yönetim Bilgi Sistemlerinin büyüklüğünü ölçmek için Allan Albrecht ve John Gaffney tarafından FSM yöntemi geliştirilmiştir [29]. Çıkış noktası, kod satır sayısı metriğinin programlama dili ve platform bağımlı olması ve yazılım büyüklük ölçümlerini bu bağımlılıktan kurtarma gerekliliğidir. Özellikle geliştirme sürecinde erken dönemde ortaya çıkan iş ürünleri üzerinden işgücü ve süre tahmini yapabilen yöntemler, sıklıkla, işlevsellik ölçümü için tercih edilmektedir.

ISO 14143-1 [12] standardına göre, bir bilgi sisteminin işlevsel büyüklüğü işlevsel bilgi sistemi gereksinimlerinin kullanıcı perspektifiyle elde edilmesidir. Albrecht ve ark. yaptıkları öneriyle [2] işlevsel büyüklük, İşlev Puanı Analizi kullanılarak elde edilmektedir. İşlev Puanı, paydaşlar için mantıksal işlevsel terimleri kullanan

yazılım büyüklüğü metriğidir. Yazılım büyüklüğünün işlevsel gereksinimlerden hesaplanması, geliştirici yeteneğinden, tasarım tekniğinden ya da programlama dilindeki değişikliklerden bağımsız bir metrik sunmaktadır.

İşlevselliği temel alan ölçme yöntemi, büyüklük, planlama, izleme, maliyet kestirimi gibi birçok proje yönetim sürecini destekleyebilen bir metriği temel alması nedeniyle yazılım proje yönetimi süreçlerinde oldukça önemli bir konumdadır.

İşlev puan sunduğu olanaklar ile oldukça ilgi çekmiş ve araştırmacılar tarafından orijinal FPA yöntemi üzerinde yapılan çeşitli değişikliklerle, ölçüm yöntemi farklı birçok FSM yöntemi geliştirilmiştir. Aşağıda bazı işlevsel büyüklük kestirim yöntemleri verilmiştir;

- IFPUG İşlev Puanı Analizi (IFPUG Function Points Analysis–IFPUG FPA) [17],
- Mark II İşlev Puanı (Mark II Function Points–MK II FP) [19],
- NESMA İşlev Puanı (NESMA Function Points) [20],
- COSMIC Tam İşlev Puanı (COSMIC Full Function Points–COSMIC FFP) [16],

Tez kapsamında önerilen çeşitli işlevsel büyüklük hesaplama yöntemleri arasından son zamanlarda en çok öne çıkan ve popülaritesi gittikçe artan COSMIC İşlevsel büyüklük hesaplama yöntemi üzerinde çalışılacaktır.

## 2.4. COSMIC İşlevsel Büyüklük Ölçümü

COSMIC yöntemi yazılım geliştirme süreçlerinde yaygın şekilde kullanılan ve diğer yazılım büyüklük ölçüm yöntemlerinin platform bağımlılığı, geliştiriciye bağımlılık vb. zayıf yönlerini taşımayan güncel ve yenilenmiş bir işlevsel büyüklük ölçüm yöntemidir [1]. COSMIC yönteminin temeli ilk kez 1997 yılında Albrecth tarafından, FPA yönteminin bazı noktalarda farklı bakış açılarıyla geliştirilmiş bir şekli olarak ortaya çıkmış ve yapılan testlerde benzer uygulamalar için başarılı sonuçlar ürettiği görülmüştür. 1998'de FFP metodunun geliştiricileri, o dönem ikinci nesil işlevsel ölçüm metodunun prensiplerini tanımlayan COSMIC grubuyla birleşmiş ve yöntem, 1999 yılında COSMIC-FFP v2.0 olarak sunulmuştur. Sürüm 3.0 ile birlikte yöntem sadece "COSMIC" olarak anılmaya başlanmıştır [1].

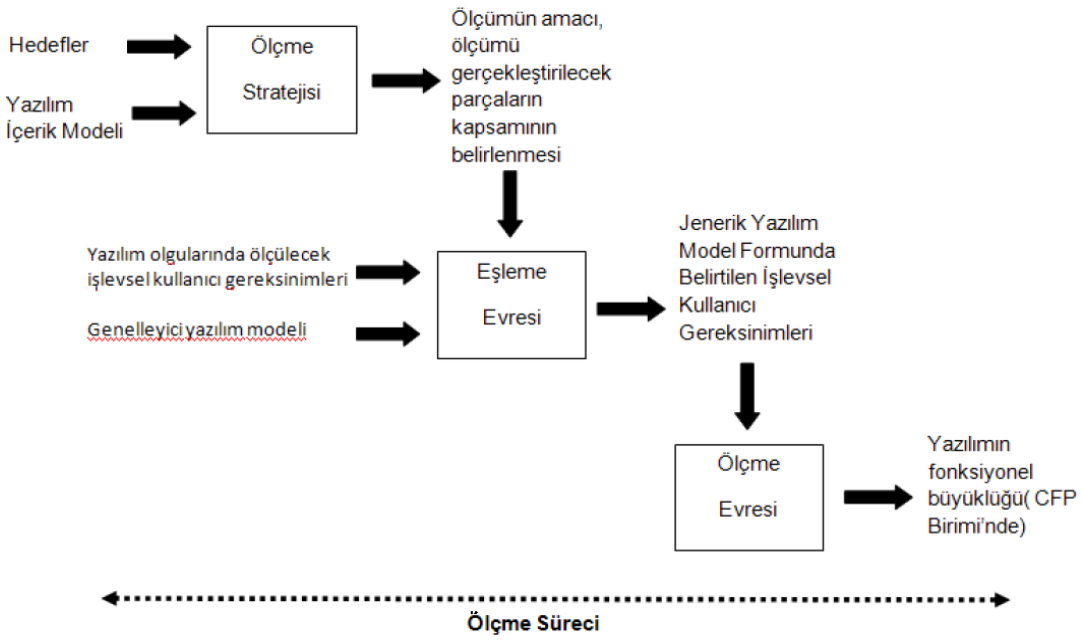
COSMIC metodunun kullanımı her ne kadar, sigortacılık, muhasebecilik gibi iş yönetimini destekleyici MIS yazılımları için daha uygun görünse de, önerilen MUB'lar ile kapsam çok daha genişletilerek her türlü uygulamada kullanılabilecek hale getirilmiştir. Yine de matematiksel karmaşıklık ağırlığının hissedildiği, benzetim yazılımları ve kendiliğinden öğrenen yazılımlar için büyüklüğün hesaplanmasında iyi bir performans sergileyememektedir.

COSMIC ölçme yönteminin hedefi yazılımların işlevselliğini ölçmektir. İşlevsellik, yazılımın kullanıcılar için gerçekleştirebildiği bilgi işleme eylemlerinin tümüdür. İşlevsel kullanıcı gereksinimleri ("Functional User Requirements" - FUR) ise işlevsel kullanıcılar için yazılımın gerçekleştirmesi gereken şeyleri tanımlar. Bu gereksinimler içerisinde teknik ya da kalite gereksinimleri yer almamaktadır.

Geliştirme safhalarının tümünde işlevsel kullanıcı gereksinimleri belirlenebilir. Geliştirme öncesinde, gereksinim tanımları ve analiz dokümanlarından belirlenebilen gereksinimler, yazılım geliştirilirken ve geliştirildikten sonra araçlar yardımıyla çalışma izlekleri ve veritabanı işlemleri üzerinden hesaplanabilmektedir.

COSMIC Ölçüm El Kitabı [1], ölçme için üç temel evre tanımlamaktadır (Şekil 2).

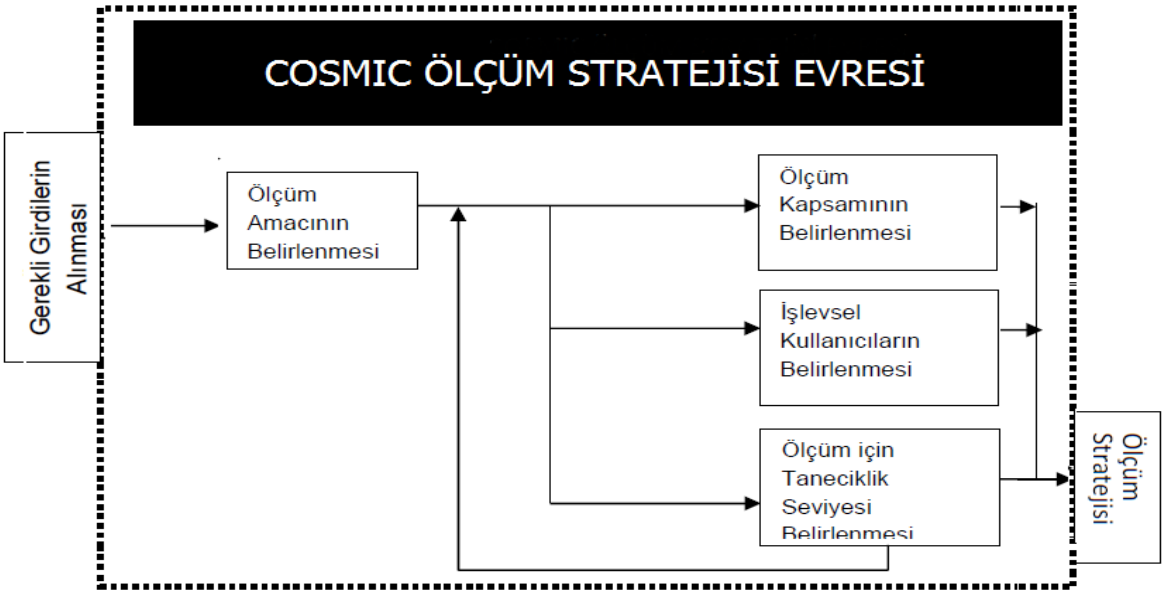
- Ölçme Stratejisi Evresi,
- Eşleme Evresi,
- Ölçme Evresi.




Şekil 2 COSMIC Yöntem Ölçme Evreleri [1]

### 2.4.1. Ölçme Stratejisi Evresi

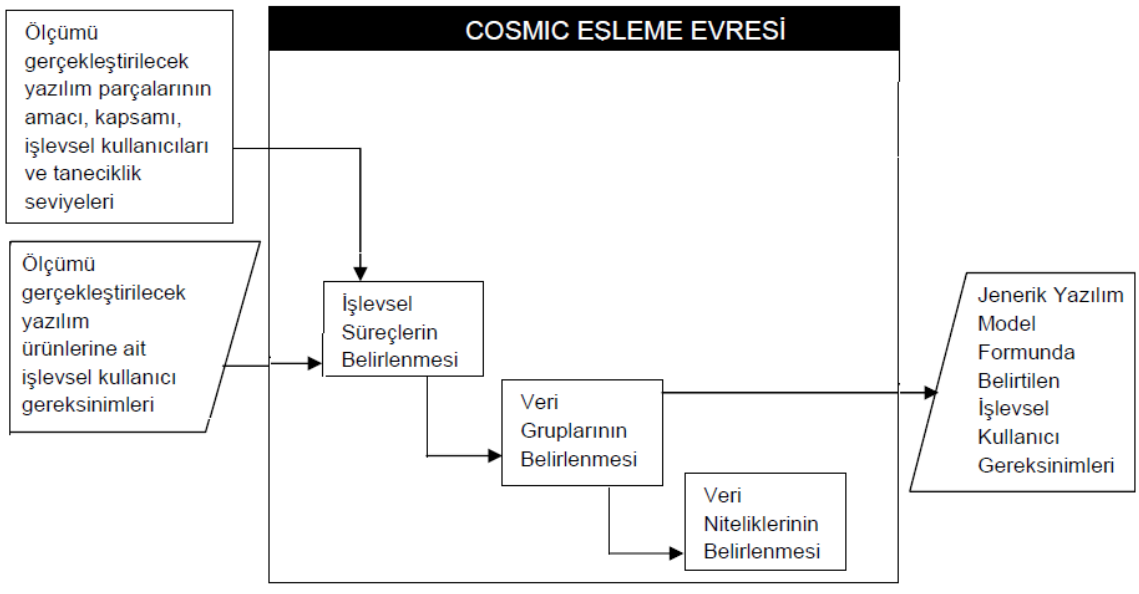
Strateji belirleme evresinde, yapılacak ölçme işleminin amacı belirlenir. Daha sonra kapsamı, hedef uygulamanın işlevsel kullanıcıları ve ölçmenin gerçekleştirileceği taneciklik düzeyi belirlenir (Şekil 3). Bu kavramlar ölçme sonrasında yapılacak karşılaştırmalar için de kullanılmaktadır.




Şekil 3 Ölçüm Stratejisi Belirlenme Süreci [1]

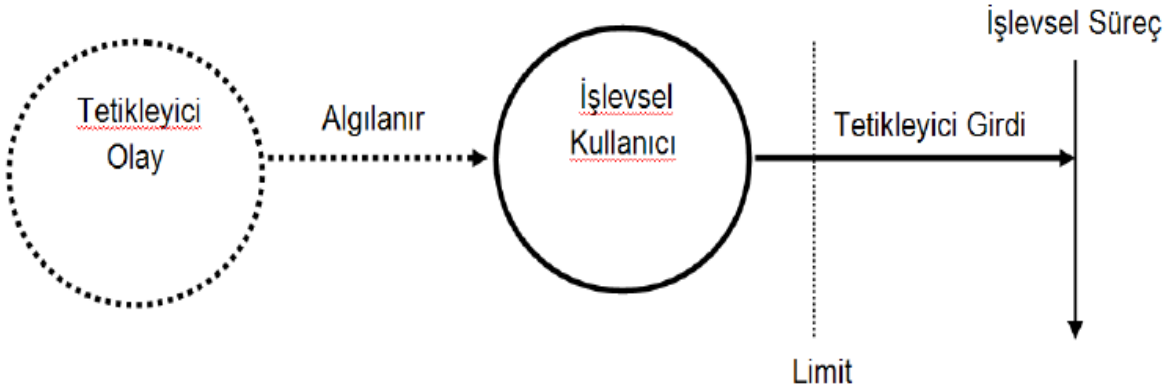
### 2.4.2. Eşleme Evresi

Eşleme evresinde, belirlenen işlevsel kullanıcıların tetikledikleri olay kümeleri yani işlevsel süreçler belirlenir. Sonrasında, bu süreçlerle ilgili veri grupları ve veri özellikleri belirlenir. Eşleme evresine ilişkin gösterim Şekil 4' te verilmiştir.




Şekil 4 Eşleme Evresi [1]

**İşlevsel Süreç Keşfi**: İşlevsel kullanıcı gereksinimleri üzerinden, yazılımın bir bölümü için işlevsel süreçlerin belirlenmesidir. Bir ya da daha fazla işlevsel sürecin başlamasını sağlayan eyleme "Tetikleyici Olay" adı verilmektedir. Tetikleyici olay, işlevsel süreç ve işlevsel kullanıcı arasındaki ilişki Şekil 5'te gösterilmektedir. COSMIC yöntem kapsamında bir işlevsel süreç bir "Giriş" ve onu takip eden "Çıkış" veya "Yazma" veri hareketleri olmak üzere en az iki veri hareketinden oluşmaktadır. İşlevsel süreç, tetiklenen olay sonucu dönen cevap ile birlikte bekleme statüsüne dönülmesiyle sonlanmaktadır.

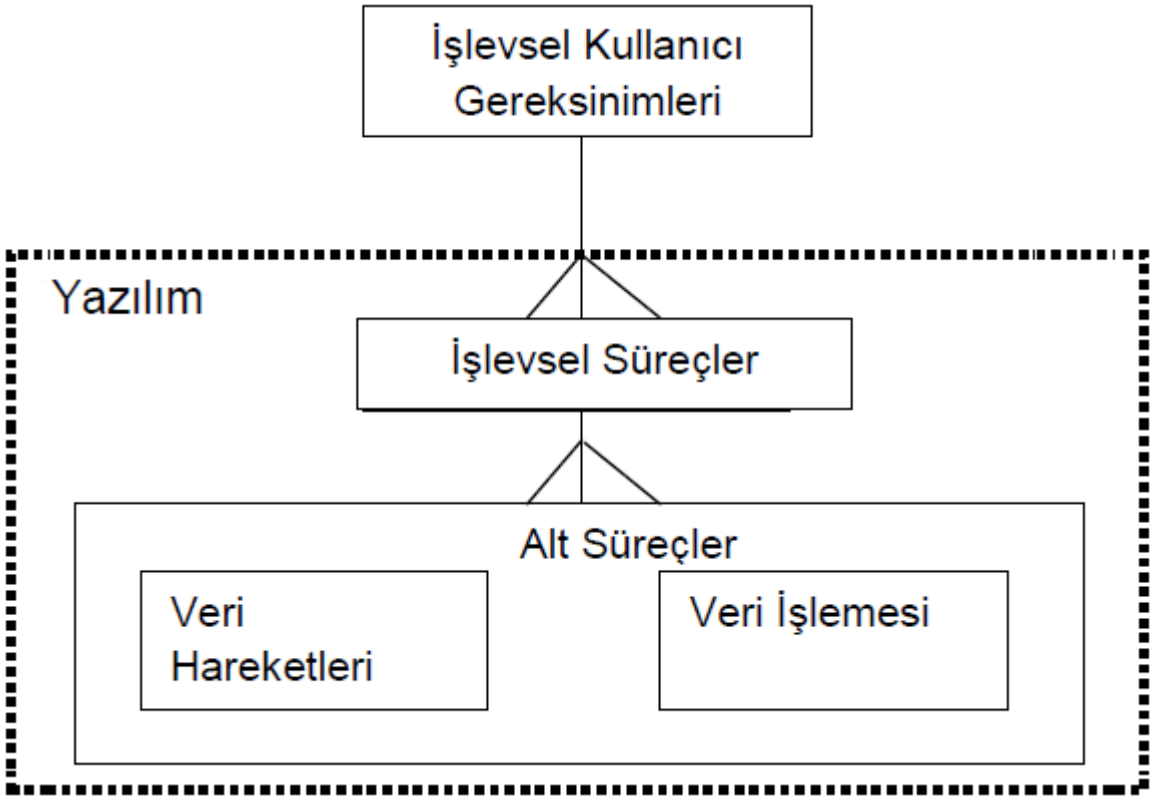



Şekil 5 Tetikleyici Olay [1]

**Veri grubu:** İş uygulamalarında özel amaçlı bir sorgu için bir araya gelen, "ilgi odağı" nitelikleri denilebilecek verilerdir. İlgi odağı işlevsel kullanıcı gereksinimleri bakış açısıyla tanımlanan her şeydir ve her bir veri grubu bir ilgi odağı ile doğrudan ilişkilidir. Veritabanında işlevsel süreçler aracılığı ile gerçekleşen okuma ve yazma veri hareketleri ile hareket eden veri grubunun sayısı, ilgilendirdiği ilgi odağı sayısı kadar veri hareketi oluşturur.

### 2.4.3. Ölçme Evresi

Bu evrede, tüm işlevsel süreçler için veri hareketleri belirlenir. Ölçme fonksiyonu uygulanır. Sonuçlar birleştirilir ve yazılımın işlevsel büyüklüğü elde edilir.




Şekil 6 İşlevsel süreçler ve alt süreçleri [1]

Bir alt süreç veri taşıma ya da veri işleme eyleminden oluşmaktadır. Şekil 6 'da işlevsel gereksinimler, işlevsel süreçler ve onların alt süreçleri arasındaki ilişkileri gösterilmektedir.

Veri manipülasyonları COSMIC kapsamında ayrıca ölçülmemektedir. Herhangi bir veri manipülasyonunun ilişkili olduğu veri hareketi üzerinden açıklandığı kabul edilmektedir.

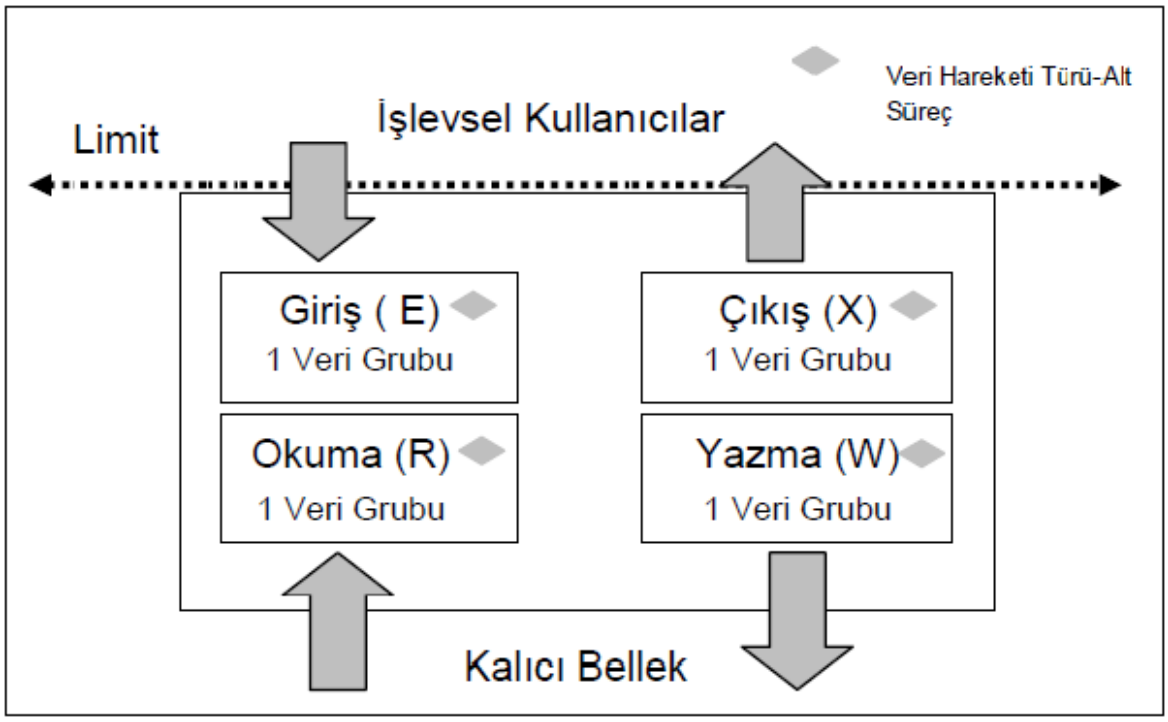
**Veri Hareketi;** Bir işlevsel sürece ait, yalnızca bir veri grubunun hareket etmesine sebep olan eylemdir. COSMIC metodu ölçme işleminde temel olarak dört veri hareketini ele almaktadır [1]. Söz konusu veri hareketi türleri aşağıda verilmiştir.

- Giriş ("Entry")
- Okuma ("Read")
- Yazma ("Write")
- Çıkış ("Exit")

Veri hareketleri, işlevsel kullanıcılar ve depolama aygıtı arasındaki ilişki Şekil 7'de gösterilmektedir.

**Giriş** veri hareketi, bir veri grubuna ait verilerin, kullanıcı tarafından sınırın diğer tarafına, yani işlevsel sürecin içerisine yaptığı harekettir. Birden fazla veri grubuna ilişkin veri taşınıyor ise, farklı veri grubu çeşidi kadar hareket tanımlanır. Giriş veri hareketinin ilişkili veri manipülasyonlarını içerdiği düşünülür.

**Çıkış** veri hareketi, bir veri grubuna ait verilerin, işlevsel süreç içerisinden sınırın diğer tarafına, kullanıcı tarafına yaptığı harekettir. Birden fazla veri grubuna ilişkin veri taşınıyor ise, farklı veri grubu çeşidi kadar hareket tanımlanır. Çıkış veri hareketinin ilişkili veri manipülasyonlarını içerdiği düşünülür.




Şekil 7 Veri Hareketleri ve Veri Hareket Yönleri [1]

**Okuma** veri hareketi, bir veri grubuna ait verilerin, kalıcı bellekten, işlevsel sürecin içerisine yaptığı harekettir. Birden fazla veri grubuna ilişkin veri taşınıyor ise, farklı veri grubu çeşidi kadar hareket tanımlanır. Okuma veri hareketinin ilişkili veri manipülasyonlarını içerdiği düşünülür.

**Yazma** veri hareketi, bir veri grubuna ait verilerin, işlevsel süreç içerisinden kalıcı belleğe yaptığı harekettir. Birden fazla veri grubuna ilişkin veri taşınıyor ise, farklı veri grubu çeşidi kadar hareket tanımlanır. Yazma veri hareketinin ilişkili veri manipülasyonlarını içerdiği düşünülür.

## 2.5. UML, Kullanım Durumu & Dizge(Sequence) Diyagramları

UML, Unified Modeling Language'in (Birleşik Modelleme Dili) kısaltmasıdır. UML, yazılım sistemlerinin tanımlanması ve tasarlanması amacıyla geliştirilmiş bir notasyon dilidir. Object Management Group (OMG) tarafından sunulan bu dil 2.2 versiyonunda olup sistem etkileşimlerini standartlaştırmak için geliştirilmiştir [21]. UML herhangi bir dilden bağımsız olmakla birlikte nesne-yönelimli bir yaklaşımı benimsediği için bünyesinde sınıf, nesne, arayüz, kalıtım gibi kavramlar ve ilgili notasyonları mevcuttur.

UML, bir yazılım geliştirme sürecinde yer alan farklı fazlara yönelik farklı açıları betimleyecek şekilde farklı diyagramlara sahiptir. 2.2 sürümünde UML yapısal ve davranışsal olarak tanımlanabilecek iki kategoride yer alan dokuz adet diyagrama sahiptir [22].

**Davranışsal Diyagramlar**

- Kullanım Durumu (Use-case) diyagramı: sistemin davranışsal yapısını ifade etmek için kullanılan diyagramlar
- Aktivite (Activity) diyagramı: kullanım durumlarının akışlarını modellemek için kullanılan diyagramlar
- Durum (State) diyagramı: karmaşık durumlara sahip olan nesnelerin davranışlarını betimlemek için kullanılan diyagramlar
- Dizge (Sequence) ve iletişim (communication) diyagramları: kullanım durumları tasarımlarını ve bu tasarımlar içinde nesnelerin dinamik davranışlarını betimlemek için kullanılan diyagramlar

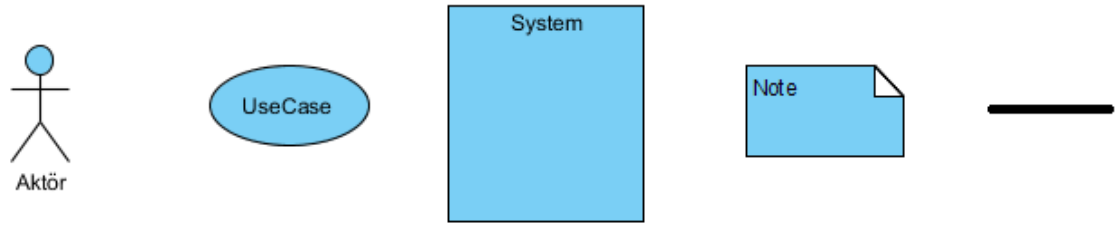
**Yapısal Diyagramlar**

- Sınıf (Class) diyagramı: Statik tip ilişkilerini ifade etmek için kullanılan diyagramlar (yapısal),
- Nesne (Object) diyagramları: Sınıf diyagramlarının bir "t" anındaki durumunu betimlemek için kullanılan diyagramlar
- Bileşen (Component) diyagramı: Modüller ve bileşenleri ile aralarındaki ilişkileri betimlemek için kullanılan diyagramlar
- Yerleştirme (Deployment) diyagramı: Yazılım sisteminin donanımsal unsurlarla birlikte test ya da gerçek ortam gibi, yazılımın kurulum yapılarını betimlemek için kullanılan diyagramlar

### Kullanım Durumu (Use Case) Diyagramı

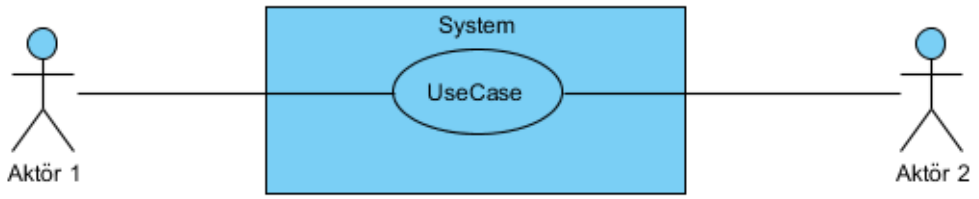
Kullanım durumu; bir kullanıcı ve bir sistem arasındaki etkileşimi anlatan senaryo topluluğudur. Ivar Jacobson'a [31] göre ise senaryo "Aktörle sistem arasında gerçekleştirilen, sonucunda aktör için fark edilir getirisi/faydası oluşan etkileşimli diyalogdur." UML kullanım durumu diyagramları sistemin işlevselliğini açıklamak amacıyla kullanılır. Sistemin birbirinden ayrı özelliklerinin detaylarını göstermekten daha çok, kullanım durumu diyagramları, sistemin mevcut işlevselliği paydaşlara aktarmak için kullanılmaktadır. [21]

Diyagram dört ana elemandan oluşmaktadır. **Aktörler**, **Sistem (**Proje kapsamını belirtir**)**, **kullanım durumları** ve bunlar arasındaki **ilişkiler**. Şekil 8'de kullanım durumu diyagramlarını oluşturan dört ana elemanın gösterim şekli bulunmaktadır.




Şekil 8 Kullanım durumu elemanları

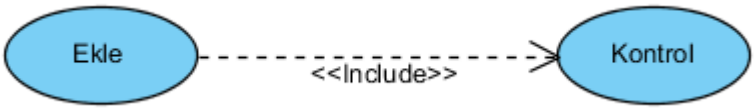
Kullanım durumu diyagramlarında senaryolar aktör tarafından tetiklenmektedir. Aşağıdaki senaryoda aktör senaryoyu başlatır, kullanım durumu sonucunda elde edilen değer diğer aktöre verilir.




Şekil 9 Kullanım durumu (Use Case) Diyagram Örneği

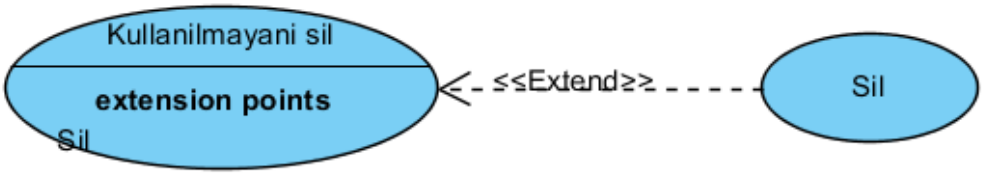
Kullanım durumları yeniden kullanılabilir birimlerdir. Yeniden kullanımı sağlayan yöntemler içerme ("inclusion") ve genişletme (extension) dır. Yeniden kullanım yöntemleri dışında kullanım durumları arasındaki ilişkiyi gösteren yöntemler ise genelleme (generalization) ve gruplama (grouping) dır. Bunlar kullanım durumları arasındaki ilişkiyi gösteren kavramlardır. Bu kavramları detaylandırmak gerekirse;

**İçerme (Include, uses):** İçerme ilişkisi kullanıcı senaryosu tekrarlanan adım içeriyor ise tekrar edilen bölümler ayrı kullanım durumu olarak ayrıştırılıp içerme ilişkisi ile kullanılmaktadır. *İçerme eylemi senaryonun içinden bir alt programa dallanıp geri dönmek gibi düşünülebilir.*




Şekil 10 İçerme (Include, uses)

**Genişletme (Extend):** Doğal akışa göre hazırlanan senaryo gruplarında koşula bağlı olarak dallanmalar meydana gelebilmektedir. Ana senaryodan ayrılma noktasından sonra yapılanlar genişletme ilişkisini tanımlamaktadır. Genelleme ilişkisine benzeyen genişletme eyleminde farklılık oluşturan durum, genişleme yapma imkanının genişleme noktalarının dışında mümkün olmayışıdır.




Şekil 11 Genişletme (Extend)

**Genelleme (Generalization):** Genelleme ilişkisi sınıflar arası türeme ilişkisi analojisiyle ifade edilebilir. Genelleme, özel bir senaryo grubunun genel bir senaryo grubundan türetilmesi eyleminde kullanılmaktadır.

**Gruplama (Grouping):** Kullanım durumlarını düzenli bir görünüme kavuşturmak için tanımlanmış olan bir tekniktir. Gruplama tekniği alt sistemler içeren daha büyük sistemleri daha kolay takip edebilmek için kullanılmaktadır.

**Kod Klişesi (Stereotype):** Diyagramlarda yer alan herhangi bir şeklin anlamını açıklamak için kullanılan özel sözcükler, (stereotype) <<...>> simgeleri arasına yazılmaktadır. Aktörler çizgi adam şeklinde gösterildiği gibi bir dikdörtgen ile de ifade edilebilmektedir. Bu durumda dikdörtgenin anlamını belirtmek için “stereotype” kullanılır.

**Dizge (Sequence) Diyagramı**

Dizge diyagramları sistemdeki nesneler ya da bileşenler arasındaki mesaj akışının olaylarını ve hareketlerini dizge şekilde modellemek için kullanılan diyagramlardır [21]. Bir dizge diyagramı; nesneler, mesajlar ve zaman çizelgesinden oluşmakta ve iki boyutta ifade edilmektedir.

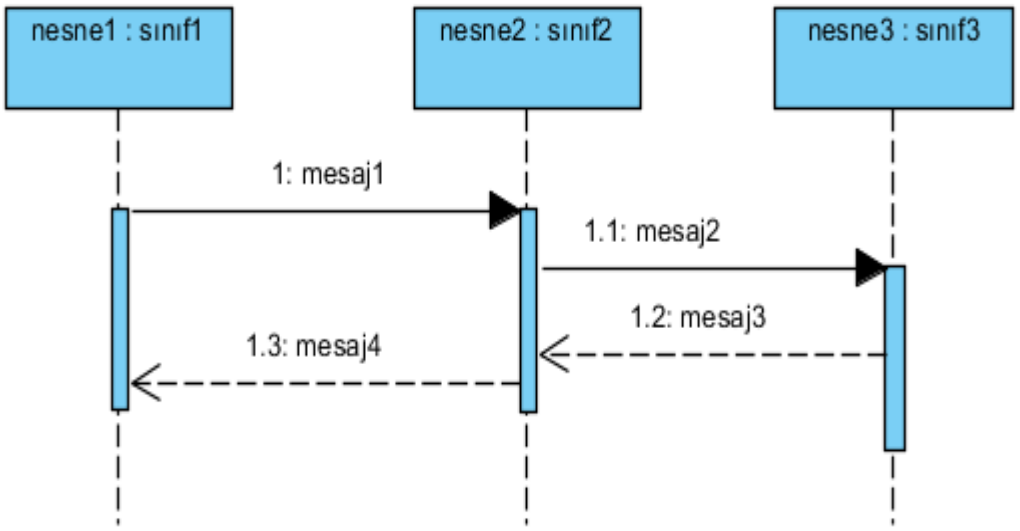
Dizge diyagramı boyutları:

- Dikey boyut: Mesajların/olayların sırasını oluşma zamanı sıralarına göre göstermektedir.
- Yatay boyut: Mesajın gönderildiği nesne örneklerini (object instances) göstermektedir.

Dizge diyagramları, tasarımı yapan kişi ya da ekibin bakış açısına, sistem gereksinimlerine ve modülün tüm çalışanlar tarafından bilinirliği başta olmak üzere birçok etmene bağlı olarak değişebilmektedir. Kolayca anlaşılabilecek sistemler için basit bir dizge diyagramı yeterli olabilecekken yeni geliştirilmekte olan bir sistem için her bir ayrıntıyı gösteren diyagramlar gerekebilmektedir. Bu ayrım gözetildiğine dizge diyagramları iki türe ayrılabilir.

- Genellikle basit bir şemanın çizildiği ve en iyi olasılıkları ele alan "*best case*" senaryo üzerinden yürüyen **örnek (instance) dizge** diyagramı.
- Yazılım modelinin her yönüyle ele alındığı ve daha kompleks bir model olan **genel (generic) dizge** diyagramı.

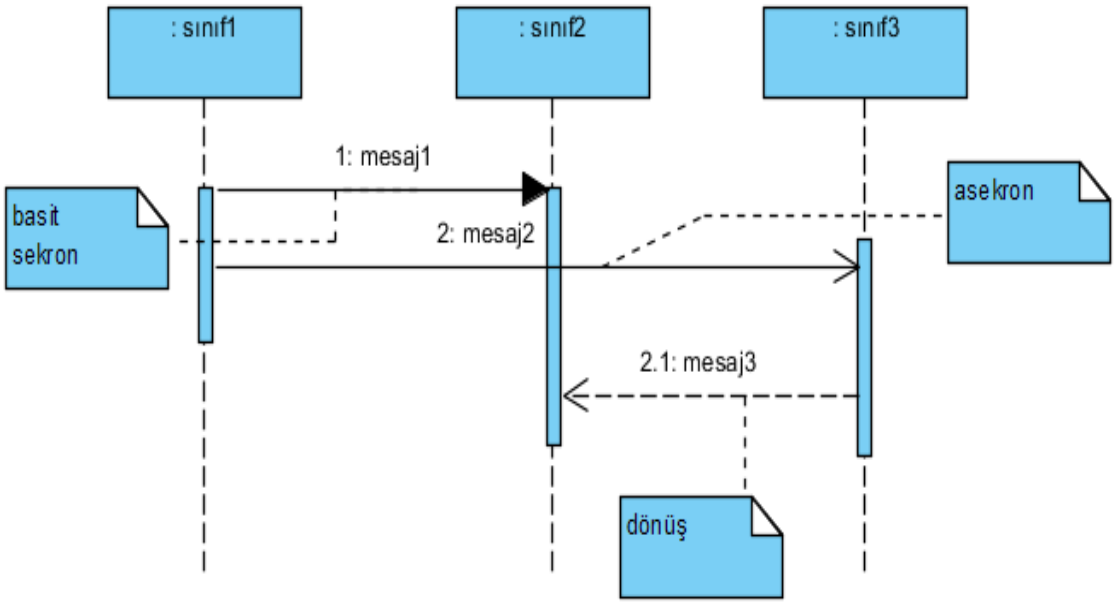
Dizge diyagramlarında işlem akışı soldan sağa doğru olmak üzere ifade edilmektedir. Okunurken soldan sağa ve üstten alta doğru okunmaktadır. Dizge diyagramları kullanım durumlarıyla (use case) doğrudan ilişkili olup başlangıç noktasını ve aktörü belirlemektedir. Her bir nesnenin altından çıkan kesikli çizgi zaman çizgisini (timeline) göstermektedir. Kesikli çizgi zaman akışını gösterirken üzerinde bulunan ince uzun dikdörtgenler o nesnenin zaman içerisinde meydana getirdiği aktivitelerdir. Teorik olarak dikdörtgenin uzunluğu aktivitenin uzunluğu ile doğru orantılıdır.




Şekil 12 Dizge diyagramı genel gösterimi

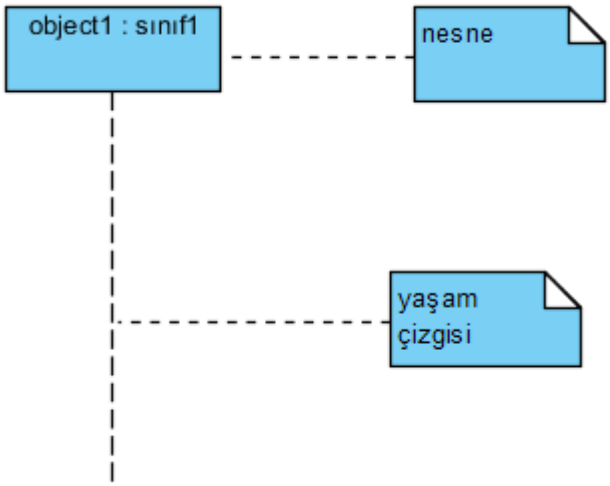
**Dizge Diyagramının Modellemesinde Kullanılan Elemanlar**

Dizge diyagramında yer alan elemanlar Şekil 13' de örneklenmektedir.




Şekil 13 Dizge Diyagramlarda Kullanılan Elemanların Gösterimi

**Nesne (Object) ve Yaşam Çizgisi (Lifeline) :** Dizge diyagramlarında yaşam çizgisi ve nesne gösterimi aşağıdaki gibidir.




Şekil 14 Nesne ve yaşam çizgisi basit gösterimi

**Mesaj**: Dizge diyagramında farklı nesnelerin birbirleri ile etkileşimi mesajlar ile gerçekleştirilmektedir. Nesnelerin birbirleri ile haberleşmesi, birbirlerine Sundukları servisleri temsil eden ve servisi veren nesneden alan nesneye doğru aktarılan mesajlar bu oklarla gösterilir. Mesaj gösterimi mesajların tipine göre değişmektedir. Dizge diyagramlarında mesajlar, basit (simple) mesajlar, nesne oluşurken ya da bellekten silinirken kullanılacak özel mesajlar ve mesaj yanıtları olarak değişik şekillerde gösterilebilmektedir. Bir nesne başka bir nesnenin dışında kendine de mesaj gönderebilmektedir (öz yineleme - “recursion”). Bunun yanında, ‘Aktör’ gibi nesne olmayan UML bileşeni de nesnelere mesaj gönderebilmektedir.

**Mesaj Tipleri ve Gösterimleri**

Dizge diyagramlarında dört çeşit mesaj tipi yer almaktadır [21]. Bunlar;

- **Basit (Simple) Mesaj Tipi:** Basit mesajlar nesneler arasındaki akış kontrolünün iletimini göstermek için kullanılır. Nesnelerin metotlarını doğrudan çağıramazlar. Sık kullanılan mesaj tipi değildir.
- **Senkron (Syncronous) / Çağrı yapan (Call) Mesaj Tipi:** Nesne, mesajı alıcı nesneye gönderir ve onun işlemini bitirmesini bekler. Bu durumda senkron mesaj tipi kullanılır. Nesne tabanlı programlamada çağırılan birçok

metod senkron çalıştığından en çok kullanılan mesaj tipi olarak gösterilebilir.

- **Asenkron Mesaj Tipi:** Senkron mesajların tersine, asenkron mesajlar nesneye mesaj gönderdikten sonra cevap beklemeden işleme devam etmesinin ifade edilebilmesini sağlayan mesaj tipidir.
- **Dönüş (Return) Mesaj Tipi:** Senkron mesajlarda alıcı nesnenin işleminin bitimini, gönderen nesneye geri bildirmesinde kullanılmaktadır.


Şekil 15 Mesaj gösterimleri

Dizge ve kullanım durumu diyagramlarının COSMIC işlevsel büyüklüğün hesaplanmasında kullanılması fikrinin oluşmasına sebep; COSMIC veri hareketleri notasyonunun, UML dizge diyagramında yer alan varlık-sınıf, nesneler ve veri depoları arasındaki iletilerle oldukça benzerlik göstermesidir [21]. Bu benzerlik sistemde yer alan işlevsel süreçlerin akışının takip edilebilmesini sağlamakta dolayısıyla ölçme uzmanına büyük kolaylık sağlayabilmektedir. Yapılan çalışma bu kolaylıktan faydalanarak COSMIC işlevsel büyüklüğün hesaplanması işleminin otomatize edilmesini hedeflemektedir.

Bu çalışmada hedeflenen ortam, Java platformudur. Kullanıcı senaryolarına karşılık gelen dizge diyagramlarının elde edilmesinde bağlam yönelimli programlama teknolojisi kullanılmıştır. Kullanılan teknoloji alt bölümlerde açıklanmaktadır.

## 2.6. Bağlam Yönelimli Programlama ve AspectJ

### 2.6.1. Bağlam (Aspect) Yönelimli Programlama

Günümüz tipik kurumsal ve web uygulamalarında güvenlik, kesintisiz işlem davranışı, loglama ve benzeri kavramlar/servisler sıkça kullanılmaktadır. Bu servisleri sunan alt sistemler modüler bir şekilde tanımlanabilmektedir. Fakat bu servislerin kullanımı, servislere ihtiyaç duyan işlemlerin onları çağırabilmesi için birbirinin kopyası birçok kod parçasının çeşitli yerlere eklenmesine sebebiyet

vermektedir. Bu tür olumsuz durumları ortadan kaldırabilecek bir çözüme ihtiyaç duyulduğu açıktır. Bağlam yönelimli programlama bu ihtiyaç üzerine geliştirilmiş bir yöntemdir [32].

Bağlam yönelimli programlama (Aspect Oriented Programming) yazılım mühendisliğinde kullanılan bir programlama yaklaşımıdır. Nesne yönelimli programlamanın ele alamadığı ilgiler (concerns) üzerine kurulmuştur. AOP olarak kısaltılan bu yaklaşım Türkçe literatüründe bağlam / cephe / kesit / görünüm yönelimli programlama şeklinde çeşitli kelimelerle ifade edilebilmektedir.

Nesneye dayalı programlama, kavramların ayrıştırılmasında oldukça başarılıdır. Buna rağmen tek bir yazılım modülünde gerçekleştirilemeyecek, tek bir program modülüne uymayacak veya birden fazla program modülüyle hemen hemen aynı derecede ilişkili kavramları sarmalamada (encapsulation) etkisiz kalmakta ve problem oluşturmaktadır [32]. Sarmalamada problem oluşturan kavramlar, servislerin kalitesi, güvenlik gibi üst düzeyli olabileceği gibi, yastıklama(buffering), bellekleme(caching) gibi alt düzeyli de olabilmektedir. Aynı zamanda bir şirkette geliştirilen çeşitli iş mantıkları gibi fonksiyonel özellikte veya senkronizasyon gibi fonksiyonel olmayan özellikte olabilmektedir. Bu tür özellikler bağlam yönelimli programlamada ilgi olarak adlandırılır. İlgilerin gerektirdiği çözümler benzer niteliktedir. AOP temelini oluşturan ilgiler aynı ya da benzer gereksinimler olduğunda çözümün tek bir noktadan sağlanmasını hedefleyen kesişen ilgiler (crosscutting concerns) kavramını doğurmaktadır.

**Kesişen İlgiler**

Kesişen ilgiler nedir sorusuna en iyi açıklama bir örnek üzerinde sağlanabilir. Örneğin bir yazılımda belirli işlemlerde bir günce (log) oluşturulmasına ihtiyaç varsa, günce oluşturan prosedürler her bir metod içine yerleştirilmesi gerekmektedir. Bu durum işlemde ve kodda tekrarlanma sıkıntısına yol açmasının yanı sıra, günce prosedürlerinin değiştirilmesine ihtiyaç olması durumunda, metod sayısı kadar değişikliğe ihtiyaç oluşturarak, hataya açık bir durum oluşmasına sebebiyet vermektedir. Bu problemlerin yanı sıra, uygulamanın özünü oluşturan kodun, günce prosedürlerini (muhtemel benzer işlevde farklı kesişen kavramlar) oluşturan kodla karışması; kod temizliği, açıklığı ve anlaşılırlığını kötü yönde etkileyerek aynı metodun farklı bir bağlamda yeniden kullanılabilirliğinin önüne

geçmesine sebep olabilmektedir. Bahsedilen problemleri çözebilmek adına, bağlam yönelimli programlama yöntemi önerilmektedir [32]. Loglama veya kimlik doğrulama gibi ilgiler, birçok modülde kesişmektedirler. İşte bu ve benzeri durumlarda kavramlarla ilişkili metotlar birbirleriyle kesişiyorsa, bu kavramlar çapraz-kesilen (crosscut) kavramlar olarak nitelendirilmektedir. Bu kesişim noktaları bir yerde, bir kez tanımlandıktan sonra kullanılması daha anlaşılabilir, düzenlenebilir ve kullanılabilir olmasının yanı sıra diğer modüllere dokunmadan bağlamlar (aspects) yardımıyla sorunsuz şekilde çalışma imkânı sağlanmaktadır.

**AOP Terimleri**

Bağlam; ilgi, birleşme noktası ve kesim noktasının birleşiminden oluşmaktadır.

Tez kapsamında yapılan çalışmada sıkça geçecek olan AOP terimlerine bakacak olursak;

- Birleşme Noktası (Join point) : Program akışında belirgin şekilde tanımlanmış bir noktadır.
- Kesim Noktası (Pointcut) : Bir grup birleşme noktası anlamında kullanılır. Hangi metotların çalışma zamanında yakalanacağını belirtirler. Çalışma alanının tanımlandığı, anlatım (expression) yapısıdır.
- Tavsiye Yordam (Advice) : İlgi içerisinde çalıştırılacak asıl program koduna denir. Ayrıca kesim noktasıyla belirtilen birleşme noktasına ulaşıldığında bağlamın önce mi, arasında mı veya sonrasında mı çağrılacağını tanımlamaktadır.
- Dokuma (Weaving) : Tavsiye yordamı belirtilen birleşme noktalarına uygulama işlemidir.
- Bağlam (Aspect) : İlgiyi oluşturan işlevselliğin gerçekleştirimini sarmalayan sınıf niteliğinde bir kavram olup birçok sınıfı kesen bir ilginin modüler hale getirilmesini sağlayan yapıdır.

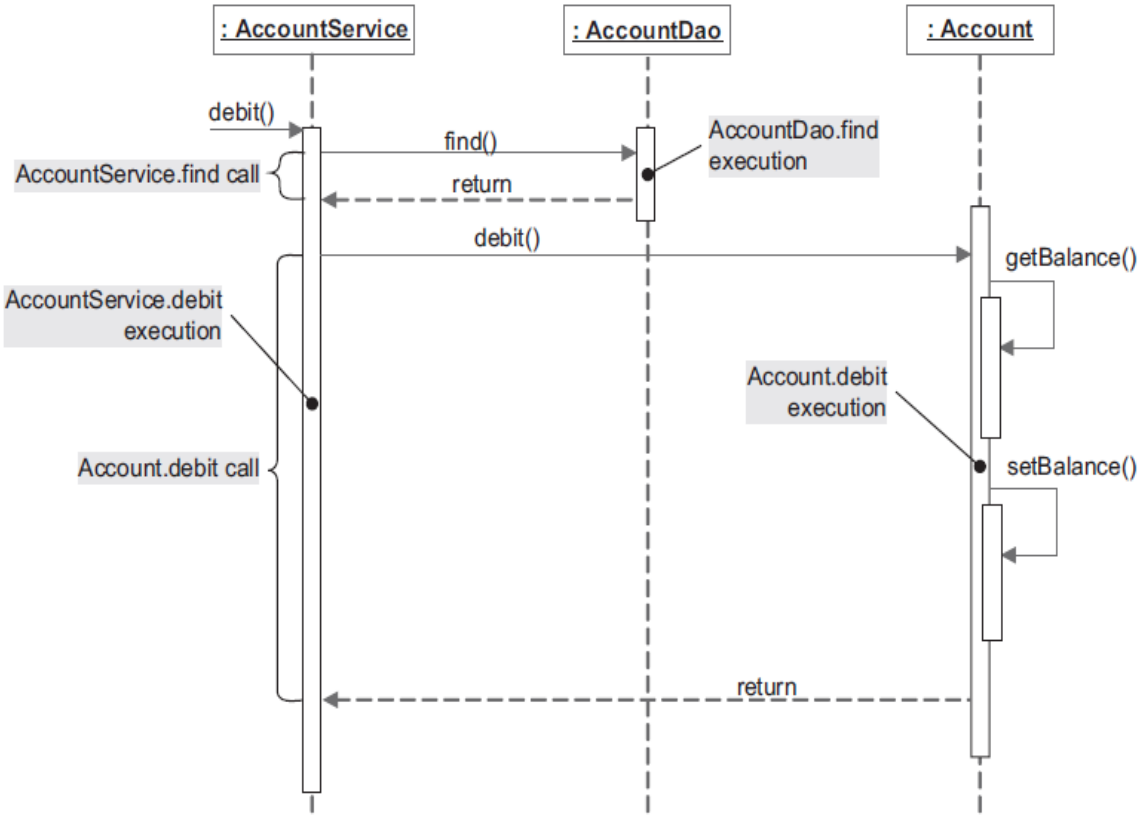
### 2.6.2. AspectJ

Bağlam yönelimli programlama konseptinin Java platformunda gerçekleştirimi AspectJ'dir. AspectJ, Java platformu için yeni bir modülerlik formu sağlayan uzantıdır. Daha önce belirtilmiş olan, sistemin modülerliğini bozabilen çapraz kesilen kavramların sarmalanmasını sağlamak için tasarlanmıştır.

**Birleşme Noktası (Joinpoint)**

AspectJ, birleşme noktası kavramını aşağıdaki durumların gerçekleşmesi durumunda kullanır.

- Metodu ya da yapılandırıcısı (constructor) çağrıldığında (call).
- Metodu ya da yapılandırıcısı (constructor) çalıştırıldığında (execute).
- Herhangi bir alana (field) erişilmek istendiğinde ya da güncelleme yapılmak istendiğinde,
- Sınıfı ya da nesne tanımlanma esnasında,
- Aykırı Durumları (Exception) ele alınırken,




Şekil 16 Program çalışma akışında birleşme noktaları [32]

Birleşme noktaları tanımlanırken en önemli şey metod imzalarıdır. Java programlama dilinde, sınıflar, arayüzler, metodlar ve alanların hepsi bir imzaya sahiptir. Fikir vermesi açısından bu imza söz dizimlerine örnek verecek olursak;

Tablo I Birleşme Noktası İmza Örüntüleri ve Eşleşen Tipler – Örnek 1

| İmza Örüntüsü | Eşleşen Tipler |
|---|---|
| `Account` | Account tipi adı |
| `*Account` | Account ile biten tüm tipler. Örneğin; SaveAccount |
| `java.*.Date` | Java paketi altında yer alan herhangi bir alt paket grubunda olabilecek tüm Date tipleri. Örneğin; java.util.Date ve java.sql.Date |
| `java..*` | Java paketi altında yer alan tüm tipler ya da tüm alt paketleri altındaki tipler. Örneğin; java.awt yada java.util ve onların alt paketleri; java.awt.event yada java.util.logging gibi |
| `javax..*Model+` | Sonu Model ile biten ve doğrudan ya da dolaylı olarak javax paketi altında yer alan tüm tipler ve alt tipleri. Örneğin; TableModel, TreeModel ve alt tipleri. |

Birleşme noktaları teker teker tanımlanabileceği gibi birleştirilerek de tanımlanabilir;

Tablo II Birleşme Noktası İmza Örüntüleri ve Eşleşen Tipler – Örnek 2

| İmza Örüntüsü | Eşleşen Tipler |
|---|---|
| `!Vector` | Vector haricindeki tüm tipler |
| `Vector \|\| Hashtable` | Vector yada Hastable tipi. |
| `javax..*Model \|\| javax.swing.text.Document` | Model ile biten ve doğrudan yada dolaylı javax paketin altında yer alan tüm tipler yada javax.swing.text.Document tipi |
| `java.util.RandomAccesss+ && java.util.List+` | Belirtilen arayüzleri gerçekleştiren tüm tipler. Örneğin bu imza söz dizimi, java.util.ArrayList tipine denk gelir. |

AspectJ, belirtildiği üzere birçok birleşme noktası tanımı içermektedir. Yapılan çalışma ve tez kapsamda yordam çağrımı ve çalıştırımı için birleşme noktası (method call & execution join point) yeterli olduğundan sadece bu birleşme

noktaları üzerinde durulacaktır. Bu birleşme noktalarının imza sözdizimiyle nasıl ifade edilebileceğine örnek verilecek olursa;

Tablo III Birleşme Noktası İmza Örüntüleri ve Eşleşen Tipler – Örnek 3

| İmza Örüntüsü | Eşleşen Tipler |
|---|---|
| `public void Collection.clear()` | Public erişime sahip, void döndüren ve parametresi olmayan, Collection sınıfındaki clear() metodu. |
| `public void Account.debit(float) throws InsufficientBalanceException` | InsufficientBalanceException istisnasını fırlatan, public erişime sahip, float parametresini alan ve Account sınıfının debit() metodu |
| `public void Account.set*(*)` | Account sınıfında yer alan, set kelimesiyle ile başlayan ve herhangi bir tipte tek parametreye sahip tüm public metodlar. |
| `public void Account.*()` | Void döndüren ve hiçbir parametre almayan, Account sınıfındaki tüm public yöntemler |
| `public * Account.*()` | Dönüş tipinin önemsenmediği, herhangi bir parametre almayan, Account sınıfındaki tüm public metodlar |
| `public * Account.*(..)` | Herhangi bir sayıda parametreye sahip, geri dönüş tipinin önemsenmediği, Account sınıfındaki tüm public metodlar |
| `* Account.*(..)` | Account sınıfındaki tüm metodlar; public, private farketmez |
| `!public * Account.*(..)` | Account sınıfındaki public olmayan tüm methodlar. Private olabilir, protected olabilir. |

*Örnek 1:* Geri dönüş tipi önemsiz, "set" kelimesiyle başlayan ve *integer* tipinde bir parametresi olan tüm yöntemleri çalıştırıldığında yakalayabilmek için alttaki birleşme noktasını tanımlarız.

```
execution(* set*(int))
```

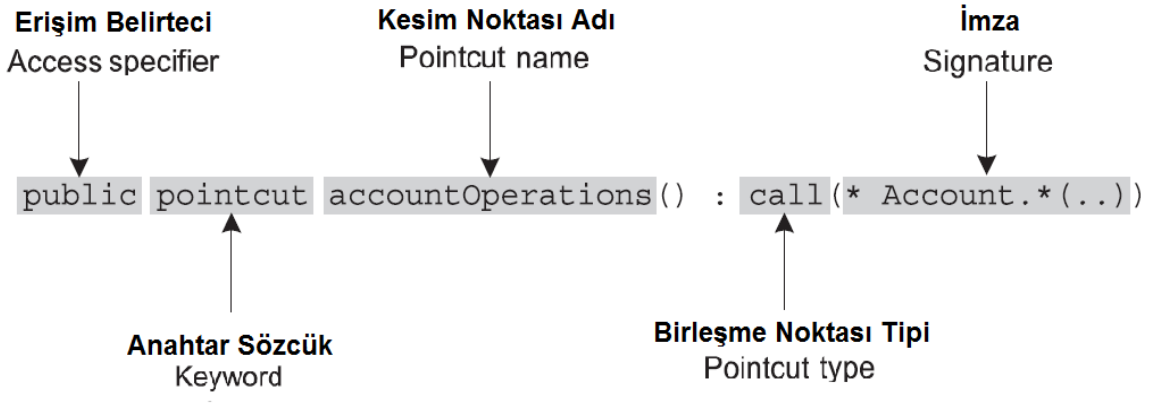
*Örnek 2:* Geri dönüş tipi karakter katarı olan, "*get*" kelimesiyle başlayan tüm yöntemleri parametre sayısı ve tipini önemsemeksizin çağrı yapıldığında yakala diyebilmek için alttaki birleşme noktasını tanımlarız.

```
call(String get*(*)
```

### Kesim Noktası (Pointcut)

Program akışı içerisinde birleşme noktalarını belirleyen, yakalayan ve tavsiye yordamların ne zaman çalışacağını kontrol etmek için kullanılan ifadedir [32]. Aynı zamanda birden fazla birleşim noktasının bir arada kullanılmasına da imkan sağlar. Birleşim noktalarının tanımlarıyla bir arada kullanılabilir.

Kesim noktalarının söz dizimi Şekil 17'deki gibidir.




Şekil 17 Kesim noktası (pointcut) söz dizimi yapısı

*Örnek 1:* "public" ve bir *integer* parametresi olan tüm metod çağrıları için tanımlanmış birleşme noktasını sarmalayan ve tavsiye yordamların kullanabileceği forma sokan kesim noktası tanımı;

```
aspect C {
      pointcut publicCall(): call(public * *(int));
}
```

Kesim noktalarının taşıdığı birçok özellik ve sahip olduğu tip mevcuttur. Fakat çalışma kapsamında ihtiyaç duyulan formu örnekte belirtilen en basit kullanım şeklidir.

**Tavsiye Yordam (Advice)**

Belirli bir birleşme noktasında tanımlı bağlam tarafından gerçekleştirilen eyleme tavsiye yordam denir. Birleşme noktasında nasıl etkileşime geçileceğini belirleyen üç farklı tipte tavsiye yordam mevcuttur. Bunlar;

1. **İçinde (Around):** Birleşme noktasının ne öncesinde ne sonrasında tamamıyla üzerinde tanımlı olduğu birleşme noktası içinde çalıştırılan tavsiye yordamdır. Bu tavsiye yordam diğerlerinden farklı olarak, eğer bir geri dönüş tipi tanımlanmışsa bir değer döndürebilir. Metot içerisinde değişiklik yapabilir. Değerleri manipüle edebilir.

   *Örnek 1:* C sınıfı içerisinde yer alan foo() metoduna bir çağrı yapıldığında üç değerini döndür ifadesi aşağıdaki gibi ifade edilebilmektedir.

```
aspect A {
    int around(): call(int C.foo()) {
       return 3;
    }
}
```

   İşlemler sonrasında orijinal yöntemi çağırabilme imkanı vardır ve bunun için özel bir söz dizimi kullanılır.

```
proceed( ... )
```

   Bu söz dizimi tanımlı olduğu birleşme noktasının tipinde ve sayısında parametre alır. Örneğin; alttaki tavsiye yordam “int C.foo(Object,int)” yöntemi çağrıldığında devreye girer ve yöntemin ikinci parametresini ikiyle çarparak orijinal yöntemi çağırır. Geri dönüş değerini alır ikiye böler ve sonucu döndürür.

```
aspect A {
  int around(int i): call(int C.foo(Object, int)) &&
args(i){
    int newi = proceed(i*2)
    return newi/2;
 }
}
```

2. **Önce (before)** : Birleşme noktasının hemen öncesinde çalıştırılan tavsiye yordamdır. Gerçek yordam işletime alınmadan hemen önce (parametre geçişinin hemen ardından) işletilir. Birleşme noktasında gerçekleşen

dallanma sonrasında devam edecek çalışma akışını durdurabilme özelliği yoktur (eğer bir istisna fırlatmaz ise).

*Örnek 1*: C adlı sınıfta yer alan foo(Object, int) metoduna bir çağrı yapıldığında, bu metodu çağırmadan önce komut satırına "foo objesi" metnini yazdır isteği alttaki gibi ifade edilebilmektedir.

```
aspect A {
      before(): call(int C.foo(Object, int)) {
         System.out.println("foo objesi");
      }
}
```

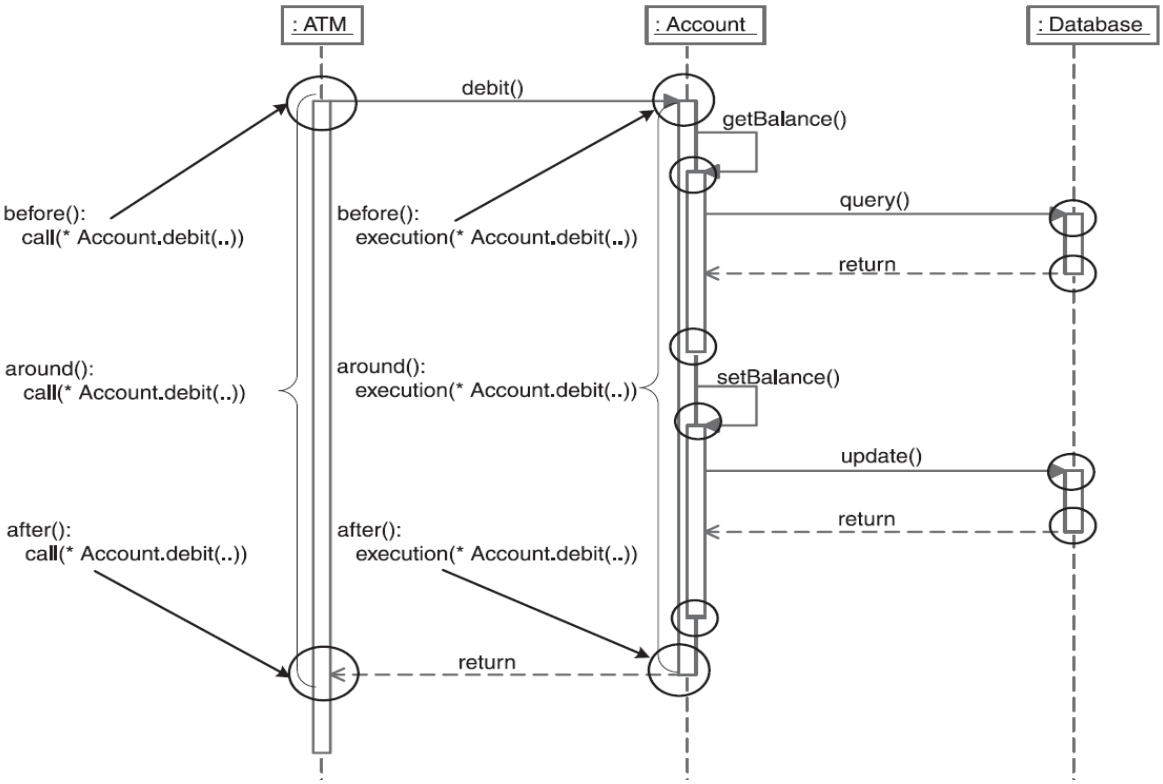
*Örnek 2*: "Node sınıfında yer alan, *set* ile başlayan ve birinci parametresi *int* olan tüm metodlara bir çağrı yapıldığında metod çağrılmadan önce metodun ilk parametresini yakala; bu parametreyi değer adlı değişkene ata ve bu değişkenin sıfır olup olmadığını kontrol et; eğer sıfırsa bir istisna fırlat" ifadesi alttaki gibi ifade edilebilmektedir.

```
aspect Ornek{
   before(int deger) : call(* Node.set*(int, .. )) &&
args(deger){
     if (deger == 0){
        throw      new      IllegalArgumentException("0
     gecersiz bir degerdir");
     }
   }
}
```

3. **Sonra (after):** Birleşme noktasının hemen bitiminde çalıştırılacak tavsiye yordamdır. Bitiş işleminin normal ya da istisnai durum sebebiyle olması tavsiye yordamın çalışmasını etkilemez. Üç farklı tipi mevcuttur: "after returning", "after throwing", "after".

```
aspect A {
   pointcut publicCall(): call(public Object *(..));
   after() returning (Object o): publicCall() {
       System.out.println("Returned normally with " + o);
   }
   after() throwing (Exception e): publicCall() {
       System.out.println("Threw an exception: " + e);
   }
   after(): publicCall(){
       System.out.println("Returned or threw an Exception");
   }
}
```

Tavsiye yordamların daha iyi anlaşılabilmesi için hazırlanan grafik Şekil 18 ‘de yer almaktadır.




Şekil 18 Dizge diyagramıyla tavsiye yordamların görselleştirilmesi [32]

Grafikte gösterilen ve her biri çemberle ifade edilen kesim noktaları önce ya da sonra tavsiye yordamının gösterildiği yerlerdir. Çemberlerin arasında kalan kesim ise tavsiye yordamıdır.

Tez çalışmasında özellikle ihtiyaç duyulan tavsiye yordamlar, önce ve sonra olarak nitelendirilen yordamlardır. Tüm çalışmada sadece bu iki tavsiye yordam kullanıldığı için diğer kısımlara değinilmemiştir.

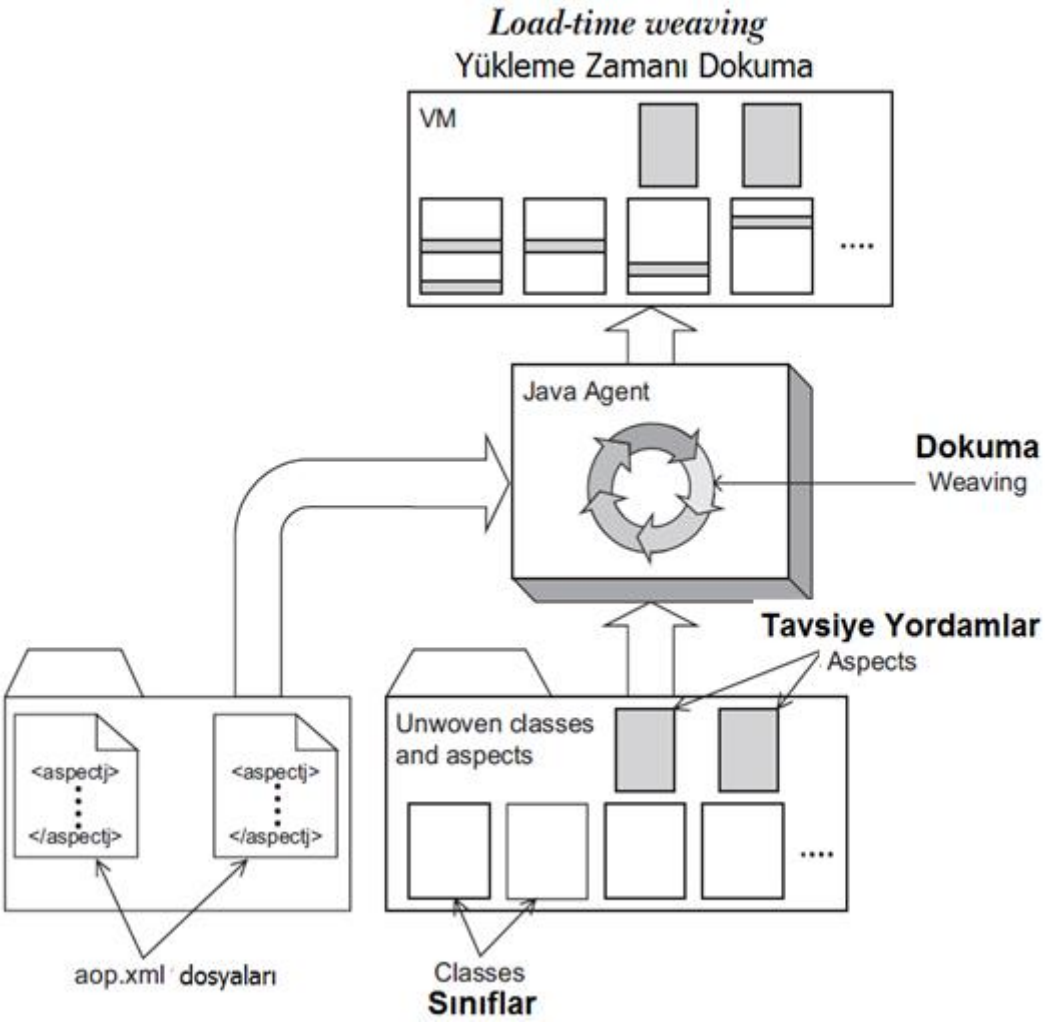
**Dokuma (Weaving)**

Tavsiye yordam ile ilişkili nesneleri oluşturarak bağlamları (aspects) uygulama ile ilişkilendirme eylemine dokuma denir. Dokuma; temel AOP yönteminin gerçekleştirimi olarak kritik bir mekanizmadır. Bu eylem üç farklı zamanda gerçekleştirilebilir:

- Derleme zamanında (kaynak kod dokuma),
- Yükleme zamanında (ikili kod dokuma),
- Çalışma zamanında.

*Çalışma kapsamında yükleme zamanı dokuma (load-time weaving) yöntemi kullanılmıştır (Şekil 19).*

Çalışma zamanı dokuma işlemi uygulama çalıştırıldığına Java sanal makinesine sınıfların yüklendiği zamanda bağlantıların yapıldığı yöntemdir. Dokuma işlemi ilgili sınıfın yüklenmesine kadar ertelenir. Bu yöntem byte kodu üzerinde işlem yapabildiği için kaynak kodun elde olmadığı durumlarda da işe yaramaktadır. Birçok eylemde oldukça avantaj sağlayan bu durumdan faydalanabilmek adına bu dokuma yöntemi tercih edilmektedir.




Şekil 19 Yükleme Zamanı Dokuma İşlemi Grafiksel Gösterimi [32]

Çalışma zamanı dokuma yöntemi Java programlama dilinin "java agent" olarak adlandırılan teknolojisinden faydalanmaktadır. "Java agent" teknolojisi Java Sanal Makinesinin izlenmesinde ve profillemesinde anahtar rol oynamasının yanı sıra, yüklenme zamanında Java sınıflarını dinamik olarak düzenleyebilmesi ile oldukça başarılıdır.

Yükleme zamanı dokuma yöntemini gerçekleştirme adımları:

1. Bir uygulamayı yükleme zamanı dokuma yöntemiyle çalıştırmak için *java* komutunun *–javaagent* opsiyonunu kullanmak ve *aspectjweaver.jar* dosyasını ajan olarak belirtmemiz gerekmektedir.

```
java –javaagent:<yol>/aspectjweaver.jar [diğer opsiyonlar] <Main-Class>
```

2. Sanal makine ajanı başlatır. Başlatma sırasında, ajan classpath'de yer alan META-INF/aop.xml doyası içerisindeki tüm dosyalar yüklenir. Şekil 20 'de Aop.xml örneği gösterilmektedir.


Şekil 20 AspectJ dokuma işlemi için kullanılan aop.xml dosyası içeriği [32]

3. Listede yer alan ve dokuma işlemine dahil edilecek ve/veya saf dışı bırakılacak bağlamlar tespit edilerek ajan tarafından yükleme işlemi başlatılır.
4. Sistem normal çalışmasına başlar. Sonra sanal makine çalışma zamanında ihtiyaç duyulan sınıfları yükler.
5. Sanal makine ne zaman bir sınıf yüklenecek olsa ajanı haberdar eder. Ajan yüklenen sınıfı inceler ve ilgilenip ilgilenmediğine karar vererek hareket eder. Eğer ilgilendiği bir sınıf ise derlenmiş tavsiye yordamlarını devreye sokar.

Özet olarak; bir uygulamayı çalışma zamanında dokuma (weaving) işlemine tabi tutarken, kesişme noktası (pointcut) ve birleşme noktası (join point) ile birlikte, belirtilen metod imzasını yakalar ve tavsiye yordam (advice) ile ne yapılacağını belirleriz.

# 3. İLİŞKİLİ ÇALIŞMALAR

Tez kapsamında yapılan çalışmalara temel olması açısından referans olarak alınabilecek birçok destekleyici çalışma mevcuttur. Bu çalışmalar UML modelleri kullanılarak COSMIC işlevsel büyüklüğün hesaplanmasının nasıl desteklenebileceğine ilişkin önemli bilgiler içermekle birlikte, UML tabanlı çalışmalar kullanılarak büyüklüğün hesaplanmasının nasıl otomatikleştirilebileceğine dair çeşitli bilgiler ve öneriler de içermektedir. Oldukça önemli olan ve yol gösterici bilgiler içeren bu çalışmalar tezin içeriğinin zenginleşmesini sağlamakta ve önerilen yöntemin sağlam temellere oturmasına yardımcı olmaktadır. İlişkili çalışmalar belirlenirken arka planda temel aldığımız esaslardan biri, UML diyagramlarını çeşitli şekillerde manuel ya da otomatik COSMIC büyüklük ölçmede kullanan yazarların çalışmalarından faydalanma isteğidir.

Özet olması açısından, UML modellerini kullanarak COSMIC işlevsel büyüklüğün ölçülmesiyle ilişkili çalışmalar başlıklar halinde verilecek olursa;

1. COSMIC ve UML kavramları arasında bir eşleşme öneren çalışma: Bévo ve ark. [6]
2. Dizge diyagramlarını kullanarak kullanım senaryolarının taneciklik problemini adresleyen çalışma: Jenner ve ark. [7]
3. İşlevsel büyüklüğün ölçülmesinde otomasyon için UML kavramlarının açılmasına dayanan çalışma: Vilus ve ark. [34]
4. Kullanım durumu modeli üzerine inşa edilmiş ölçme çalışması: Habela ve ark. [33]
5. Rational Unified Process (RUP) yöntemi kullanılarak otomatik olarak COSMIC tabanlı işlevsel büyüklük hesaplaması: Azzouz ve ark. [11]
6. COSMIC büyüklük ölçümünü desteklemek için UML kullanılabilirliğini inceleyen çalışma: Lavazza ve ark. [15]
7. UML in İşlevsel Büyüklük ölçümüne nasıl destek olabileceğine ilişkin bir gereksinimler uzayı çalışması: Van den Berg ve ark. [35]
8. UML kullanım durumları ve dizge diyagramları üzerinden elle ölçmenin kolaylaştırılmasına ilişkin bir çalışma: Levesque ve ark. [13]

9. Dizge diyagramı elementlerinin COSMIC artefaktlarıyla eşleştirilmesini temel alan çalışmalar: Fehlmann ve ark. [9]

Referans alınan ilişkili çalışmaları izleyen paragraflarda sunulmuştur.

Bévo ve ark. [6] çalışmalarında COSMIC işlevsel büyüklük ölçme modeli ile UML sınıf ve kullanım durumu diyagramlarına ait kavramların eşleştirilmesi çalışmaları yapmışlardır. İşlevsel büyüklüğün hesaplanmasında kullanım durumlarından yola çıkarak büyüklüğün hesaplanması hedeflenmektedir. Yapılan eşleşme çalışmalarında, her bir kullanım durumunun (use case) bir işlevsel sürece, kullanım durumuyla ilişkili aktörün işlevsel kullanıcıya, kullanım durumlarındaki karşılıklı etkileşimlerin veri hareketlerine eşleştirilmesini önermişlerdir. Yönetim bilgi sistemlerini hedefleyen yönteme ilişkin yapılan durum çalışmalarıyla, önerilen yöntemle elde edilen sonuçların elle yapılan ölçümlerle arasındaki oranın %11 ila %33 arasında değişiklik gösterdiği sonucuna ulaşılmıştır. Yöntemde Metric Xpert adlı araç kullanarak çalışmalar gerçekleştirilmiştir.

Jenner ve ark. [7] yaptıkları çalışmada Bévo ile benzer şekilde UML diyagramlarını temel olarak almaları açısından paralel bir yöntem önermişlerdir. Çalışmalarında, Bévo'dan farklı olarak, UML dizge diyagramlarını temel alan Jenner ve ark.'nın ölçme işlemlerinde eşleşme yaptıkları konsept dizge diyagramlarına ait elemanlardır. Her bir işlevsel süreç bir dizge diyagramı olarak değerlendirilmekte ve işlevsel süreci temsil eden bir soyutlama katmanı olarak ele alınmaktadır. Dizge diyagramında yer alan aktör (işlevsel kullanıcı) ve sınıflar arası etkileşimler veri hareketleri olarak ele alınmıştır. Önerilen yöntemin işlevselliğini göstermek için bir araç geliştirilmiş ve durum çalışması ile desteklenmiştir. Hedef alan ise yönetim bilgi sistemleridir.

Levesque ve ark. [13] kullanım durumları ve dizge diyagramlarını kullanarak COSMIC büyüklük hesaplama işlemlerinin gerçekleştirilebileceğini gösteren bir çalışma yapmışlardır. Bu çalışmada bir işlevsel süreç içerisinde yer alan veri hareketleri iki kategoride ele alınmıştır. Bunlar; veri taşıyan hareket tipleri ve veri değiştiren hareket tipleridir. Veri değiştirme hareketi yazılım karmaşıklığının ölçülmesi ile doğrudan ilişkilidir. UML diyagramları ve COSMIC yöntemin eşleştirilmesi noktasında her bir kullanım durumu bir işlevsel sürece, dizge diyagramlarındaki her bir nitelik bir veri grubuna ve dizge diyagramları etkileşim

mesajları ise veri taşıma hareketlerine karşılık gelmektedir. Bévo ve Jenner yöntemlerinde olduğu gibi önerilen yöntemin hedef alanı yönetim bilgi sistemleridir. Levesque'nin yöntemi uzman tarafından elle yapılan bir ölçme şeklinde olup sadece UML diyagramları üzerinden yapılmaktadır. Durum çalışmalarında "Rice Cooker" örneğiyle karşılaştırma yapılarak bir ölçüm gerçekleştirilmiş ve önerilen yöntemle elle yapılan ölçüm arasında %8'lik bir fark oluştuğu gözlemlenmiştir.

Van den Berg ve ark. [35] UML'in kullanıcı gereksinimleri belirtimleri için oldukça uygun olduğunu belirtmektedirler. Çalışmalarında, kullanıcı gereksinimleri belirtimlerinin farklı düzeylerde detaylandırılabilecek şekilde UML tarafından desteklenebileceğini göstermek için bir gereksinimler uzayı önermektedirler. Bu gereksinimler uzayı, her düzeyde tutarlı olan belirli sayıda UML diyagramına dayanmaktadır. Yazarlar sistem sınırının kullanım durum diyagramlarıyla belirlenebileceğini savunmaktadırlar. Buna ek olarak, veri yapılarını ve kullanılan öznitelikleri ölçmek için sınıf diyagramının kullanılmasını önermektedirler. Son olarak, olay akış kavramını temel alan işlevsel süreçleri ifade etmek için aktivite diyagramlarının kullanılmasını önermektedirler. Önerilen yaklaşım küçük bir durum çalışması ("The Hotel Case") ile desteklenmiştir. Ölçme aşaması farklı kullanım durumlarını tanımlayan aktivite diyagramına dayanmaktadır. Her bir kullanım durumu için, yazarlar veri hareketlerinin sayısının nasıl hesaplanabileceğini göstermektedirler. Tüm veri hareketlerinin toplamıyla COSMIC büyüklük elde edilmektedir. Bununla birlikte, aktivite diyagramlarının kullanılmasındaki mantık tam olarak anlaşılamamaktadır.

Vilus ve ark. [34] işlevsel büyüklüğün hesaplanmasını kolaylaştırmak için bir spesifikasyonda yer alan kavramların örneklerinin genişletilmesini sağlayan bir yöntem önermektedirler. FSM otomasyonu bağlamında sundukları bu çalışma Bévo'nun çalışmasını tamamlayan bir öneridir. MetricXpert adlı aracı kullanarak bir prototip geliştirmiş olan yazarlar Rational Rose veya bir CASE aracıyla tanımlanmış spesifikasyonlardan otomatik işlevsel büyüklük hesaplayabilmektedirler. Prototip, MOF (Meta Object Facility) meta-modelleri elde edildikten sonra bütün ilişkili kavramları içeren veritabanı mahiyetindeki XMI dosyalarını kullanmaktadır. Üç ayrı kullanım durumu çalışmasıyla test edilen yöntem yaklaşık %20'lik bir aralıkta hata ile sonuçlar üretebilmektedir.

Habela ve ark. [33] COSMIC tabanlı yazılım ölçümlerini desteklemek için çalışmalarında UML kullanım durumu modellerine odaklanmaktadırlar. Yazarlar veri gruplarının karmaşıklığına bakmaksızın, veri özniteliklerinin ölçülmesi sürecine odaklanmaktadırlar. Getirdikleri yaklaşım işlevsel büyüklüğün ölçülmesini sağlamak için detaylı senaryoların kullanımından oluşmaktadır. Tekrarlayan işlevsel süreçlere dikkat çekilmektedir ve ölçme işleminde meydana gelebilecek tekrarlardan kaçınılabilmesi için «include» ve «extend» ve genelleştirme için farklı bir bağlantı oluşturulması gerektiğini vurgulamaktadırlar. Ayrıca önerdikleri kullanım durumu şablonunda UML spesifikasyonunda yer almayan iş kuralları, ön şartlar ve son-şartlar gibi bazı elemanların sağlıklı ölçüm için gerekli olduğunu belirtmektedirler.

Lavazza ve ark. [15] çalışmalarında UML modellerinin COSMIC yöntemin büyüklük hesaplamasını desteklediği ve UML dilinin yaygın kullanımı sebebiyle UML tabanlı büyüklük hesaplama yöntemlerinin önemini vurgulamışlardır. Çalışmalarında, yazarlar ayrıca COSMIC ve UML kavramlarının eşleştirilmesi üzerine de deneyler gerçekleştirmişlerdir. Bu kapsamda COSMIC yöntem kurallarına göre kolayca ölçümü yapılabilen UML modellerinin oluşturulması hakkında bilgiler verilmiş ve bu konuyu örnekleyen durum çalışması yapılmıştır. “Rice Cooker Revisited” olarak adlandırılan durum çalışmasında UML kullanımının COSMIC büyüklük ölçümündeki pratikleri nasıl geliştirdiğini göstermişlerdir. Çalışmada öne çıkan UML diyagramları, bileşen diyagramı ve kullanım durum diyagramıdır. Ayrıca bu diyagramlar kullanarak işlevsel olmayan kullanıcıların nasıl tespit edilebileceğine dair bilgiler verilmiştir.

Fehlmann ve ark. [9] çevik takımların çalışma rutinlerinden etkilenerek hızlı bir şekilde COSMIC büyüklük hesaplaması için UML diyagramlarının kullanılabileceğini öngörmüştür. Önerilerinde dizge diyagramlarının kullanılarak kolay ve hızlı bir şekilde COSMIC işlevsel büyüklüğün hesaplanabileceğini savunmaktadırlar. Makalelerinde öne çıkan nokta COSMIC yöntemin artefaktlarını dizge diyagramı elemanlarına eşlemeye çalışmalarıdır. Yaptıkları durum çalışmasıyla kanıtladıkları şey ise dizge diyagramında yer alan iki veri objesi arasındaki mesajların toplanarak sayılması ile veri hareketlerinin sayısı arasında bir fark olmadığıdır.

Paton [10] kaynak kod temelli çalışmasında, bir yazılımın veri akış tabloları şeklinde ifade edilmesinin işlevsel büyüklük analizinde oldukça kullanışlı olduğunu vurgulamaktadır. Bu amaçla yazılımları, 'uygulama kesitleme' yöntemine benzer bir yaklaşım kullanarak, dinamik ve statik kod analizi yöntemleri ışığında veri akış tablolarına dönüştürmenin, işlevsel büyüklüğün hesaplanmasında kolaylaştırıcı bir etken olacağı öne sürülmüştür.

Azzouz ve ark. [11] çalışmalarında, RUP sürecini kullanan ve Rational Rose ortamında geliştirilen yazılımların işlevsel büyüklüklerinin otomatik olarak ölçülmesinin yolunu açan bir yaklaşım ve araç önermektedirler. Söz konusu araç COSMIC ile UML kavram ve gösterimleri arasında eşlemeye dayalı olarak bir değerlendirme yapmaktadır. Önerilen yöntemle, geliştirme safhalarının üç farklı evresi için otomatik yazılım büyüklüğü elde edilebilmektedir. İş modelleme ve gereksinim analizi evresinde kullanım durum diyagramları ile analiz ve tasarımın ilk evrelerinde senaryolar üzerinden, analiz ve tasarımın ileri evrelerinde ise detaylandırılmış senaryolar üzerinden büyüklük ölçümü yapılabilmektedir. Söz konusu yöntem "Rice Cooker" durum çalışması üzerinden yapılan testlerle doğrulanmıştır.

Kusumoto ve ark. [5] UML dizge diyagramları yardımı ile COSMIC büyüklüğü otomatikleştirme çalışmaları yapmışlardır. Çalışmaları göstermektedir ki UML modelleri otomatik COSMIC büyüklük hesaplanması yöntemlerinde oldukça etkili ve başarılıdır. Önerdikleri yöntem yazılım kaynak kodunu kullanarak COSMIC işlevsel büyüklüklerin hesaplanmasında kullanılabilmektedir. Daha önce de IFPUG yönteminin otomatik hesaplanmasında kullanılan istatistiki yöntemleri düzenleyerek COSMIC yönteme uygulamışlardır. Durum çalışması ile desteklediklerini yöntemin etkin ve büyük oranda doğru sonuçlar ürettiğini belirtmektedirler.

İlişkili çalışmaların çıktı ve hedef alanlarının yanı sıra üzerinde yoğunlaştığı UML konseptine dair bilgiler Tablo IV' de gösterilmektedir.

Tablo IV Referans Çalışmalar Öznitelikler Tablosu

| **Öneri Sahibi** | **Uygulama Alanı** | **UML Konsepti** | **Otomasyon ve Doğruluk Varyasyon Oranı** | **Araç Destekli Otomasyon** |
|---|---|---|---|---|
| Bévo ve ark. | MIS | Kullanım durumları, senaryoları ve sınıf diyagramları | Evet;<br>- %11-33 | Metric Xpert |
| Jenner ve ark. | MIS | Kullanım durumları ve dizge diyagramları | Evet<br>- | Kestirimler için özel bir araç |
| Azzouz ve ark. | Gerçek Zamanlı | Kullanım durumları, senaryoları ve sınıf diyagramları | Evet<br>- | RUP COSMIC |
| Van den Berg ve ark. | MIS | Kullanım durumları, aktivite ve sınıf diyagramları | Hayır<br>- | Yok |
| Lavazza ve ark. | Gerçek Zamanlı | Kullanım durumları, bileşen, sınıf ve dizge diyagramları | Hayır<br>- | Yok |
| Kusumoto ve ark. | MIS | Dizge diyagramları | Evet<br>- %5-10 | Özel tasarım bir araç |
| Paton | MIS | Dizge diyagramları | Hayır<br>- | Yok |
| Felhmann ve ark. | MIS | Dizge diyagramları | Hayır<br>- | Yok |
| Habela ve ark. | MIS | Kullanım durumları | Hayır<br>- | Yok |
| Levesque ve ark. | MIS | Kullanım durumları ve dizge diyagramları | Hayır<br>- %50-85 | Yok |

# 4. OTOMATİK ÖLÇME YÖNTEMİ

Tez çalışması kapsamında önerilen yöntemin temel amacı COSMIC büyüklük hesaplama işleminin otomatik hale getirilmesidir. Bu bölümde, önerdiğimiz yöntemin temelleri ve nasıl gerçekleştirilebileceğine dair bilgiler verilecektir. Ayrıca yöntemin işleyişini gösterebilmek adına geliştirilen prototip üzerinde durulmuştur.

## 4.1. Önerilen İşlevsel Büyüklük Ölçme Yöntemi

İlişkili çalışmalardan da görüleceği üzere UML dizge diyagramlarının COSMIC ölçme yöntemiyle bağdaştırılabilmesi birçok açıdan avantaj yaratmaktadır. Bilindiği üzere UML dizge diyagramları yazılım geliştirme süreçlerinin ilk aşamalarında projenin anlaşılması ve tüm paydaşlarla ortak dil oluşturması açısından oldukça yararlıdır. Bu yarar sadece proje başlangıcında değil daha sonraki geliştirme aşamalarında da geçerliliğini sürdürmektedir.

COSMIC işlevsel büyüklük hesaplanması karşılaştırma yapılabilmesi adına çok önemli proje girdileri sunabilmektedir. Fakat daha önce geliştirilen yazılımların işlevsel büyüklüğü bilgisine sahip olunmaması, hiçbir dokümantasyon bilgisinin bulunmaması veya geliştirilen yazılım hakkında bilgi sahibi kimsenin kalmaması durumlarında yazılımın işlevsel büyüklüğünün nasıl hesaplanabileceği konusu büyük bir soru işareti oluşturmaktadır. Manuel ölçümlerin alacağı süre ve oluşturacağı maliyet geçmişe yönelik böyle bir çalışmanın yapılmasına engel teşkil edebilmektedir. Çalışmamızın temel motivasyonunu teşkil eden “Bir uygulamanın büyüklüğünü koda müdahale etmeden ve uygulamayı kapalı kutu gibi ele alarak nasıl ölçebiliriz?” sorusu bizi daha önce belirtilen durumları göz önünde bulundurarak değinilen problemlere çözüm bulabilmek adına iki farklı soruya yönlendirdi. Birinci sorumuz “çalışan bir uygulamadan UML dizge diyagramlarını elde edebilir miyiz?”, ikincisi ise “UML dizge diyagramlarını kullanarak COSMIC büyüklüğü otomatik olarak ölçebilir miyiz?” idi.

Sorulan sorular çerçevesinde literatür taraması yapılmasına karar verildikten sonra yapılan araştırmalar ve incelenen makaleler sonucunda Jenner [7] ve Lévesque [13] adlı araştırmacıların UML dizge diyagramları ve COSMIC büyüklük kavramlarını eşleştirme çalışmalarına ulaşıldı. Makalelerinde, yaptığımız

çalışmaya uygun önerilere değinen araştırmacıların, dizge diyagramlarının COSMIC işlevsel büyüklüğün ölçülmesi için uygun yeterlilikte olduğunu ispatlamış olduklarını gördük. Lévesque ve ark. [13] 'nın veri hareketlerinin UML dizge diyagramlarının mesajlarının eşleniği olarak ele alınabileceğini ve aynı diyagramda veri işleme hareketlerinin çeşitli şekillerde hata mesajları gibi ele alınabileceğini göstermesi oldukça umut vericiydi. Bütün bu bilgiler sonucunda, dizge diyagramlarında veri değişiminde kullanılan mesajların bir araya getirilmesiyle ilgili işlevsel sürecin büyüklüğünün hesaplanabileceği bilgisine ulaşmak, yapılacak çalışma için oldukça motive edici bir etmen olarak karşımıza çıktı.

Yapılacak çalışmanın fizibilitesini çeşitli araştırmacıların desteğiyle sağladıktan sonra, başlama noktamızdaki sorularımıza geri döndük. İlk sorumuz olan "çalışan bir uygulamadan UML dizge diyagramlarını elde edebilir miyiz?" sorusuna diğer kısıtları da göz önünde bulundurarak cevap ararken yapılan araştırmalar bizi bağlam yönelimli programlamaya yöneltti. Bağlam yönelimli programlamanın sunduğu olanaklar kullanılarak kod üzerinden dizge diyagramlarının elde edilebileceği bilgisine ulaşıldığında, aslında tüm düğüm çözülmüş bulunmaktaydı. (Bağlam yönelimli programlama hakkında detaylı bilgi 2.6 bölümünde yer almaktadır). İlk araştırmalarda tersine mühendislik yönteminin üzerine gidilmiş fakat yazılımsal olarak işlenebilir çıktıların elde edilmesinde yaşanan zorluklar ve UML dizge diyagramların tersine mühendislikle elde edilebilmesi için gereken manuel işlemlerin oluşturduğu iş yükü - zaman maliyeti nedeniyle vazgeçilmiştir.

Çalışmanın detaylarına ve ilgili teknolojilerin nasıl kullanıldığına dair bilgilerin detayına girmeden önce fikir vermesi açısından yapılan çalışmayı dört adımda özetlersek;

- Kesme Noktalarının Hazırlanması – Java platform ve AspectJ sözdizimi kullanarak birleşme noktalarının tanımlanması. Daha sonra veri hareketleri ve işleme eylemlerini yakalayabilecek kesme noktalarının tanımlanması. (Java, AOP ve COSMIC ölçüm yöntemi konusunda uzman tarafından)
- Dinamik Analizin Yapılması – Yükleme zamanında yapılan dokuma prosedürü ile hedef uygulamanın birinci adımda tanımlanmış tavsiye yordamlarıyla normal bir kullanıcı olarak kullanılması. (Normal kullanıcı/ölçüm yapan kişi tarafından)

- Dizgelerin Metin Formunda Elde Edilmesi – Ölçülmesi hedeflenen her bir işlevsel sürecin tetiklenmesi sonucu etiketlenmiş bir akış stilinde dizge diyagramının metin halinde elde edilmesi. (Normal kullanıcı/ölçüm yapan kişi tarafından)
- COSMIC Kurallarının Uygulanması ve Büyüklüğün Ölçülmesi - Metin formunda oluşturulan dizge diyagramlarındaki işlevsel süreçlerin tespit edilerek COSMIC kuralları çerçevesinde sayma işlemlerinin gerçekleştirilmesi (Normal kullanıcı/ölçüm yapan kişi tarafından)

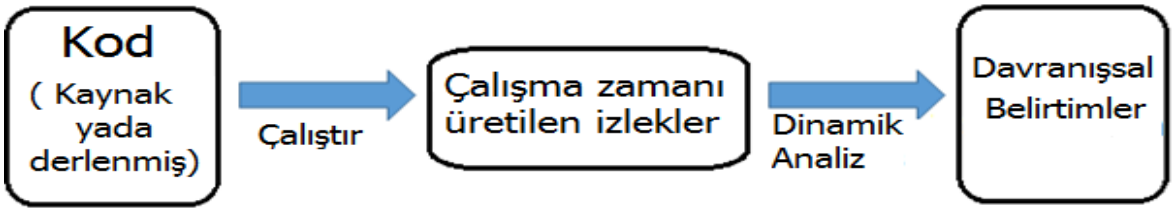
Birinci adım olarak tanımlanan kesme noktalarının hazırlanması eylemi, tamamıyla yazılım kodundan UML dizge diyagramı elde etmek için gerekli olan materyalin hazırlanması eylemidir. Bu eylem belirli bir kütüphane oluşturmak amacıyla da kullanılabilir veya daha önce hazırlanmış kütüphanelerin kullanılmasıyla da gerçekleştirilebilir.

Dizge diyagramlarını işleyip otomatik olarak COSMIC büyüklük hesaplama işlemini gerçekleştirebilmek için yazılımın soyut bir akış bilgisini işlenebilir formatta elde etmek gereklidir. Bunun için yapısal özellikleri bulunan ve metin biçeminde okunabilir çıktılar hazırlanmaktadır.

Belirtilen eylemin nasıl gerçekleştirildiğinin ayrıntılarına girmeden önce kaynak üzerinden yapılan işlemlerde ortaya çıkan, Jenner ve Bévo [7] [6] önerilerinde bahsettikleri “tetikleyici eylem” problemini aşabilmek adına, analiz yöntemine değinmek gereklidir.

Tetikleyici eylemin tespit edilmesi kodda var olabilecek ölü kod, gereksiz işlevler vb. gibi durumlar nedeniyle problem teşkil etmektedir. Bu problemi aşabilmek adına “Dinamik Analiz” yöntemi kullanılmıştır. Böylece çalışma zamanında ve kullanıcı tarafından tetiklenebilen işlevsel süreçler tespit edilmiş ve büyüklük hesaplamasında değerlendirmeye alınmıştır.

Dinamik analiz yöntemi, işlemci üzerinde koşturulan uygulamanın tahlil edilmesi işlemine dayanmaktadır. Ölçme eyleminin hedefi olan uygulamanın dinamik davranışları bu tür analizlerin temel hedefidir. Şekil 21' de dinamik analiz yönteminin soyut bir gösterimine yer verilmiştir. Dinamik analiz belirtildiği gibi iki adımdan oluşmaktadır; Birinci adım uygulamanın çalıştırılması, ikinci adım ise çalışma akışını belirten işletim izlerinin elde edilmesidir.




Şekil 21 Hedef uygulamanın dinamik analizi

Dinamik analizin temel hedefi ise uygulamanın çalışma zamanında gerçekleştirdiği eylemleri açığa çıkarmak ve bunları belirli bir kalıp içerisinde çıktı olarak sunmaktır. Bu eylem ile metod çağrıları ve nesne oluşturma gibi işlemlerin yakalanması ve etiketlenmesi gerçekleştirilmektedir. Etiketleme işlemi daha anlaşılır olarak kısaca şu şekilde ifade edilebilir; "Kullanıcı tarafından tetiklenen eylem sonucunda herhangi bir metod çağrısı yapıldığında, o çağrı ve alt çağrılarına ait izlekleri tanımla, ilgili çağrının imzasını yakala ve önceden belirlenmiş bir anahtar kelime (start, end, swing, jdbc vb.) kullanarak dosyaya yazdır."

Hedef yazılımın Şekil 21'de gösterildiği gibi gerçekleştirilecek dinamik analiz eylemiyle etiketlenmesi işinde AspectJ devreye girmektedir. Uygun kesme noktalarının tanımlanmasıyla, AspectJ'yi kullanarak tüm metod çağrılarının yanı sıra nesne oluşturma ve yürütme eylemlerini etiketleme fırsatı elde edilmektedir. Bu etiketlemeyi yapabilmek için uygun metod çağrısı örüntülerinin kesme noktalarıyla tanımlanması ve ilgili tavsiye yordamların hazırlanması gerekmektedir. Bu işlem, Java diline ve AspectJ'ye hakim uygulama geliştiriciler tarafından gerçekleştirilir.

Etiketleme işlemi için, Java programlama dili çerçevesinde meydana gelen ve ölçme eylemine konu olan metod çağrılarına ilişkin kesme noktaları ve tavsiye yordamlarına ihtiyaç vardır;

- Tanımlanan AspectJ kesme noktaları ile hedef metod çağrıları yakalanır.
- İşlevsel süreçlerin metin biçeminde dizge diyagramlarını içeren çıktılar, metod çağrılarının "önce, sonra" sonrasında çalıştırılacak tavsiye yordamlar ile oluşturulur.

Dinamik çalışma izlekleri üzerinden elde edilen metin formundaki dizge diyagramı çıktılarını elde ettikten sonra, ikincil eylem olarak COSMIC kuralların UML dizge diyagramlarına uygulanması aşamasına geçilebilir. Bu aşamanın temel adımları

aday işlevsel süreçlerin ve ilgili sınırların tespit edilmesidir. Çalışmamızda UML diyagramları ve kullanıcı ara yüzünden tetiklenebilen işlevlerin dinamik analiziyle, sınırların başlangıç noktaları kolaylıkla tespit edilebilmektedir. Çünkü dinamik analiz esnasında sınırların (çalışmada tetikleyici eylem) başlangıç noktası halihazırda etiketlenmiş durumdadır. Örneğin çalışmamızda kullandığımız kullanıcı sınırı Swing arayüzü olduğu için etiketimiz “<start:SWING>” şeklindedir. Bu ve bitiş etiketi arasında yer alan bütün alt çağrıların toplamının aday işlevsel süreci oluşturduğu düşünülerek hareket edilmektedir.

Tetikleyici eylem aracılığıyla tespit edilen her bir aday işlevsel süreç için, veri taşıma ve işleme eylemleri ele alınmaktadır. Bir işlevsel sürecin büyüklüğünün veri hareketleri (girdi, çıktı, okuma ve yazma) sayısının toplamından oluşması nedeniyle, tespit edilen hareket tipleri toplanarak büyüklük elde edilir.

Aday işlevsel süreçlerin değerlendirilmesi noktasında üç problem ortaya çıkmaktadır;

1. Birbirinin kopyası aday işlevsel süreçler.
2. Giriş noktası, tetikleyici eylemi aynı olan fakat dizge boyutu birbirinden farklı olan aday işlevsel süreçler.
3. COSMIC Manuel’ine göre işlevsel kullanıcı gereksinimi olarak değerlendirilmeyen dizgelerin oluşturduğu aday işlevsel süreçler.

Birinci problem işlevsel süreçlerin tespitinde dinamik analiz yapıldığı için, kullanıcının bilerek ya da bilmeyerek aynı tetikleyici eylemi birden fazla olacak şekilde kullanmış olmasından kaynaklanmaktadır. Bu da birbirinin aynı birden fazla aday işlevsel sürecin adaylar listesinde yer almasına sebebiyet verir. Göreceli olarak basit bir şekilde çözülebilecek bu problemde işlevsel süreçlerin dizge boyutları da birbiriyle aynı olduğundan tüm işlevsel süreçler taranarak birbirinin aynı dizgeler (bir kopyası kalacak şekilde) aday listesinden düşürülerek problem çözülmektedir.

İkinci problem ise aynı işlevsel sürecin birden fazla akışının olabilmesi durumunda, seçeneklere bağlı olarak dallanabilmesinden kaynaklanmaktadır. Karmaşıklık oluşturan bu problemin basit bir çözümü bulunmamakla birlikte işlevsel büyüklüğün hesaplanmasında hedef en yakınsayan sonucun elde edilmesi olduğundan, birbirinin aynı fakat farklı dizge boyutuna sahip aday işlevsel

süreçlerden en büyüğü seçilerek toplam işlevsel puana dahil edilebilir. Ya da ölçüm yapan kişiye karmaşıklık oluşturan işlevsel süreçler sunularak, toplam puan üzerinde inisiyatif kullanabilme yetkisi verilmesiyle bu durum aşılabilir. Temelde bu problem Eriksson, H.-E., ve M. Penker'in çalışmasında [23] gösterdiği üzere dizge diyagramları üzerinden hesaplama yönteminin en temel problemidir. Geliştirilen uygulamada, ölçülecek yazılımda bu problemle karşılaşıldığında ölçüm yapan kişi uyarılarak oluşabilecek hatadan haberdar edilmekte ve inisiyatif alması beklenmektedir.

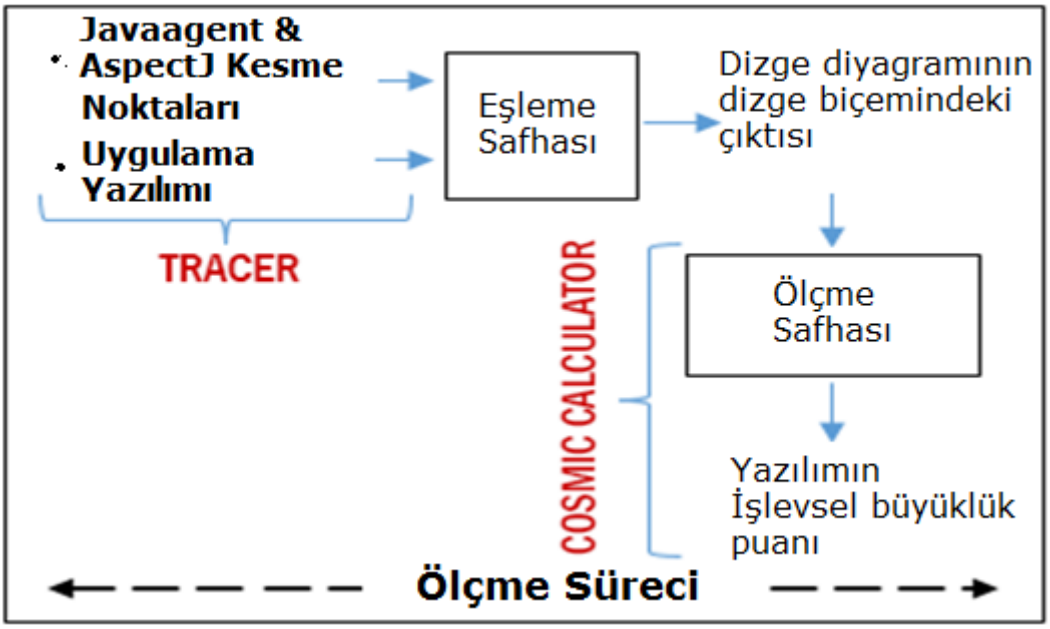
Üçüncü problem ise kullanıcının arayüzden erişim yaparak işlevsel süreçleri tetikleyici eylemlerde bulunması, dolayısıyla kontrol grupları, eylemleri vb. hareketlerin de aday süreçler olarak yapısal metin formundaki dizge diyagramlarına çıktı olarak yansımasıdır. Bunların temizlenmesi için aday işlevsel süreçlerden ayrıştırılması gerekmektedir. Gözden kaçan bir durum olarak oluşturabileceği sıkıntılardan dolayı, bir problem olarak ele alınmalıdır. Çalışmamızda, yöntem imzaları ve veri hareketleri incelenerek bu tür süreçler aday olarak değerlendirilmektedir.

## 4.2. İşlevsel Puan Ölçüm Aracı Prototipi

Bu bölümde önerilen koddan elde edilen UML dizge diyagramları üzerinden COSMIC işlevsel büyüklüğün hesaplanması yönteminin uygulanabilirliğini ve maliyet-etkinliğini değerlendirebilmek için geliştirilen prototip hakkında bilgiler verilmektedir.

Önerilen yöntem hakkında kavram kanıtlama işlemi için, "**Cosmic Solver**" adında bir prototip hazırlandı. Araç, Java programlama dili kullanılarak geliştirildi. Temel olarak iki bileşene sahip olan uygulama iki işlevi gerçekleştirmektedir. Bu bileşenler;

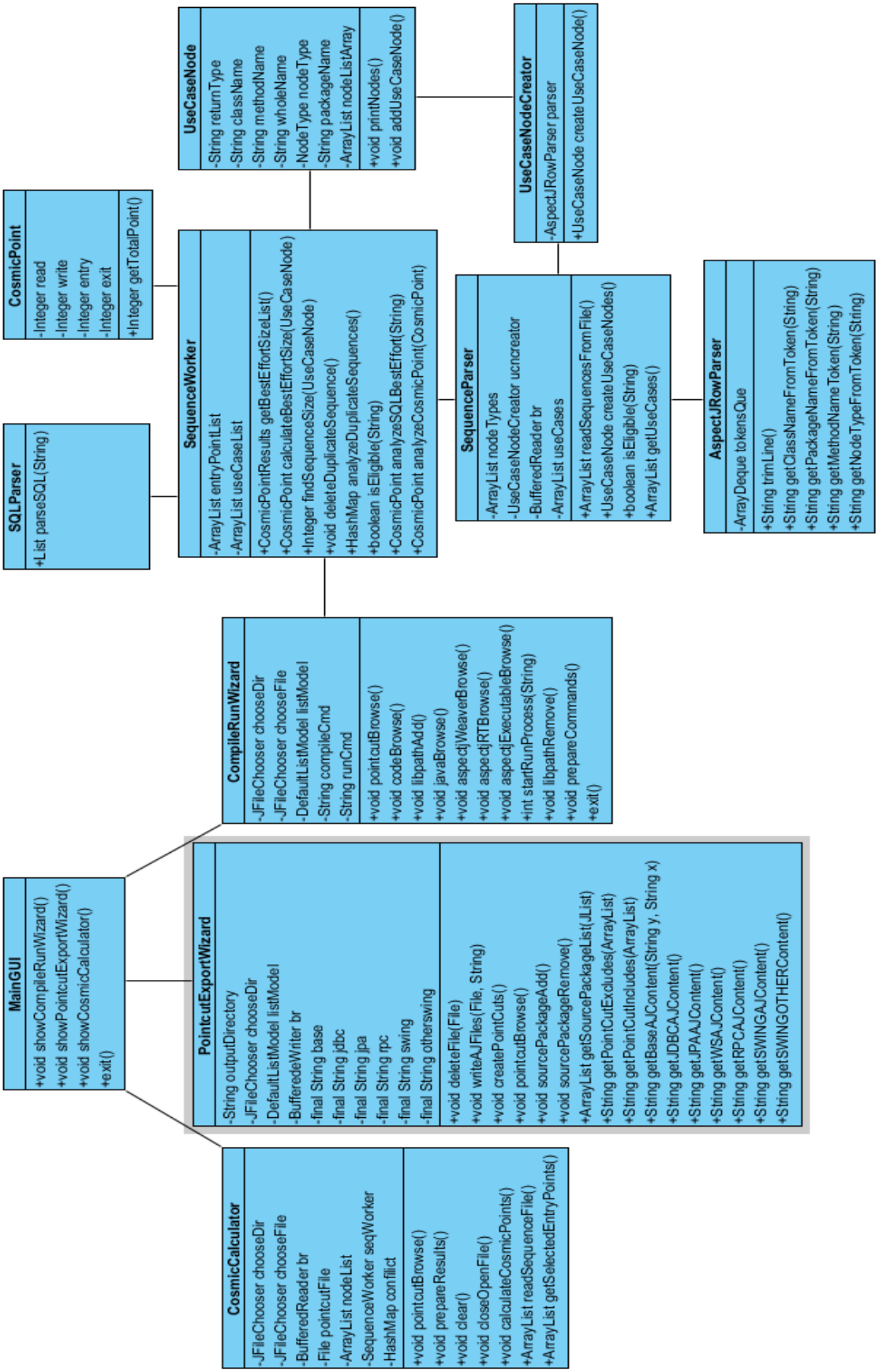
- "**Tracer**"; dinamik analiz yöntemiyle uygulamanın dizge diyagramlarını elde etmek için gerekli AspectJ ilişkili işlemlerde yardımcı olmaktadır.
- "**Cosmic Calculator**"; AspectJ yardımıyla elde edilen dizge diyagramlarının metin sürümünü işleyerek kurallar çerçevesinde COSMIC büyüklüğün hesaplanmasını sağlamaktadır.



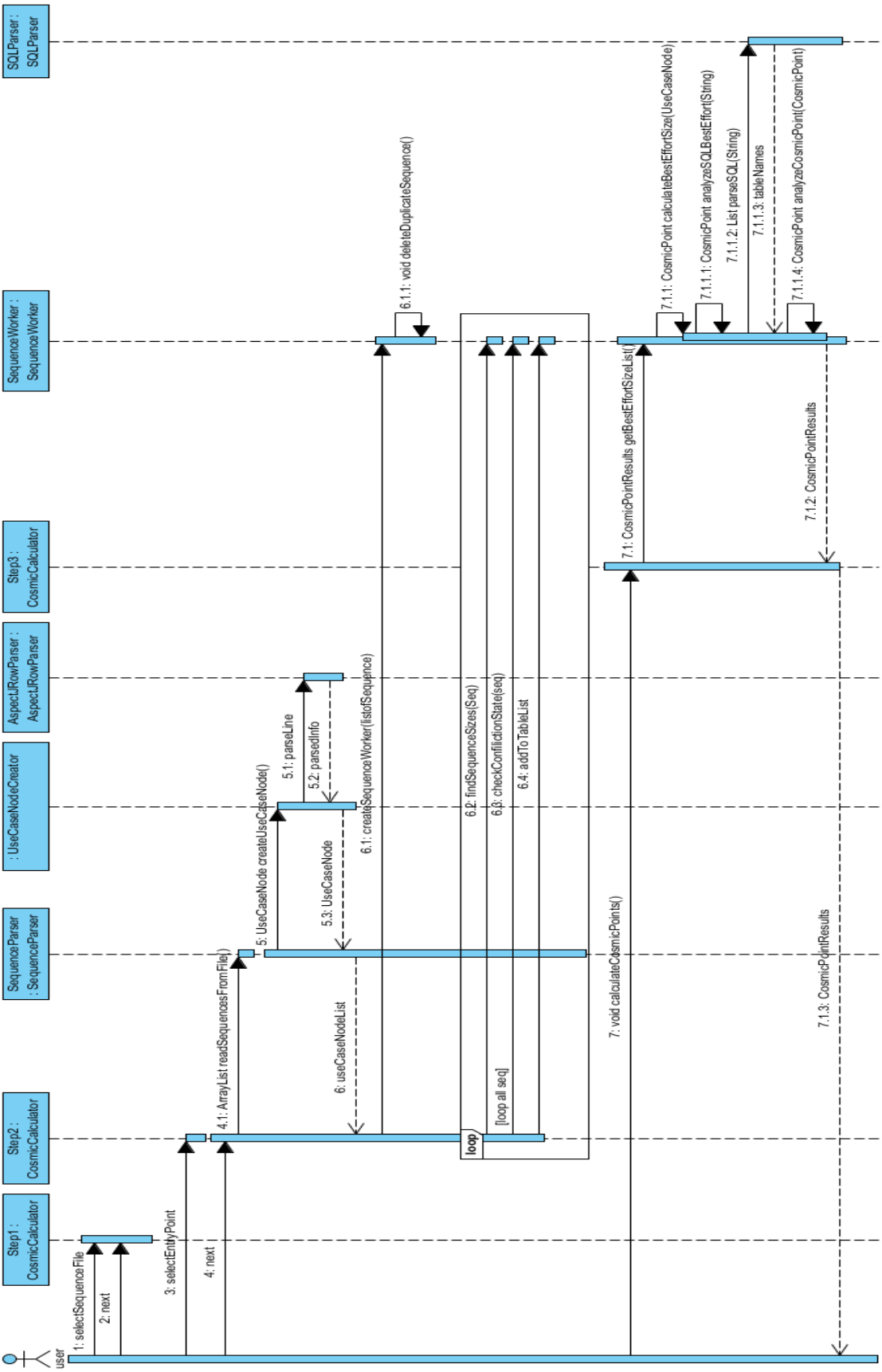

Şekil 22 Prototip Uygulama Bileşenleri

Daha iyi anlaşılabilmesi için grafiksel olarak göstermek gerekirse; Uygulamanın barındırdığı iki temel bileşenin sunduğu eylemlerin kullanım durumları Şekil 23'te gösterilmektedir. Şekil 24'te uygulamada yer alan temel sınıflarla ilgili UML diyagramı, Şekil 25'te ise alt bileşenlerden "Cosmic Calculator" dizge diyagramı yer almaktadır.


Şekil 23 CosmicSolver Prototip Uygulaması Kullanım Durumları




Şekil 24 CosmicSolver Prototip Uygulaması Sınıf Diyagramı




Şekil 25 CosmicSolver Prototip Uygulaması - Calculator Bileşeni Sınıf Diyagramı

### 4.2.1. Varsayım ve Kısıtlar

**Genel Kısıtlar**

“Manuel 1.3 - COSMIC Yazılım Bağlam Modeli İlkeleri” ölçülecek yazılım tarafından karşılanmalıdır [1]. Bu ilkeler;

a) Yazılım donanım tarafından sınırlandırılmıştır.

b) Yazılım tipik olarak katmanlar halinde yapılandırılmıştır.

c) Bir katman birden fazla eş katman içerebilir ve yazılımın parçaları eş bileşenlerden oluşabilir.

d) Ölçülecek her yazılım, ölçme kapsamıyla belirlenmelidir.

e) Ölçülecek bir yazılımın parçasının kapsamı ölçme amacına bağımlıdır.

f) Bir yazılım parçasının kullanıcısı, ölçülecek işlevsel kullanıcı gereksinimlerinden belirlenir.

g) Bir yazılım parçası kendi işlevsel kullanıcıları ile çeşitli şekildeki veri hareketleri aracılığıyla etkileşime geçer.

h) Yazılım İKG’leri farklı düzeyde taneciklik seviyesine sahip olabilir.

i) Ölçümlerin gerçekleştirildiği taneciklik düzeyi işlevsel süreçler olmalıdır.

j) Eğer işlevsel sürecin taneciklik düzeyini ölçmek mümkün değilse, yazılımın İKG’leri kestirim yaklaşımıyla ölçülmelidir ve işlevsel süreçlerin taneciklik düzeyine ölçeklenmelidir.

“Manuel 1.4 - Genel Yazılım Model İlkeleri” ölçülecek yazılım tarafından karşılanmalıdır [1]. Bu ilkeler;

a) Yazılım veri girdilerini işlevsel kullanıcılarından alır ve çıktılarını işlevsel kullanıcılarına gönderir.
b) Ölçümü yapılacak bir yazılım parçasının işlevsel kullanıcı gereksinimleri biricik işlevsel süreçlere eşlenebilir.
c) Her bir işlevsel süreç alt süreçlerden oluşur.
d) Bir alt süreç ya bir veri taşımadır ya da bir veri işleme eylemidir.

e) Bir işlevsel kullanıcı tarafından bir Giriş veri hareketi eylemiyle tetiklenen her işlevsel süreç bir olayı betimler.
f) Bir veri hareketi tek veri grubunu taşır.
g) Eşsiz bir veri öznitelikleri kümesinden oluşan bir veri grubu bir tek ilgi objesini oluşturur.
h) Bir işlevsel süreç hiç yoksa bir Giriş veri hareketine eşlik eden bir Yazma ya da Çıkış veri hareketinden oluşan iki veri hareketini içerir.
i) Ölçme hedefleri olarak bir kestirim, veri manipülasyonu alt süreçlerini ayrıntısıyla içermez; herhangi bir veri manipülasyonu işlevselliği ilişkili olduğu veri hareketinin sorumluluğundadır.

**Prototip Uygulama Kısıtları**

Uygulama geliştirilirken hedefe uygun olarak altta belirtilen kısıtlamalar dahilinde geliştirilmiştir. Önerilen yöntemi test ederken sonuçların doğruluğunu etkileyecek kısıtları belirtmek gerekmektedir. Bu kısıtlar;

- Büyüklük ölçümü yapılacak uygulama veritabanı işlemlerinde Persistence API kullanıyor ise, çalışma izlekleri üzerinden dizge diyagramlarını elde etmek için yapılan çalışma öncesinde bellekleme (caching) mekanizmasının kapatılması gerekmektedir. Eğer bu mekanizma kapatılmaz ise, Persistence API'ın birçok veriyi bellekten getirebilmesi durumu ortaya çıkacak ve veritabanı sorguları çalıştırılmayacaktır. Etiketleme yapılarak değerlendirmeye alınacak veritabanı eylemleri yakalanamayacağından, doğru sonuçlar için kapatılması şarttır.
- COSMIC Yöntem El Kitabı 3.0: İş Uygulamaları Rehberinde belirtilen çoklu-düzey birleştirme işlemleri içeren raporlamalar, prototip uygulama tarafından adreslenerek ele alınmamıştır. Eğer bir uygulama bu tür raporlar ağırlıklıysa elde edilecek sonuçlar güvenli olmayacak ve doğruluk oranı kötü yönde etkilenebilecektir.
- Çoklu-izlek operasyonları ele alınmamıştır. Eğer uygulamada işlevsel süreçler çoklu-izlek yeteneğine sahip ve aynı anda birden fazla izlek tarafından çalıştırılıyorken ölçüm yapılırsa metod çağrıları birbirleriyle karışacağı için sağlıklı sonuçlar elde edilemeyecektir. Bu durumda sonuçların kesinliği etkilenecektir.

**Prototip Uygulama Profili**

Bu uygulamanın temel hedefi Java iş uygulamalarında COSMIC büyüklük ölçümünün otomatik olarak gerçekleştirilmesidir. Kısa profil bilgileri maddeler halinde sıralanmıştır.

1. Hedef, uygulama katmanında hizmet veren ve Java SE 1.5+ ile geliştirilen Java iş uygulamalarıdır.
2. Prototip, JAX-RPC, JPA, JDBC standartlarını desteklemektedir.
3. Uygulama ölçüm sınırı olarak, Java SWING Grafik arayüzü desteklenmektedir, dolayısıyla işlevsel kullanıcıları bireysel aktörlerdir.
4. UML dizge diyagramının metin versiyonunu girdi olarak kullanmaktadır. Toplu işlem uygulamaları henüz desteklenmemektedir.
5. Dosya (file) operasyonları desteklenmemektedir.
6. Çoklu-izlek karakteristiğindeki uygulamalar ve paralel işlemler ele alınmamıştır.
7. COSMIC ölçme kısıtlarını sağlayan uygulamalarda çalışabilmektedir.
8. Hedef uygulama iki ve üç katmanlı mimariye sahip yazılımların ölçümünü desteklemektedir.
9. Dizge diyagramı Java Swing arayüzü üzerinden tetiklenen kullanıcı gereksinimlerini desteklemektedir. (Kesme noktaları geliştirilerek diğer arayüzler de desteklenebilir.)
10. Kaynak kodu olmayan derlenmiş Java uygulamalarıyla da çalışabilmektedir.
11. Geliştirilen uygulama bir prototip bir uygulamadır.
12. Uygulama iki bileşenden oluşmaktadır; *Tracer* (kesme noktası oluşturucu, derleyici), *Cosmic Calculator*.

### 4.2.2. Prototip Bileşeni : Tracer

*Tracer* adlı uygulama bileşeni bir yardımcı uygulama olup çalışma izlekleri üzerinden nesneler arası etkileşimleri (mesajları) yakalamak için gerekli altyapının hazırlanmasını sağlamaktadır. Temelde önceden tanımlanmış AspectJ kesme noktalarını ve önce/sonra tavsiye yordamlarını kullanmaktadır. Önerilen yöntemin büyük bir kısmı dizge diyagramlarının metin formda elde edilebilmesi eyleminden meydana geldiği için önemli bir bileşendir.

*Tracer* bileşeni, dizge diyagramının yapısal metin formunda oluşturulabilmesi için hedefe uygun olarak uzman tarafından belirlenmiş ve birleşme noktalarında tanımlı kesme noktalarını kullanmaktadır. Metod çağrılarının meydana geldiği kesme noktalarının öncesinde ve sonrasında çağrılan tavsiye yordamlarla metod imzası çıktı olarak kaydedilmektedir. AspectJ, tüm sistem ve kullanıcı çağrılarını yakalayabilme kapasitesine sahiptir. Fakat tüm çağrıların yakalanmasının COSMIC büyüklüğün hesaplanmasında bir anlam ifade etmemesi nedeniyle, sadece gerek-yeter şarta sahip ve kullanıcı alanına ait metod çağrılarının yakalanmasıyla optimal dizge diyagramına ulaşılması hedeflenmektedir.

*Tracer* bileşeni altında gerçekleştirilen işlemlerin önemini daha iyi anlayabilmek için dizge diyagramları üzerinden COSMIC büyüklük hesaplaması yöntemleri üzerinde çalışan diğer araştırmacıların önerilerinde değindiği soruna göz atmak gereklidir. Örneğin; Bévo ve Jenner [6] [7]'in çalışmalarındaki temel farklılık, "tetikleyici eylem" konseptinden kaynaklanmaktadır. Problem teşkil edebilme potansiyeline sahip bu durum COSMIC büyüklük ölçme yönteminin otomatikleştirilmesinde oldukça önemlidir. *Tracer* bileşeni altında gerçekleştirilen eylemler tetikleyici eylem problemini çözebilmesi nedeniyle büyük önem arz etmektedir.

Tetikleyici olaydan başlayarak, giriş noktaları, veritabanı çağrıları, diyalog kutuları vb. operasyonları etiketlenmiş şekilde içeren ve metod imzalarından oluşan dizge diyagramı formundaki çıktıları elde etmek için AspectJ kesme noktaları kullanılmaktadır. Bu kesme noktalarının önceden tanımlı olarak hazırlanması gerekmektedir. Daha önce de belirtildiği gibi bu kesme noktalarının tespiti için Java, AspectJ ve COSMIC hakkında bilgi sahibi olmalıdır. Prototip uygulamada halihazırda kullanılan kesme noktaları Tablo V'de yer almaktadır.

Tablo V Veri Hareketlerini Yakalamak İçin Tanımlanmış Kesme Noktaları

| **ETİKET** | **KESME NOKTASI** |
|---|---|
| SWING: | execution(* java.awt.event.ActionListener+.actionPerformed (..)) |
| DIALOG: | call(* javax.swing.JComponent+.show*(..)) |
| JDBC: | execution(* java.sql.Statement.exec*(..)) |

| | |
|---|---|
| | call(* java.sql.Statement.exec*(..)) |
| JPA: | execution(* java.sql.Connection.prepareStatement(..))<br>call(* java.sql.Connection.prepareStatement(..)) |
| JAX-RPC: | call(* javax.xml.rpc.Service+.createCall(..)) |
| OTHER: | execution(* <PackageName>..*(..)) \|\|<br>call(* <PackageName>.<methodName>(..)) |

Java programlama diliyle geliştirilmiş herhangi bir uygulama Tablo V'de kesme noktaları şeklinde tanımlanmış metod imzalarını içerebilmektedir. Çok daha geniş bir yelpazeye sahip olan Java işlevselliği, farklı standartlar çerçevesinde arayüzler sunması nedeniyle önceden tanımlanmış bu kesme noktalarının tam bir ölçüm için yeterli olmayacağı açıktır. Bu durumda *Tracer* bileşeninin daha etkin çalışabilmesi ve veri hareketlerini tespit etmek için daha geniş kapsamlı bir biçimde dizge diyagramları oluşturabilmesi adına, yeni kesme noktalarının tanımlanması bir gerekliliktir.

Her bir kesme noktası bir tavsiye yordamla ilişkilendirildiğinden, *Tracer* uygulamasında yer alan bu yordamlara da değinmek gereklidir. Daha önce belirtildiği üzere AspectJ üç çeşit tavsiye yordamı desteklemektedir. Önerdiğimiz yöntemde önce (before) ve sonra (after) tavsiye yordamları kullanılması gerek ve yeterlidir. Kesme noktasıyla belirlenmiş metod imzası yakalandığında, metod çalıştırılmadan önce ve metodun işlemi bittiğinde, çağrılan tavsiye yordamlarla metin formunda dizge diyagramları elde edilebilmektedir.

Tablo VI AspectJ Tavsiye Yordamları & Yapısı (JDBC Kesme Noktaları İçin Tanımlanmış Tavsiye Yordamlar ve Yapıları)

```
before() : jdbcCall() || jdbcExecution() {
        [Start:{tag}>] <print signature>
}
after() : jdbcCall() || jdbcExecution() {
        [End:{tag}>] <print signature>
}
```

Tablo VI belirtilen tavsiye yordam kalıbını kullanarak oluşturulan dizge diyagramı çıktılarının yapısı Tablo VII de belirtildiği gibi olacaktır.

Tablo VII Trace Log Çıktı Yapısı

```
<Start:tag>> returnType PackageName. ClassName. MethodName([Parameters])
 [(if exist)SQL Statement]
<End:tag>> returnType PackageName. ClassName. MethodName([Parameters])
```

Yükleme zamanında yapılan dokuma işlemiyle çalıştırılan hedef uygulama ile gerçekleştirilen tüm etkileşim, kesme noktaları ve tavsiye yordamları yardımıyla metin formunda dizge diyagramlarının oluşturulmasını sağlamaktadır. Oluşturulan dizge diyagramı metod çağrılarının ağaç yapısında çıktılarını oluşturmaktadır. Her bir metod çağrısı ve varsa alt çağrıları “Start” ve “End” etiketlerini taşımaktadır. Bu bilgi dizge diyagramları üzerinden COSMIC kurallarının uygulanması esnasında önemli bir bilgidir.

### 4.2.3. Prototip Bileşeni : COSMIC Calculator

İkinci bileşen olan “COSMIC Calculator”, metin formunda UML dizge diyagramlarını girdi olarak kullanıp tespit edilen aday işlevsel süreçlerin COSMIC büyüklüklerinin hesaplanması için ilgili kuralları işletip, sonucu hesaplamaktadır.

COSMIC işlevsel puanlar hesaplama işlemi üç adımda gerçekleştirilmektedir;

- Aday İşlevsel Süreçlerin Analizi
- Uygulama Sınırlarının Tanımlanması
- İşlevsel Süreçlerin Değerlendirilerek Büyüklüğün Hesaplanması

### Aday İşlevsel Süreçlerin Analizi

Dizge diyagramları dinamik analiz yöntemiyle elde edilmektedir. Kullanıcı kontrolünde gerçekleşen bu eylem nedeniyle kullanıcı istemsiz olarak birden fazla olacak şekilde belirli bir işlevsel kullanıcı gereksinimini kullanmış ve dolayısıyla ilgili dizge diyagramı çıktı dokümanında birden fazla yer almış olabilir. Birbirinin kopyası aday işlevsel süreç dizgelerinin oluşması sebebiyle sonuçların etkilenmemesi için bu durum dikkate alınmakta ve analiz aşamasında kontrol altında tutulmaktadır. Diğer bir olası problem, hedef yazılımdaki işlevsel süreçler, COSMIC ölçüm el kitabında vurgulanan taneciklik (granularity) yapısını bozucu dallanmalara sebebiyet veren bir karakteristik taşıması nedeniyle aynı metod imzasına sahip birden fazla farklı işlevsel büyüklükte süreçlerin oluşturduğu bir problemdir. Bu problem birden fazla kopya probleminden çok daha büyük bir sorun oluşturması nedeniyle ilgili durumlar tespit edilmekte ve aday işlevsel süreç işaretlenerek kullanıcı bilgilendirilmektedir.

### Uygulama Sınırlarını Tanımla

İşlevsel kullanıcı ya da harici uygulamaların sistem ile etkileşime geçtiği arayüzler, uygulamaların COSMIC büyüklüğünün ölçümünde sınır olarak tanımlanır. Sınırlar çerçevesinde yapılan büyüklük ölçümleri ile hangi işlevselliğin puanlamaya dahil edileceği ya da kapsam dışı bırakılacağı belirlenmektedir.

Ölçümü gerçekleştirilecek olan hedef uygulamanın kaynak kodu ve test senaryoları yazılabilme imkanı mevcut ise Kusumoto ve ark. [5]'nın önerdiği gibi işlevsel süreçlerin giriş noktalarının tanımlanıp büyüklük hesaplamasının yapılabilmesi oldukça kolaylaşmaktadır. Fakat çeşitli sebepler (koda ulaşılamadığı, dokümantasyon problemleri olduğu veya kod hakkında bilgi sahibi kimsenin olmadığı durumlar) nedeniyle test senaryolarına erişebilme imkanı yok ise aday işlevsel süreçlerin koddan tespit edilememesi problemi ortaya çıkmaktadır. Tetikleyici eylem problemi olarak da tanımlanan bu problem dizge diyagramları

üzerinden büyüklük ölçümünün önündeki en büyük engeli teşkil etmektedir. Bu problemi gidermek için, daha önce belirtilen dinamik analiz yöntemi kullanılmakta olup, kullanıcı etkileşimiyle tetiklenen eylemler girdi noktaları olarak etiketlenmektedir. Etiketlenmiş bu noktalar üzerinden aday işlevsel süreçler tespit edilmesi işlemleri gerçekleştirilmektedir.

Örneğin, ölçülecek uygulama Swing ya da AWT grafiksel arayüzü kullanıyorsa, kullanıcı bu arayüzle etkileşime geçtiğinde tetiklenen ilk temel işlev uygulama sınırı olarak tanımlanır. Eğer uygulama işlevsel gereksinimlerin tanımını web tabanlı Java uygulamalarında olduğu gibi bir dosyada (web.xml) tutuyorsa bu dosyadaki işlevlerin yakalanmasıyla başlangıç noktaları tespit edilebilir. Eğer arayüzler belirtilen tiplerden farklı ise tüm iş programcıya kalmaktadır ve test senaryoları yöntemi kaçınılmaz olmaktadır.

**İşlevsel Süreçlerin Değerlendirilerek Büyüklüğün Hesaplanması**

İşlevsel sürecin COSMIC Ölçüm El Kitabında [1] yapılan tanımına göre, bir işlevsel kullanıcı gereksiniminin işlevsel süreç olarak tanımlanabilmesi için, o gereksinimin işlevsel kullanıcı dünyasından tetiklenen bir eylem ile ve bağımsız bir şekilde çalıştırılabilmesi gereklidir. Buna ek olarak, kullanıcı gereksiniminin ana işlevi gerçekleştiğinde, tetikleyici eyleme bir geri bildirimde bulunmalıdır.

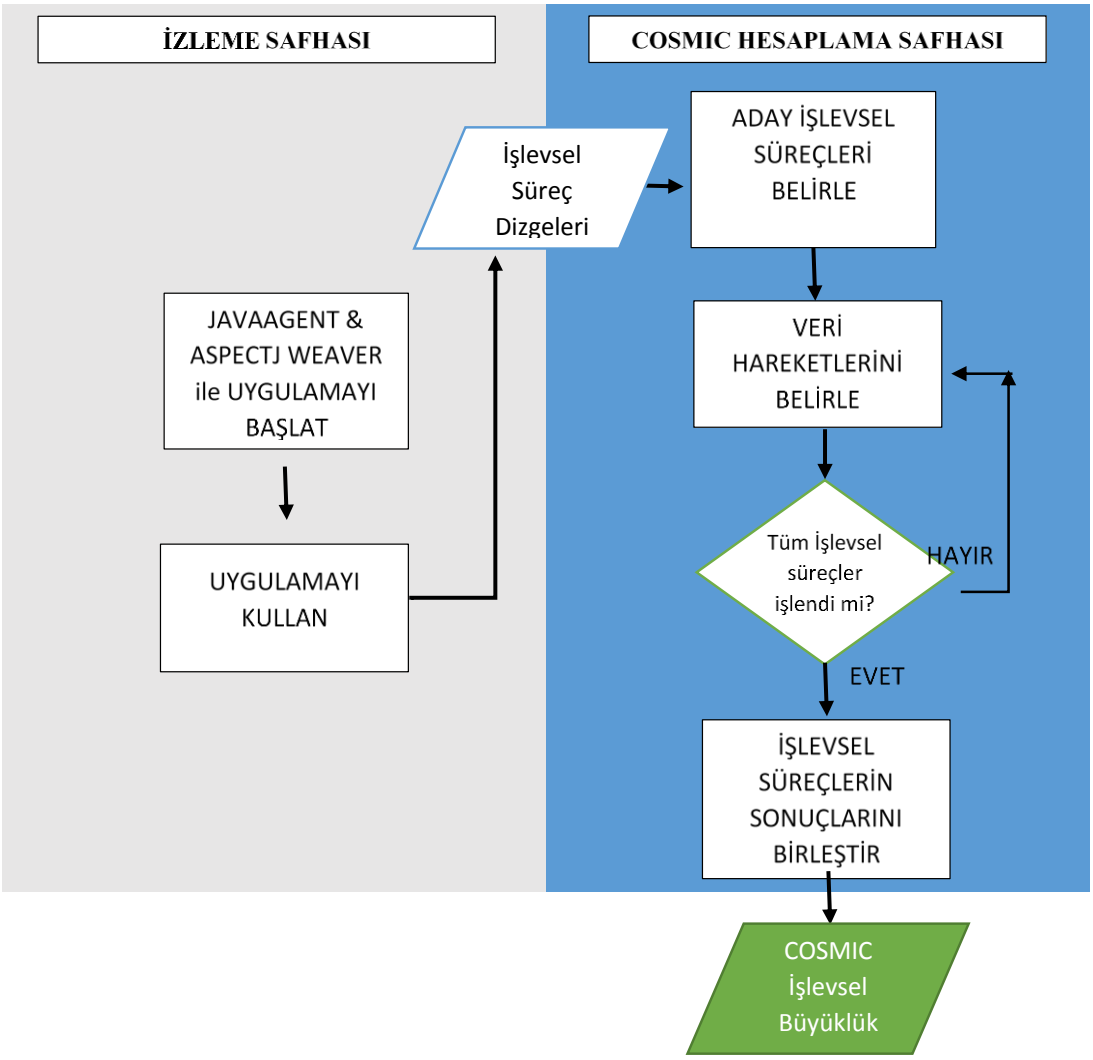
İşlevsel süreçler hakkında belirtilen gereklilikler, önerdiğimiz yöntemle kolayca karşılanabilmektedir. Etiketlerin yardımıyla belirlenen işlevsel süreçler ve veri hareketleri tespit edilerek ölçme kuralları uygulanmaktadır.

Uygulama derlenmiş AspectJ kesme noktalarıyla yükleme zamanı dokuma yöntemiyle çalıştırılıp normal bir kullanıcı olarak kullanıcı arayüzü etkileşimleri ile kullanıldıktan sonra elde edilen UML dizge diyagramı formundaki çıktılar “Cosmic Calculator” ile hesaplanması için hazır hale gelmiştir.

“Cosmic Calculator” girdi olarak kullandığı metin formundaki dizge diyagramlarını işleyerek, aday işlevsel süreçleri belirler. Daha sonra olası eşlenik işlevsel süreç adayları ayıklanır ve etiketlenmiş girdilerin yardımıyla COSMIC ölçüm kuralları ve altta belirtilen kriterler uygulanır. Sonuçta, işlevsel süreçlerin listesi ve COSMIC büyüklüğe ilişkin veri hareketleri büyüklükleri elde edilir.

Uygulamada her bir aday işlevsel sürecin değerlendirilmesi ve o sürecin COSMIC İşlevsel Süreç olup olmadığına karar verilmesinde dikkate alınan kriterler;

- Bir İKG kümesinde gerçekleşen eşsiz, birleşik ve bağımsız şekilde çalıştırılabilir veri hareketleri üzerinde mi işlem yapmaktadır.
- Bir eylem tarafından mı tetiklenmiştir (tetikleyici olay)?
- Tetikleyici eylem yazılım sınırı ötesinden mi gerçekleşmiştir?
- Tetikleyici eyleme karşı sürecin tümü çalıştırılarak geribildirim oluşturulmakta mıdır?




Şekil 26 *CosmicSolver* Hesaplama Adımlarının Görsel ifadesi

# 5. UYGULAMALAR

## 5.1. Ön Uygulama

COSMIC büyüklük ölçümü yapılacak yazılım tipik bir Java Masaüstü Uygulaması olup, CRUDL (Create, Read, Update, Delete ve List) operasyonlarının gerçekleştirilebildiği personel ve ödeme veritabanı uygulamasıdır. Nesneye yönelik yaklaşımla geliştirilen uygulama, klasik iş uygulamalarının temel operasyonlarını örnekleyerek önerilen yöntemin doğrulanması amacıyla geliştirilmiştir.

Java uygulamasının kapsamı, personel ve ödeme bilgilerinin eklenmesi, silinmesi, güncellenmesi, aranması ve listelenmesi işlemlerini kapsamaktadır. Grafiksel bir arayüze sahip olan uygulamanın işlemleri kullanıcı etkileşimiyle gerçekleştirilmektedir. Dolayısıyla, işlevsel gereksinimler kapsamında başarı ve hata mesajları yine görsel arayüz üzerinden sunulmaktadır.

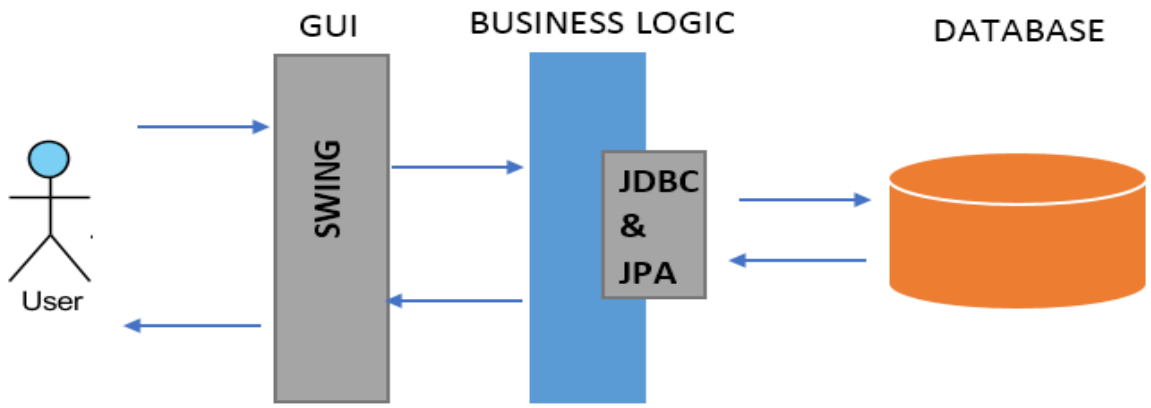
**Hedef Uygulama Bilgileri**

Önerilen yöntem ve geliştirilen aracın test edilmesi için gerçekleştirilecek ölçüm için kullanılacak hedef uygulama hakkında kısa bilgiler Tablo VIII ’de yer almaktadır.

Tablo VIII Hedef Uygulama Hakkında Kısa Bilgiler

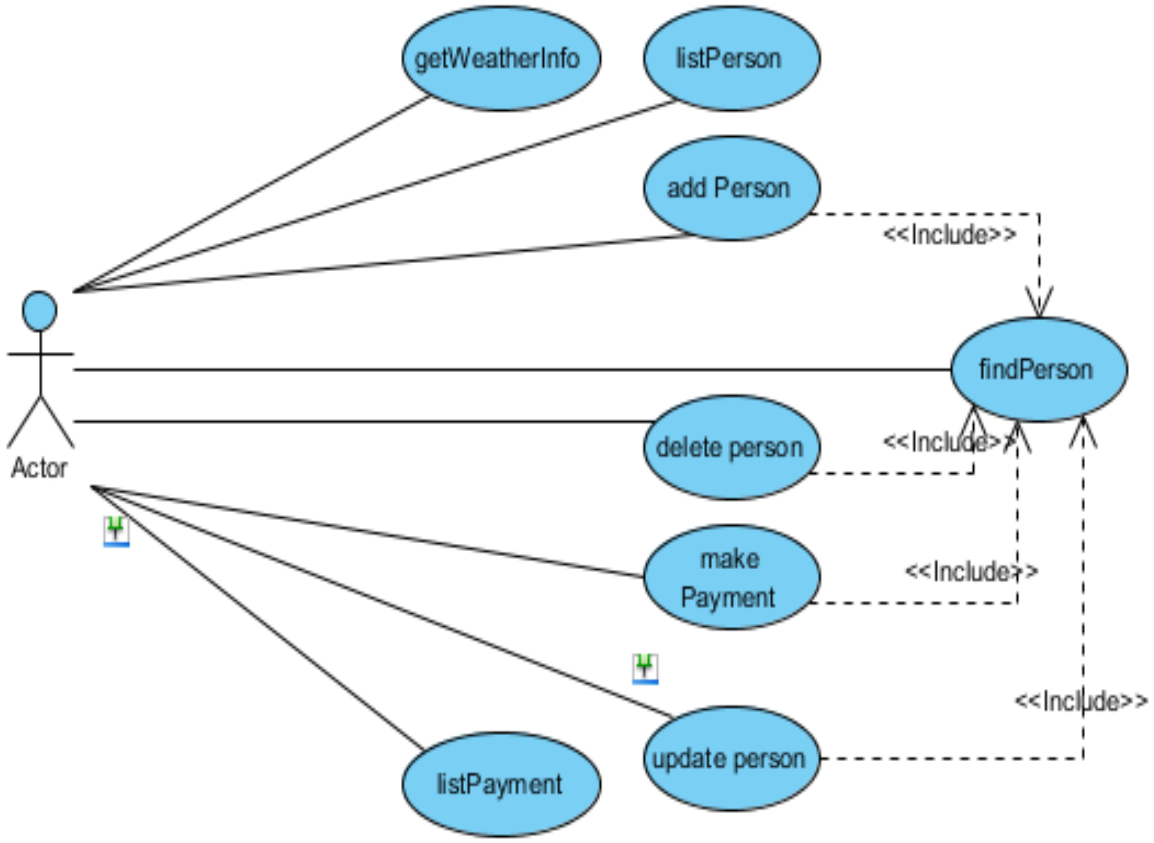
| | |
|---|---|
| **Java Sürümü:** | Java SE 1.6 |
| **GUI :** | Java Swing |
| **Depolama Bağlantıları:** | JDBC, JPA |
| **Sınıf Sayısı / Toplam Satır:** | 15 / ~4000 |
| **Paket Sayısı:** | 3 |
| **Paket Adları:** | cosmicer, cosmicer.gui, cosmicer.logic |
| **Geliştirme Ortamı:** | Eclipse 4.3 (Arayüzler NetBeans’ten alınmıştır.) |

Hedef uygulama soyut görünümü;



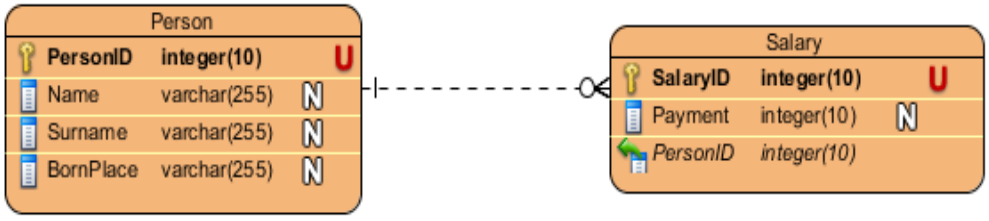

Şekil 27 Personel ve Ödemeler veritabanı soyut gösterimi

Hedef uygulama Java iş uygulamalarının geneli gibi CRUDL operasyonları ağırlıklı bir yazılımdır ve Şekil 28'deki diyagramda belirtilen kullanım durumlarını gerçekleştirmektedir. Tanımlanmış yedi kullanım durumuyla hizmet sunan uygulama, kullanıcı etkileşimli olmak üzere görsel arayüz üzerinden kontrol edilmektedir.



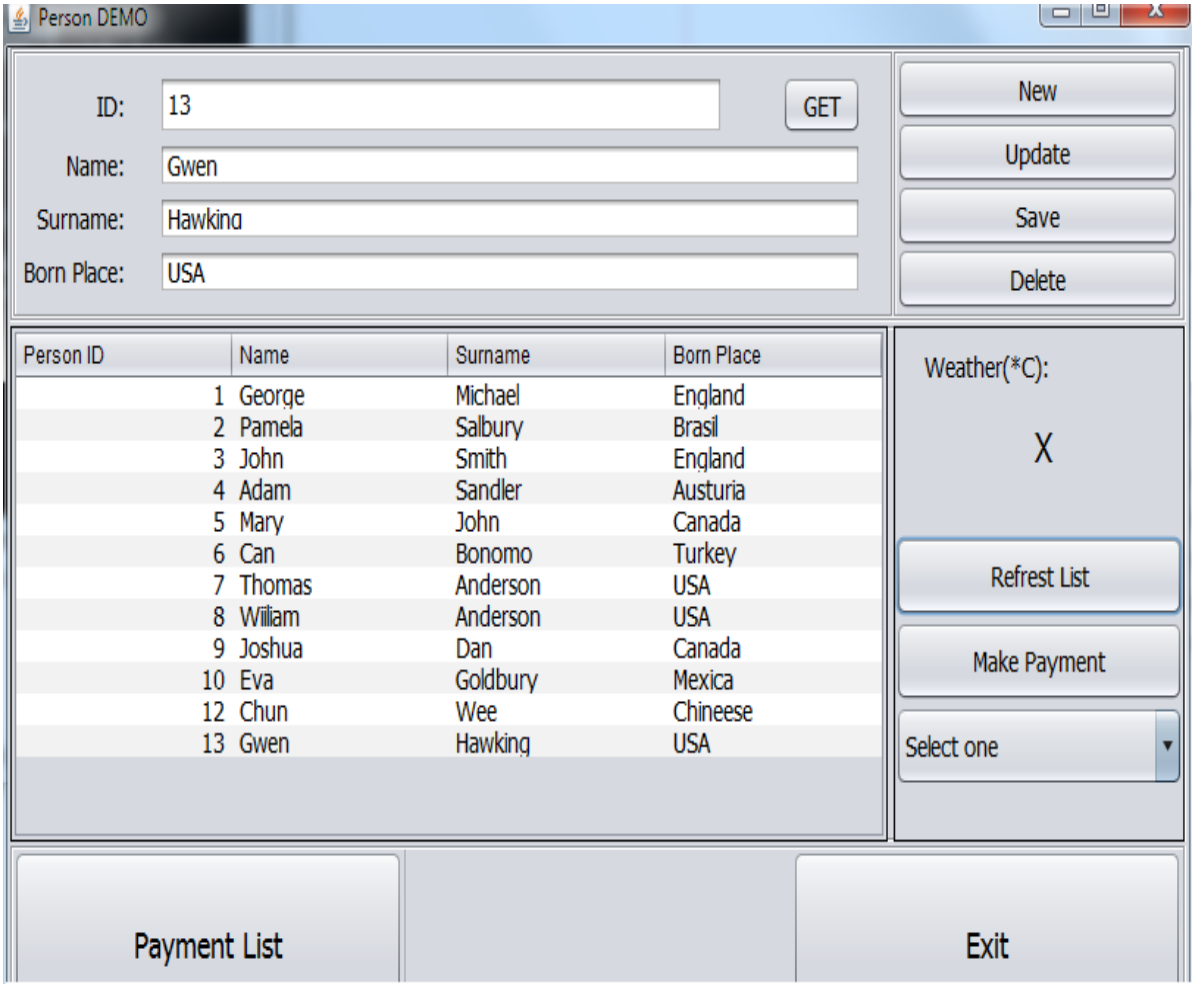

Şekil 28 Personel ve Ödeme Veritabanı uygulaması kullanım durumları

Uygulama kişi ve maaş adlı iki varlıktan (entity) oluşmaktadır. Kişi varlığı, personel bilgilerini ve maaş varlığı ise personel ödeme bilgilerini saklamaktadır. Şekil 29 ‘da varlık öznitelikleri ve varlık-bağıntı diyagramına yer verilmiştir.




Şekil 29 Personel ve Ödemeler Veritabanı Uygulaması Varlık-Bağıntı Diyagramı

Uygulamanın görsel arayüzü Java Swing Kütüphanesi kullanarak geliştirilmiştir ve arayüz Şekil 30 'da gösterildiği gibidir.




Şekil 30 Personel ve Ödemeler Veritabanı Uygulaması Grafiksel Arayüzü

Uygulamanın gerçekleştirdiği kullanım durumu senaryoları Tablo IX ’de yer almaktadır.

Tablo IX Hedef Uygulama Kullanım Durumu Senaryoları

| Personel Ekle (Yeni-Kaydet) : | |
|---|---|
| • Kullanıcı personel bilgilerini girer (id, name, surname and birthplace) ve kaydet (save) butonunu tıklar.<br>• Sistem personel id si üzerinden veritabanını kontrol eder.<br>• (Aynı id ye sahip kayıt bulunmazsa) yeni bir personel oluşturularak sisteme kaydedilir<br>• Bilgi mesajı ile kullanıcı bilgilendirilir. | |
| Personel Ara (Getir) : | |
| • Kullanıcı personel id bilgisini girer ve getir (get) butonunu tıklar.<br>• (Girilen id sistemde yer alıyorsa) personel bilgisi sistemden getirilir.<br>• Sistem personel bilgilerini kullanıcıya gösterir | |
| Personel Bilgisi Güncelle (Güncelle) : | |
| • Kullanıcı daha önceden getirilmiş personel bilgisi üzerinde düzeltme yapar ve güncelle (update) butonunu tıklar.<br>• Sistem personel bilgilerini veritabanından kontrol eder.<br>• Personel id sinin değiştiriliebilmesi durumu nedeniyle sistem ilgili diğer nesneleri kontrol eder.<br>• Sistem değiştirilmiş bilgiyi veritabanına kaydeder.<br>• Bilgi mesajı ile kullanıcı bilgilendirilir. | |
| Personel Sil(Sil) : | |
| • Kullanıcı personel id bilgisini girer ve sil (delete) butonunu tıklar.<br>• Sistem personel id sini veritabanından kontrol eder.<br>• Sistem personeli veritabanından siler.<br>• Bilgi mesajı ile kullanıcı bilgilendirilir. | |
| Personelleri Listele : | |
| • Kullanıcı yenile (refresh) butonunu tıklar.<br>• Sistem tüm personel bilgisini veritabanından okur.<br>• Sistem personel listesini görüntüler. | |
| Personel Ödemesi Gerçekleştir (Ödeme Kaydet) : | |
| • Kullanıcı personel id bilgisini girer.<br>• Kullanıcı personel ödeme miktarını girer.<br>• Sistem kullanıcı id sini veritabanından doğrular.<br>• Sistem ödeme bilgisini veritabanına kaydeder.<br>• Bilgi mesajı ile kullanıcı bilgilendirilir. | |
| Ödeme Listesini Görüntüle (Ödemeleri Getir) : | |
| • Kullanıcı ödeme listesini getir butonunu tıklar<br>• Sistem personel listesini veritabanından okur.<br>• Sistem ödeme listesini sistemden okuyarak personelle eşleştirir.<br>• Sistem personel ve ödeme listesini kısmen görüntüler | |
| Hava durumu bilgisini getir (Listeden bir şehir seçilir) : | |
| • Kullanıcı şehirler listesinden bir şehir seçer.<br>• Sistem şehir posta kodunu uzaktan çağrı yöntemiyle diğer sisteme gönderir.<br>• Sistem hava durumu bilgisini alır.<br>• Sistem hava durumu bilgilerini kullanıcıya gösterir | |

Kullanıcı senaryolarına karşılık gelen işlevsel süreçler Tablo X gösterilmektedir. Geliştirilen ölçüm aracıyla elde edilen sonuçlara karşılık uzman tarafından elde edilmiş sonuçların farklılık oluşturan veri hareketlerini içeren işlevsel süreçler Tablo XI da yer almaktadır.

Tablo X Cosmic Calculator İşlevsel Puan Sonuçları

| **İşlevsel Süreç** | **E** | **R** | **W** | **X** | **Toplam** |
|---|---|---|---|---|---|
| Personel Ekle | 1 | 1 | 1 | 1 | 4 |
| Personel Ara | 1 | 1 | - | 1 | 3 |
| Personel Bilgisi Güncelle | 1 | 2 | 1 | 1 | 5 |
| Personel Sil | 1 | 1 | 1 | 1 | 4 |
| Personelleri Listele | 1 | 1 | - | 1 | 3 |
| Personel Ödemesi Yap | **1** | 2 | 1 | 1 | 5 |
| Ödemeleri Listele | 1 | 2 | 1 | 1 | 5 |
| Hava durumu bilgisini getir | 1 | - | - | 1 | 2 |
| **Toplam**: | 8 | 10 | 5 | 8 | 31 |

Tablo XI Uzman Tarafından Elde Edilen Sonuçların Farkı

| **İşlevsel Süreç** | **E** | **R** | **W** | **X** | **Total** |
|---|---|---|---|---|---|
| Personel Ödemesi Yap | **2** | 2 | 1 | 1 | 6 |
| **Toplam** : | 9 | 10 | 5 | 8 | 32 |

Tablolarda yer alan sonuçlar incelendiğinde “CosmicSolver” tarafından elde edilen sonuçların uzman tarafından hesaplanan sonuçlara oldukça yakın olduğu görülmektedir. CosmicSolver bir tek işlevsel süreç hariç diğer işlevsel süreç veri hareketlerini uzmanla benzer şekilde hesaplamaktadır. Farklılık oluşturan “Personel Ödemesi Yap“ işlevsel süreci incelendiğinde “Girdi” puanlarında bir farklılık oluştuğu görülmektedir. Ayrıntıya inildiğinde ise problemin JPA kütüphanesinin işleyiş biçiminden kaynaklandığı görülmektedir. Uzman “Personel Ödemesi Yap“ kullanım durumunda personel id’sinin “person” nesnesine ve miktar bilgisinin ise “salary” nesnesine ait olduğunu algılamakta ve iki farklı nesne için farklı bilgiler girildiğinden girdi sayısını 2 olarak hesaplamaktadır. CosmicSolver ise personel id’sinin “salary” nesnesinin bir özniteliği olduğunu düşündüğünden tek bir nesne üzerinde işlem yaptığına karar vermekte ve 1 puan olarak hesaplamaktadır. Farklılık bundan kaynaklanmaktadır.

Sonuç olarak uzmanın gerçekleştirdiği ölçümlere çok yakın olacak şekilde sonuçlar elde edildiği düşünüldüğünde önerilen yöntemin başarılı olduğu görülmektedir.

.

## 5.2. Durum Çalışması

Önerilen “Kod Üzerinden Otomatik Olarak COSMIC Büyüklük Hesaplama Yönteminin” kendi geliştirdiğimiz küçük bir prototip uygulama üzerinde çalışabilirliğini gösterdikten sonra, yöntemin üçüncü bir kişi-organizasyon tarafından geliştirilmiş daha geniş kapsamlı bir yazılım üzerinde test edilebileceği bir durum çalışmasının yapılmasına karar verilmiştir.

Bu durum çalışmasında cevap aranacak sorular;

1. Geliştirilen yöntem yazılımın işlevsel büyüklüğünün hesaplanmasında etkili midir?
2. Önerilen yöntem zaman maliyeti açısından etkin midir?

olarak belirlenmiştir.

### 5.2.1. Durum Çalışması Hazırlıkları

Durum çalışması için hazırlanan sorulara cevap bulabilmek için bir ön çalışma yapılması gerekmiştir. Bu ön çalışma önerilen yöntemin uygulanacağı, sonuçların elde edilerek değerlendirilebileceği uygulamaların araştırılması ve kriterlere uygunluğunun test edilerek onaylanması süreçlerini içerir. Bu çalışmalar neticesinde üçüncü parti tarafından geliştirilmiş bir Java iş uygulamasının belirlenen sorular çerçevesinde önerilen yöntemle ölçülmesi işlemi gerçekleştirilmiştir.

Tez çalışması konusundan gelen kriterlerin yanı sıra, önerilen yöntemin prototip olması nedeniyle ortaya çıkan kısıtların belirlenmesi ve bu kısıtlar çerçevesinde arama yapılması gerekmiştir.

Ölçümü yapılacak uygulamanın belirlenmesinde göz önünde bulundurulan kısıtlar;

- Uygulamanın Java programlama diliyle geliştirilmiş olması gereklidir.
- Uygulamanın Java SE 1.5 ve üzeri sürümle geliştirilmiş olması gereklidir.
- Uygulama ara yüzünün Java Swing ile geliştirilmiş olması gereklidir.
- Uygulamanın iki ya da üç katmanlı mimari ile geliştirilmiş olması gereklidir.
- Uygulamanın veritabanı işlemleri ağırlıklı bir yapıda olması gereklidir.

- Uygulama, kullanılması durumunda JAX-RPC, JPA, JDBC standartlarına uygun geliştirilmiş kütüphaneler kullanıyor olmalıdır.
- COSMIC Manuel'de [1] yer alan "COSMIC Yazılım Bağlamı Model İlkeleri" başlığı altındaki şartları sağlayan bir uygulama olması gerekmektedir.
- COSMIC Manuel'de [1] yer alan "Jenerik Yazılım Modeli İlkeleri" ni sağlayan bir uygulama olmalıdır.

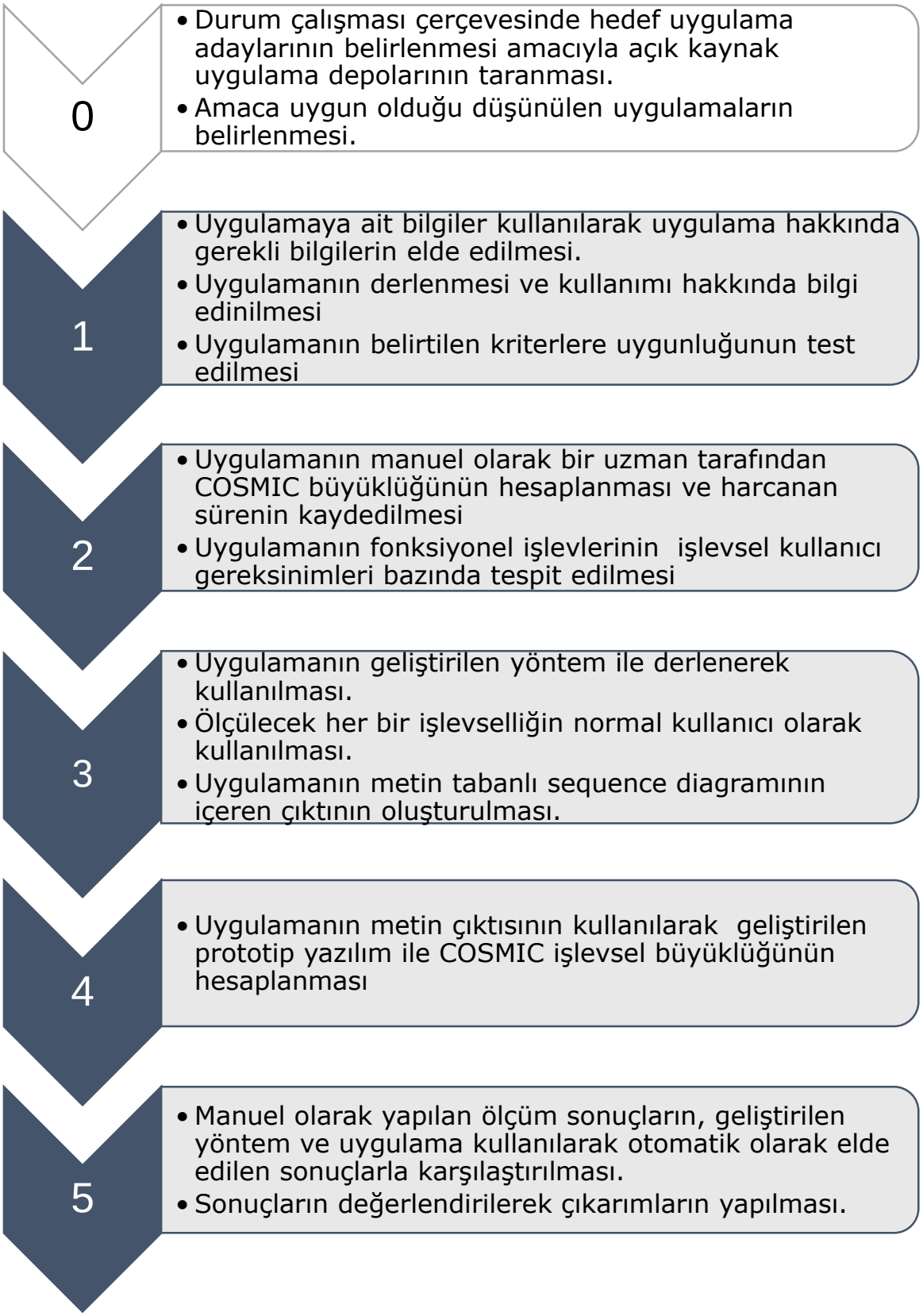
Uygulamanın üçüncü kişi ve kuruluşlar tarafından geliştirilmiş olması önerdiğimiz yöntemin uygulamaya özgü sonuçlar üretip üretmediğini test etmek açısından önemlidir ve sonuçların güvenilir olması için gereklidir. Bu yüzden belirtilen kriterlere uygun bir tarama gerçekleştirilmiştir.

Amacın gerçekleştirilmesi ve durum çalışması için hazırlanmış sorulara cevaplar için açık kaynak yazılımlara başvurulmuştur. Açık kaynak yazılımların sunulduğu, *GitHub*, *SourceForge*, *Google Code* gibi yazılım depoları taranarak aday yazılımlar tespit edilmiştir. Daha sonra aday yazılımlar kriterler doğrultusunda seçme işlemine tabi tutularak durum çalışmasının uygulanacağı yazılım belirlenmiştir.

Ölçüm yapılacak yazılım, veritabanı ağırlıklı işlemlerin gerçekleştirildiği Java programlama diliyle geliştirilmiş, Swing arayüz kütüphanelerini kullanan bir uygulamadır. *Uygulama hakkında detaylı bilgiler ilerleyen bölümlerde yer almaktadır.*

### 5.3. Durum Çalışması Eylem Planı

Durum çalışması yapılacak hedef uygulama belirlendikten sonra yapılacak işlemler önerilen yöntemin çalışabilirliğinin testi için geliştirilen prototip yazılımı temel alınarak belirlenmiştir. Eylem planına göre durum çalışmasında izlenecek adımlar Şekil 31'de gösterilmiştir.




Şekil 31 Durum Çalışmasında İzlenen Adımlar

## 5.4. Durum Çalışması Gerçekleştirimi ve Sonuçları

### 5.4.1. Ön Çalışma: Araştırma

Eylem planında “0” no’lu adım olarak belirlenen ve aday uygulamaların belirlenmesi amacıyla *GitHub*, *SourceForge* ve *Google Code* yazılım depoları taranmıştır. Ölçüm kriterlerini sağlayabileceği düşünülen aday uygulamalar belirlenmiştir. Bu uygulamalardan kriterleri sağlayabileceği düşünülen *SourceForge* kaynaklı “Nyagua Aquarium Application” durum çalışması için hedef uygulama olarak belirlenerek eylem planı ön çalışması tamamlanmıştır.

Uygulama deposu üzerinden sunulan uygulamanın sadece kaynak kodu yer almakta olup, uygulamaya ait dokümanlara ulaşılamamaktadır. Bu durum genel bir problem olup otomatikleştirme yöntemi önerisinin de faydalarından birini oluşturmaktadır. Dolayısıyla gerçek bir vaka üzerinden sadece kaynak kodu kullanarak büyüklüğün hesaplanması hedefini sağlayabilen bir durum çalışması olacaktır.

### 5.4.2. Birinci Adım: Bilgi Edinme

Eylem planının birinci adımında yer alan maddeler gereğince uygulama geliştiricisi sitesinden ve *SourceForge* uygulama deposundan uygulama ile ilgili bilgiler elde edilmiştir. Bu bilgiler Tablo XII ve Tablo XIII’de gösterilmektedir.

Tablo XII Durum Çalışması İçin Belirlenen Uygulamaya Ait Genel Bilgiler

| | |
|---|---|
| **Uygulama Adı:** | Nyagua Aquarium Management Application |
| **Uygulama Tanımı:** | Nyagua akvaryum sahiplerinin ya da yöneticilerin işlemlerini kolaylaştıran; genel olarak bir akvaryumda yer alan bitki, balık, omurgasızlar gibi canlıların takibinin yanı sıra, bakım, harcama ve çeşitli ölçümlerin takip edilebilmesi imkanlarını sunan bir yönetim destek uygulamasıdır. |
| **Platform:** | GNU/Linux olarak geliştirilen uygulama Java uygulama geliştirme dilinin sahip olduğu özellikler nedeniyle hemen her işletim sisteminde çalışabilecek şekilde adaptasyona sahiptir. |

| | |
|---|---|
| **Lisans:** | Uygulama GNU GPL v.2 lisansına sahiptir. Dolayısıyla uygulamanın bir garantisi olmayıp ücretsiz dağıtıma sahiptir. |

Tablo XIII Durum Çalışması İçin Belirlenen Uygulamaya Ait Teknik Bilgiler

| | |
|---|---|
| **Java Sürümü:** | Java SE 1.7 |
| **GUI:** | Java Swing |
| **Veritabanı:** | SQLITE |
| **JDBC / Persistence:** | Native JDBC |
| **JDBC URL:** | jdbc:sqlite://localhost/<file> |
| **Geliştirme Platformu:** | NetBeans |
| **Kullanılan Kütüphaneler:** | sqlite-jdbc-3.7<br>Jcalendar – 1.4<br>JDK 1.7 |

***Uygulama Kullanım Bilgileri***;

Uygulama kaynak kod olarak sunumu *NetBeans* projesi şeklindedir. Dolayısıyla ücretsiz olarak indirilip kullanılabilecek olan NetBeans geliştirme ortamına "import" edilerek uygulama derlenebilir. Uygulamayı derlemeden önce sistemde Java JDK 1.5+ geliştirme kütüphanesi yüklü olmalıdır. Ayrıca, uygulamanın *SourceForge* uygulama deposundaki tanıtım sayfasında yer alan indirmeler bölümünde halihazırda derlenmiş hali de sunulmaktadır. Basit bir kurulum sihirbazı eşliğinde sisteminize yüklenen uygulama, Java Sanal Makinesi aracılığı ile koşturulmaktadır.

Uygulamayı koşturmak için sihirbaz ya da *NetBeans* projesi derleminden başka bir şeye ihtiyaç yoktur. Bu anlamda kullanıcıya oldukça kolaylık sunmaktadır.

***Uygulamanın kriterleri sağlayıp sağlamadığının kontrolü;***

Uygulamanın belirtilen kriterler açısından değerlendirilebilmesi için kaynak koduna veya uygulama dokümanlarına ihtiyaç bulunmaktadır. Elimizde uygulamaya ait dokümanlar bulunmadığından kriter kontrolü kaynak kod üzerinden gerçekleştirilmiştir. Buna göre uygulamanın Java programlama diliyle gerçekleştirilmiş olması, Swing ara yüzünü kullanıyor olması, veritabanı ağırlıklı işlevsel gereksinimler sunması, iki katmanlı bir mimari sunması vb. gibi kriterleri sağladığı tespit edilmiş ve değerlendirmeye alınmıştır.

### 5.4.3. İkinci Adım: Manuel Hesaplama

Uygulamanın, kaynak kod kullanılarak manuel şekilde işlevsel büyüklüğü tespit edilmiştir. İşlevsel büyüklüğün yanı sıra işlevsel süreçlerin belirlenmesi ve kriterler kapsamında toplam işlevsel büyüklüğe eklenip eklenmeyeceğine karar verilmiştir.

Yapılan çalışmada elde edilen bilgiler Tablo XIV' da yer almaktadır.

Tablo XIV Durum Çalışması İçin Belirlenen Uygulamaya ait Manuel İşlevsel Büyüklük Ölçüm Bilgileri

| | |
|---|---|
| **Uygulamada yer alan işlevsel kullanıcı gereksinimi sayısı:** | 45 |
| **Kriterler kapsamında değerlendirmeye alınan ve manuel olarak büyüklükleri tespit edilen işlevsel kullanıcı gereksinimi sayısı:** | 39 |
| **Değerlendirmeye alınmış işlevsel kullanıcı gereksinimlerine karşılık gelen işlevsel süreçlerin büyüklüğü:** | 710 |
| **Toplam İşlevsel büyüklüğün hesaplanmasında harcanan süre:** | ~6 saat |

* Değerlendirmeye alınmayan 6 işlevsel kullanıcı gereksinimi (Bkz: Tablo XVII), daha önce belirtilen kriterleri (Bkz: Sf. 47) sağlamadığı için değerlendirmeye alınmamıştır. Değerlendirmeye alınabilmesi için prototipte henüz devreye alınmamış dosya operasyonları modülünün tamamlanmış olması gerekmektedir.

### 5.4.4. Üçüncü Adım : Otomatik Hesaplama

Uygulamanın önerilen yöntem ve geliştirilen prototip kullanarak ölçülmesi işleminde izlenen adımlar;

- Hedef uygulama kütüphaneleri ile birlikte Eclipse 4.5 AJDT geliştirme yazılımına *import* edildi.
- Yöntem çerçevesinde geliştirilen kesme noktaları (Pointcut) paketi uygulamaya dahil edildi ve proje AspectJ uygulamasına dönüştürüldü.
- Uygulama çalıştırma yöntemi olarak yükleme zamanı dokuma (LTW-Load-time Weaving) modu seçildi ve kesme noktaları (Pointcuts) dizini çalıştırma konfigürasyonuna eklendi.
- Uygulama çalıştırılarak ölçülmesi hedeflenen tüm süreçler bir işlevsel kullanıcı modunda koşturuldu.
- AspectJ LTW modunda çalıştırılan uygulama kesme noktaları vesilesiyle uygulamanın metin tabanlı UML dizge diyagramını üretildi. (Uygulama dizininde sequence.output)

Örnek “sequence.output” çıktısı:

```
Start:SWING>void util_panels.SolutionsPanel.6.actionPerformed(ActionEvent)
 Start:>void util_panels.SolutionsPanel.access$8(SolutionsPanel, ActionEvent)
  Start:>void  util_panels.SolutionsPanel.methodComboBoxActionPerformed(ActionEvent)
    Start:>void util_panels.SolutionsPanel.setTarget()
    End:>void util_panels.SolutionsPanel.setTarget()
    Start:>void util_panels.SolutionsPanel.switchMethodFields(boolean)
    End:>void util_panels.SolutionsPanel.switchMethodFields(boolean)
    Start:>void util_panels.SolutionsPanel.checkFields()
      Start:>boolean nyagua.Util.CheckTestFields(JTextField[])
      End:>boolean nyagua.Util.CheckTestFields(JTextField[])
    End:>void util_panels.SolutionsPanel.checkFields()
  End:>void util_panels.SolutionsPanel.methodComboBoxActionPerformed(ActionEvent)
 End:>void util_panels.SolutionsPanel.access$8(SolutionsPanel, ActionEvent)
End:SWING>void util_panels.SolutionsPanel.6.actionPerformed(ActionEvent)
```

- Prototip olarak geliştirilen *CosmicSolver* uygulaması çalıştırılarak üretilen çıktının işlenmesi ve hesaplama işlemlerinin gerçekleştirilmesi sağlandı.

Geliştirilen *CosmicSolver* ile hedef uygulama için gerçekleştirilen hesaplamaya ait sonuçlar Tablo XV' de yer almaktadır.

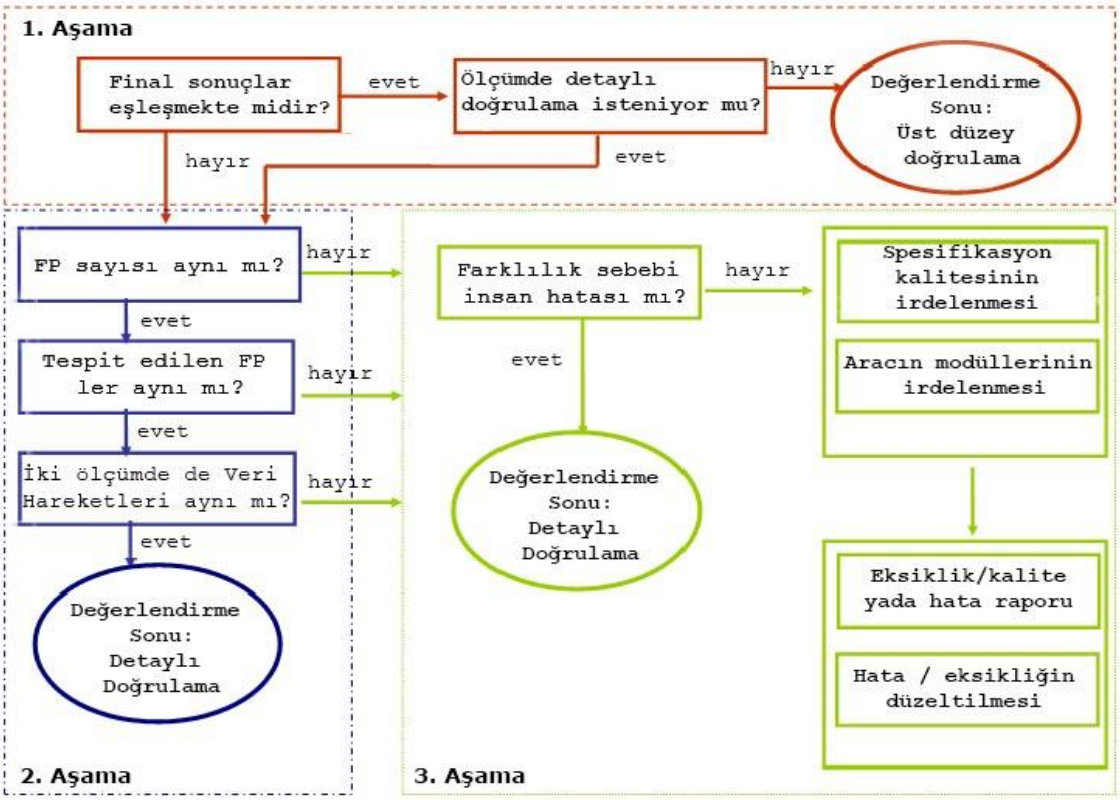
Tablo XV Durum Çalışması İçin Belirlenen Uygulamanın *CosmicSolver* ile Elde Edilen İşlevsel Büyüklük Ölçüm Bilgileri

| | |
|---|---|
| **Uygulamanın yakaladığı aday işlevsel süreç sayısı:** | 41 |
| **Asil işlevsel süreç sayısı:** | 40 |
| **Değerlendirmeye alınmış işlevsel kullanıcı gereksinimlerine karşılık gelen işlevsel süreçlerin büyüklüğü:** | 719 |
| **Toplam İşlevsel büyüklüğün hesaplanmasında harcanan süre:** | 30 dk.* |

** Normal şartlar altında (bu süreye uygulamanın çalıştırılmasına ilişkin hazırlıklar dahildir.)*

### 5.4.5. Dördüncü Adım : Değerlendirme

Elde edilen sonuçların detaylı olarak karşılaştırılabilmesi ve sonuçların doğruluğunun teyidi için Soubra ve ark. [30] makalesinde önerilen yöntem izlenmiştir. Soubra ve ark. makalesinde otomatik işlevsel büyüklüğün hesaplanmasında elde edilen sonuçların doğrulaması için üç aşamalı bir yöntem önermektedir. Bu yöntemi kısaca bir akış şeklinde özetlemek gerekirse Şekil 32'de belirtilen adımlar izlenerek sonuçlar karşılaştırılmalıdır.




*Şekil 32 Doğrulamaya ilişkin aşamalar ve adımlar [30]*

1. Aşama: Üst Düzey Doğrulama

Önerilen yöntemde ilk olarak toplam büyüklüğün otomatik ve manuel ölçümlerden elde edilen sonuçlarını dikkate alarak bir yönlendirme yapılmaktadır. Toplam sonuç değerler birbirine eşit ise üst düzey doğrulamanın sağlanabildiği düşünülmekte ve işlem sonlandırılmaktadır.

Geliştirdiğimiz prototipin durum çalışması kapsamında yapılan ölçümde elde ettiği sonuçlar, değerlendirmeye aldığımız işlevsel kullanıcı gereksinimlerinin uzman tarafından elle yapılan ölçme işlemi sonuçlarıyla paralellik göstermektedir. Bu durum, üst düzey doğrulama aşamasında, değerlendirmede göz önünde bulundurulan toplam büyüklük değerinin önerilen yöntem kullanılarak elde edilen sonuçlara çok yakın olduğunu ve dolayısıyla başarılı olduğunu göstermektedir.

| **Elle Yapılan Ölçüm** | **Otomatik Olarak Yapılan Ölçüm** |
|---|---|
| Toplam COSMIC Puan: 710 | Toplam COSMIC Puan: 719 |

2. Aşama: Detaylı Doğrulama

Birinci aşama temel düzey olup genel anlamda bir bilgi verse de daha detaylı bilgiler uygulamanın değerlendirilmesinde etkin rol oynamaktadır. Soubra ve ark. bu aşamada sırasıyla işlevsel süreçlerin sayısının, tespit edilen işlevsel süreçlerin aynı olup olmamasının ve veri hareketlerinin sayılarının ayrı ayrı ele alınması gerektiğini belirtmektedir. Bu değerlendirme sonucunda önerilen yöntemin ve uygulamanın sağlamlık ve doğruluğu güvence edilmiş olacaktır.

Değerlendirmenin sağlıklı yapılabilmesi ve manuel ölçme işlemlerinde kolaylık sağlaması açısından değerlendirmeye alınan kullanım durumları (use cases) uzman tarafından belirlenmiştir. Buna göre uygulamada yer alan ve ölçümü gerçekleştirilecek kullanım durumları listesi Tablo XVI'de belirtilmektedir.

Tablo XVI Değerlendirmeye alınan Nygua Aquarium Management Uygulaması Kullanım Durumları

| **Kullanım Durumları** | **Object-of-Interest (İlgi Nesnesi)** |
|---|---|
| **Ekle, Sil, Güncelle, Getir** | Aquarium |
| **Ekle, Sil, Güncelle, Getir** | Maintenance |
| **Ekle, Sil, Güncelle, Getir** | FishBase |
| **Ekle, Sil, Güncelle, Getir** | InvertebrateBase |
| **Ekle, Sil, Güncelle, Getir** | PlantBase |
| **Ekle, Sil, Güncelle, Getir** | Fish |
| **Ekle, Sil, Güncelle, Getir** | Plant |
| **Ekle, Sil, Güncelle, Getir** | Invertebrate |
| **Ekle, Sil, Güncelle, Getir** | Expense |
| **Ekle, Sil, Güncelle, Getir** | Measurement |
| **Ekle, Sil, Güncelle, Getir** | Device |
| **Ekle, Sil, Güncelle, Getir, İptal, Sonlandır** | Schedule |

Uygulamada yer alan fakat prototipin kısıtları nedeniyle ele alınmayan işlevsel süreçlerin listesi ve ele alınmama sebepleri Tablo XVII'de verilmiştir:

Tablo XVII Değerlendirmeye alınmayan Nygua Aquarium Management Uygulaması Kullanım Durumları

| **Kullanıcı Gereksinimi** | **Değerlendirmeme Sebebi** |
|---|---|
| **Export Aquarium Data** | Prototip uygulama dosya okuma yazma işlemlerini ele alacak şekilde yapılandırılmadığı için ele alınmamıştır. |
| **Import Aquarium Data** | |
| **Backup Aquarium Data** | |
| **Convert metrics** | Prototipte ele alınmayan Swing arayüz bileşenleri nedeniyle dahil edilmemiştir. |
| **Nutrient calculator** | |
| **Calculators** | |

Önerilen yöntemle otomatik olarak prototip uygulama tarafından hesaplanan "Nyagua Aquarium Management" yazılımının işlevsel kullanıcı gereksinimleri bazında veri hareket detayını içeren COSMIC büyüklüğü Tablo XVIII 'de gösterilmektedir. Listede yer alan işlevsel süreçler Tablo XVI' de belirtilen kullanım durumlarına karşılık gelen süreçlerdir.

Tablo XVIII Nygua Aquarium Management yazılımının işlevsel kullanıcı gereksinimleri ve büyüklük puanları

| **No** | **İşlevsel Süreçler** | **Otomatik Büyüklük** | **Manuel Büyüklük** | **Fark** |
|---|---|---|---|---|
| **1** | mainTreeMouseClicked(getAqu.) | 121 | 121 | 0 |
| **2** | getAquariumDataFromTable | 5 | 0 | 5 |
| **3** | aquariumSave | 77 | 77 | 0 |
| **4** | aquariumDelete | 79 | 79 | 0 |
| **5** | fishbaseSave | 8 | 8 | 0 |

| | | | | |
|---|---|---|---|---|
| **6** | fishbaseTable | 5 | 5 | 0 |
| **7** | fishbaseDelete | 11 | 11 | 0 |
| **8** | invertebratebaseGet | 5 | 5 | 0 |
| **9** | invertebratebaseSave | 11 | 9 | 2 |
| **10** | invertebratebaseDelete | 11 | 9 | 2 |
| **11** | plantbaseGet | 5 | 5 | 0 |
| **12** | plantbaseSave | 8 | 8 | 0 |
| **13** | plantbaseDelete | 11 | 11 | 0 |
| **14** | yesterdayList | 9 | 9 | 0 |
| **15** | schedSave | 56 | 56 | 0 |
| **16** | schedDelete | 51 | 51 | 0 |
| **17** | schedDoneTask | 57 | 57 | 0 |
| **18** | cancelTaskButton | 57 | 57 | 0 |
| **19** | readingsTable | 3 | 3 | 0 |
| **20** | readingsSaveButton | 7 | 7 | 0 |
| **21** | readingsDeleteButton | 6 | 6 | 0 |
| **22** | maintGet | 5 | 5 | 0 |
| **23** | maintSave | 8 | 8 | 0 |
| **24** | maintDelete | 9 | 9 | 0 |
| **25** | expensesGet | 9 | 9 | 0 |
| **26** | expensesSave | 10 | 10 | 0 |
| **27** | expensesDelete | 11 | 11 | 0 |
| **28** | devicesGet | 3 | 3 | 0 |
| **29** | devicesSave | 6 | 6 | 0 |
| **30** | devicesDelete | 7 | 7 | 0 |

| | | | | |
|---|---|---|---|---|
| **31** | fishTableMouseClicked | 3 | 3 | 0 |
| **32** | fishSave | 6 | 6 | 0 |
| **33** | fishDelete | 7 | 7 | 0 |
| **34** | invertsGet | 3 | 3 | 0 |
| **35** | invertsSave | 6 | 6 | 0 |
| **37** | invertsDelete | 7 | 7 | 0 |
| **38** | plantsGet | 3 | 3 | 0 |
| **39** | plantsSaveButton | 6 | 6 | 0 |
| **40** | plantsDeleteButton | 7 | 7 | 0 |
| | **Toplam:** | **719** | **710** | **9** |

Sonuçlar incelendiğinde manuel ve geliştirilen araçla otomatik olarak elde edilen işlevsel kullanıcı gereksinimleri sayısında ve bazı işlevsel kullanıcı gereksinimlerinin büyüklük puanlarında farklar olduğu gözlemlenmiştir.

İşlevsel kullanıcı gereksinimleri sayısı elle yapılan ölçüme göre bir fazladır. Toplam puanın farkının bir bölümü buradan gelmektedir. Diğer farklılık ise veri hareketlerindedir. İki işlevsel süreçte meydana gelen farklılık “çıktı” veri hareketi tipindedir. Karşılaştırılan değerler iki boyuttadır; birincisi işlevsel süreç sayısı, diğeri ise toplam işlevsel büyüklüktür. Elle yapılan ölçüme göre toplam işlevsel büyüklük doğruluk oranı yaklaşık %98’dir. İşlevsel süreç sayısı açısından yapılan karşılaştırmaya göre ise bu oran %97’dir.

3. Aşama : Düzeltme & Geliştirme

Aşamalar arasında bağıntı döngüsel olup aşamalarda verilen cevaplara göre eylemler dallanmaktadır. Üçüncü aşama olarak tanımlanan fazda düzeltmeler, iyileştirmeler ve eklentiler ile doğruluk oranının artırılması hedeflenmektedir. Programlama dillerinin esnek yapısı ve sürekli gelişmesi nedeniyle olası kapsam dışı eylemlerin tespiti ve gerekli düzeltmeler ve eklemeler yapılması bir zorunluluk olarak ortaya çıkmaktadır.

Durum çalışmasında hedef uygulama prototip kısıtlarına uygun olarak seçildiği için prototipte geliştirme ve genişletme işlemlerine ihtiyaç duyulmamakla birlikte, geliştirici kaynaklı birkaç hatanın düzeltilmesi gereği ortaya çıkmıştır. Prototipte gerçekleştirilen hata düzeltmeleri sonrasında tekrarlanan ölçme işlemleriyle hedefe ulaşılmıştır. Her yeni uygulama ve geliştirme sayesinde doğal evrimleşme süreci yaşayan prototipin sağlam bir temele oturması, hataların azalması ve genişletme maliyetinin sıfıra yakınsaması mümkün olacaktır.

### 5.4.6. Beşinci Adım : Çıkarımlar

COSMIC büyüklüğün otomatik olarak kaynak kod üzerinden hesaplanabilmesi için önerilen yöntemin işe yarar olduğu çalışmaların sonuçlarından görülmektedir. COSMIC yöntemin yapısından kaynaklı bazı durumlar nedeniyle oluşan çelişkilerden dolayı ortaya çıkabilecek farklılıklar bir kenara bırakılacak olursa oldukça yüksek bir oranda doğru hesaplama yapılabilmesi, prototipin geliştirilerek kullanıma sunulabileceğini göstermektedir. Diğer yandan sonuçları ve farklılıkları daha iyi anlayabilmek adına bir kritik yapılacak olursa; farklılıkların kaynağının iki noktada ortaya çıktığı görülebilir. Bunlar;

1. Olası işlevsel süreç sayısı farklılığı
2. Veri hareketleri sayısı farklılığı
    a. İşlevsel süreç için Girdi (Entry) sayısı farklılığı
    b. İşlevsel süreç için Çıktı (Exit) sayısı farklılığı

Önerilen yöntemde ortaya çıkan işlevsel süreçlerin sayısının farklılığı önerilen yöntemin ölçme biçiminden (eylem tabanlı) kaynaklanan ve Sf. 43'de belirtilen sorunlardan kaynaklanmaktadır.

Veri hareketlerinin sayısında ortaya çıkan farklılık ise girdi ve çıktı tipindeki hareketlerde görülebilmektedir. Girdi ve çıktıların tespiti otomatik ölçme işleminde en zor adımı oluşturmaktadır. Girdi ve çıktıların çok farklı şekillerde meydana gelebilecek olması işi zorlaştıran nedendir. Önerilen yöntemde girdi ve çıktıların kontrol altında tutulabilmesi için prototipin kapsamı belirlenmiş ve çalışma için Java Swing ara yüzü ile limit koyularak çalışma yapılmıştır. Yine de durum çalışmasında bazı işlevsel süreçlerde (9 ve 10 nolu süreçler için) farklı girdi/çıktı sayısına rastlanılmıştır. Bu durumun sebepleri araştırıldığında karşımıza iki neden

ortaya çıkmaktadır. Birincisi, prototipteki geliştirici kaynaklı hatalar (bug), diğeri ise Swing grafiksel ara yüz bileşenlerinin tümünün prototip kapsamında değerlendirmeye alınmamış olmasıdır.

Ölçme zamanı yönünden değerlendirilecek olursa, önerilen yöntemin oldukça başarılı olduğu ve yaklaşık 1/10 oranında bir kazanç sağladığı görülmüştür. Yazılım olgunluğa eriştikçe ve kapsamı genişledikçe Soubra ve ark.'ın [30] önerdiği doğrulama sürecinde ihtiyaç duyulan döngülerin oluşturacağı zaman kayıpları önlenecek ve ölçme süresi tüm ölçümler için standart hale gelecektir.

Sonuç olarak, prototipin doğası nedeniyle var olan eksiklikler ve geliştirici kaynaklı hataları ayrı tutarsak elde edilen sonuçlar oldukça yüksek doğruluk oranına sahip olmakla birlikte ölçme için ayrılan zaman önerilen yöntemde minimumdur. Yapılan durum çalışması yöntemin başarısını teyit etmekte ve hedeflenen sonuca ulaşıldığını göstermektedir.

## 5.5. Durum çalışmasının geçerliliğini etkileyebilecek faktörler

Bu bölümde, deneysel araştırmaların kalitesini ortaya koymak için kullanılan dört test yöntemi kapsamında durum çalışmasının geçerliliği irdelenmektedir. Yin [37] tarafından özetlenen bu testler; yapısal geçerlilik, iş geçerlilik, dış geçerlilik ve güvenilirliktir.

### Yapısal Geçerlilik

Yapısal geçerlilik çalışılan konsept için doğru metriklerin belirlenmesiyle ilgilidir [37].

Önerilen yöntem ölçüm eylemlerinin otomatikleştirilmesini hedeflemektedir. Durum çalışmasının hedefi ise önerilen yöntemin hesaplamadaki doğruluk oranının ve etkinliğinin test edilmesidir. Otomatikleştirme işlemiyle ölçüm eylemlerinde zaman kayıplarının önüne geçilebileceği iddiasıyla metriklerden biri zaman olarak belirlenmiştir. Diğer metrik ise geliştirilen aracın ölçüm işlemlerinde elle yapılan ölçümlere yakınsayan sonuçların elde edilebileceği iddiasıyla doğruluk oranı olarak belirlenmiştir. Metrikler önerilen yöntemin hedefleriyle birebir uyum içerisinde olduğundan durum çalışması için yapısal geçerliliğin sağlandığı söylenebilir.

Belirlenen metrikler manuel ve otomatik ölçümlerde kullanılarak karşılaştırma yapılmış ve oran hesaplaması gerçekleştirilmiştir.

Araç kullanılarak yapılan ölçümlerde elde edilen zaman değerinin belirlenmesinde, uygulamanın nasıl çalıştırılacağı, kullanım bilgileri ve ortamın hazırlanması gibi işlemlerin oluşturduğu maliyet göz ardı edilmiştir. Kullanılan kütüphanelerin incelenmesi, dizge diyagramlarının kullanıcı etkileşimleriyle oluşturulması, elde edilen dizge diyagramı çıktılarının genel hata kontrolü ve büyüklüğün hesaplanması ise araç yardımıyla gerçekleştirilen ölçme işleminin süresini belirlemektedir. Ölçülecek yazılımın büyüklüğü, dizge diyagramlarının kullanıcı etkileşimleriyle oluşturulması nedeniyle zaman değeri üzerinde doğrudan bir etki oluşturmaktadır.

Doğruluk oranı, iki açıdan incelenmektedir; birincisi manuel ve otomatik ölçmede elde edilen işlevsel kullanıcı gereksinimleri oranı, ikincisi ise toplam işlevsel büyüklük puanlarının oranıdır.

Otomatik ölçme eyleminde kullanıcı etkileşimiyle oluşturulan dizge diyagramları aday işlevsel süreçleri içermektedir. COSMIC Manuel [1]'de belirtilen işlevsel süreç niteliklerini sağlamayan, fakat kullanıcı tarafından tetiklenmiş eylemler, ilgili kurallar çerçevesinde ayıklanarak asil işlevsel süreçler ortaya çıkarılmaktadır. Asil olarak belirlenen süreçlerin büyüklük hesaplaması veri hareketlerinin sayılması ile gerçekleştirilmektedir. Tüm işlevsel süreçlerde gerçekleştirilen sayma işlemi sonrasında toplam büyüklük puanı elde edilmektedir.

Elle yapılan ölçümlerde ise, kullanıcı gereksinimleri dikkate alınmakta ve uygulamanın sunduğu işlevsellik, kod üzerinde yapılan çalışma ile elde edilmektedir. Belirlenen işlevsel süreçlerin büyüklük ise COSMIC Manuel [1] ve Java İş Uygulamaları Büyüklük Ölçümü Rehberi [25]'de yer alan kurallar çerçevesinde hesaplanmaktadır. Sayma işlemlerinin tüm aşamaları zaman maliyeti olarak değerlendirilmektedir.

**İç Geçerlilik**

İç geçerlilik, çalışmayla elde edilen sonuçlar üzerinde nedensel etki oluşturabilecek unsurlarla ilişkilidir [37]. İç geçerlilik, deneme sonucu olarak bağımlı değişkende meydana geldiği görülen gelişim, değişme ve farkı etkileyen faktörün deneysel değişken veya değişkenler olup olmadığı konusudur [37].

Durum çalışmasında seçilen uygulama iki katmanlı bir Java uygulamasıdır. Sunduğu işlevsellik açısından bakıldığında orta ölçekte bir büyüklüğe sahiptir. Özellikle veritabanı etkileşimleriyle hizmet sunmaktadır. Genel olarak Java kütüphanelerini kullanmakta olan yazılım üçüncü parti olarak *jcalendar* adlı takvim ve *sqlite* veritabanı sürücüsünden hizmet almaktadır. Kullanıcı etkileşimi ise Java Swing Kütüphanesi ile sağlanmaktadır. Taşıdığı özellikler itibariyle otomatik ölçüm eylemi için geliştirilen prototipin kriterlerini sağlaması iç geçerlilik oranını olumlu yönde etkilemektedir.

İç geçerliliğe tehdit olabilecek unsurlardan biri, ölçümü yapan ve otomatik ölçme aracını geliştiren kişinin sertifikalı bir uzman olmamasından kaynaklı olarak ortaya çıkan hata riskidir. Diğeri ise, uygulama seçimini yapan ve otomatik ölçme yöntemini geliştiren kişinin aynı olmasıdır.

**Dış Geçerlilik**

Dış geçerlilik araştırma sonucu elde edilen bulguların genellenip genellenemeyeceği durumudur. Bulunan, tanımlanan ve ölçülen sonuç, gelişme ya da farkın gerçekte bir anlamı olup olmaması, varsa bunun seviyesi ve diğer durumlara genelleyebilme olasılığı dış geçerliliğin derecesini gösterir [37].

Kriterlere (Bkz: Sf. 47) uyularak seçilen yazılımların yapısal geçerlilik başlığı altında belirtilen metrikler üzerinden elde edilen sonuçların değerlendirilmesiyle elde edilen veriye dayanılarak oransal bir genelleme yapılamaz. Gerçekleştirilen ölçümlerin kısıtlı sayıda olması dış geçerliliğin zayıf kalmasına sebebiyet vermektedir.

**Güvenilirlik**

Güvenirlik, aynı şeyin bağımsız ölçümleri arasındaki kararlılıktır; Ölçülmek istenen belli bir şeyin sürekli olarak aynı sembolleri almasıdır; Aynı süreçlerin izlenmesi ve aynı ölçütlerin kullanılması ile aynı sonuçların alınmasıdır; Ölçmenin, tesadüfi yanılgılardan bağımsız olmasıdır [37].

Aracın her ölçümde aynı sonuçlar verdiği, kendi içinde tekrarlanan ölçümlerde farklılıklar oluşmadığı saptanmıştır. Ayrıca sunulan materyal, bilgi ve kullanım kılavuzları aracılığıyla ölçümlerin üçüncü kişiler tarafından tekrarlanabilir kılınması güvenilirlik düzeyini artıran bir etkendir.

# 6. SONUÇLAR

Yazılım geliştirme projelerinde işlevsel büyüklük ölçümlerinin yadsınamaz konumu göze alındığında otomatikleştirme eyleminin önemi görülmektedir. Bu durum göz önüne alınarak önerilen çalışma Java iş uygulamaları için işlevsel büyüklük hesaplanması eyleminin otomatikleştirilmesi hedeflemektedir. Çalışmada önerilen yöntem hedefe iki adımda ulaşmaktadır; uygulamanın işlevsel süreçlerinin AspectJ yardımıyla metin formunda dizge diyagramlarının elde edilmesi ve UML dizge diyagramları üzerinden işlevsel büyüklük hesaplanması yönteminin referans alınarak ilgili COSMIC kurallarının işletilmesiyle büyüklüğün hesaplanması.

Önerilen çalışmada işlevsel süreçlerin dinamik analiz yöntemi kullanarak elde edilebilmesi için bağlam yönelimli programlama kullanılmaktadır. Bağlam yönelimli programlamanın Java dilinde gerçekleştirimi olan AspectJ'nin sunduğu olanaklar sayesinde, ölçümü yapılacak uygulama hakkında çok az bilgi olsa ve elde sadece derlenmiş kodu olsa dahi işlevsel büyüklük ölçümü için gerekli olan metin formunda dizge diyagramları elde edilebilmektedir.

Örnek bir uygulamanın yanı sıra bir durum çalışması eşliğinde yapılan çalışmalarda COSMIC işlevsel büyüklük, önerilen yöntem kapsamında geliştirilen araçla (CosmicSolver) otomatik şekilde ölçülmüştür. Daha sonra, elde edilen değerler, ölçüm uzmanının aynı uygulamalar üzerinde elle yaptığı hesaplamayla elde ettiği değerlerle karşılaştırılmıştır. Elde edilen sonuçlar göstermektedir ki önerilen yöntemle yapılan otomatik ölçüm ve manuel yöntem sonuçları birbirine oldukça yakınsamaktadır. Bu da UML dizge diyagramları büyüklük ölçümlerinin otomatikleştirilebilmesinin yanı sıra zaman, maliyet ve insan faktörlerinin, işlevsel büyüklük ölçümlerindeki etkisinin minimize edilerek tutarlı sonuçlar elde edilebilmesinin mümkün olduğu sonucuna ulaştırmaktadır.

Yöntemin genişletilebilir yapısı nedeniyle, hedef uygulama alanı genişletilerek Java Swing uygulamalarının yanı sıra, web uygulamaları, toplu işlem süreçleri gibi eylemleri içeren uygulamaların dahi ölçülebilme imkanını barındırması, geliştirilen prototip uygulamanın daha geniş çerçevede ele alınarak bir ürüne dönüştürülebileceğini göstermektedir.

# KAYNAKLAR


[1] COSMIC – Common Software Measurement International Consortium: The COSMIC Functional Size Measurement Method - version 3.0.1 Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003), **2009**

[2] A. J. Albrecht: *Function point analysis, Encyclopedia of Software Engineering*, 1. John Wiley & Sons, **1994**

[3] Akca, Ahmet Ata, and Ayca Tarhan. "Run-Time Measurement of COSMIC Functional Size for Java Business Applications: Is It Worth the Cost?" *(IWSM-MENSURA), 2013 Joint Conference of the 23rd Int. Workshop on*. IEEE, **2013**.

[4] Robiolo, G., "How Simple is It to Measure Software Size and Complexity for an IT Practitioner?," *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* , vol., no., pp.40,48, 22-23 doi: 10.1109/ESEM.2011.12, **2011**

[5] S. Kusumoto et al., “Function point measurement from Java programs,” proceedings of ICSE, **2002**

[6] Bévo, V., Lévesque, G., Abran, A.: Application de la méthode FFP à partir d'une spécification selon la notation UML In: Proc. 9th Int. Workshop Soft. Measurement. Lac Supérieur,Canada, 230-242, **1999**

[7] Jenner, M.S.: COSMIC-FFP 2.0 and UML: Estimation of the Size of a System Specified in UML - Problems of Granularity. In: Proc. 4th Eur. Conf. Soft. Measurement and ICT Control. Heidelberg, 173-184, **2001**

[8] B. A. Kitchenham: The problem with function points", IEEE Software, 14(2):, pp. 29-31, **1997**

[9] Fehlman, T. M., and Eberhard Kranich. "COSMIC Functional Sizing based on UML Sequence Diagrams." *MetriKon, Kaiserslautern*, **2011**

[10] K. Paton, “Automatic function point counting using statistics and dynamic code analysis,” Int. Workshop on Soft. Measurement, IWSM’99, **1999**

[11] Azzouz, S. and A. Abran, "A proposed measurement role in the Rational Unified Process (RUP) and its implementation with ISO 19761: COSMIC FFP," Software Measurement Euro. Forum Rome, Italy, **2004**.

[12] ISO. Information technology – Software measurement – Functional size measurement. Part 1: Definition of concepts. *International Standard ISO/IEC 14143-1*, **2007**

[13] Levesque, G., Bevo, V., & Cao, D. T. Estimating software size with UML models. *Proceedings of the 2008 C3S2E Conference on - C3S2E ’08*, 7. doi:10.1145/1370256.1370268, **2008**

[14] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J.-M., & Irwin, J. Aspect-Oriented Programming. *ACM Computing Surveys*, *28*, 220–242. doi:10.1145/242224.242420, **1997**

[15] Luigi Lavazza, and Vieri Del Bianco. IWSM/Mensura, volume 5891 of Lecture Notes in Computer Science, page 101-121. Springer, **2009**

[16] International Standards Organization and International Electrotechnical Commission, "COSMIC: a functional size measurement method," ISO/IEC 19761,**2011**

[17] International Standards Organization and International Electrotechnical Commission, "IFPUG functional size measurement method," ISO/IEC20926,**2009**

[18] Dean Wampler, Use Cases as Aspects – An Approach to Software Composition, AspectProgramming Inc

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

[20] 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**

[21] OMG Unified Modeling Language™ (OMG UML), Superstructure V2.2 pp. 506-524, http://www.omg.org/spec/UML/2.2/Superstructure /PDF/[seen 2-May-2011, **2009**

[22] Wang, Yi; Zhao, Jianjun (July 2007). "Specifying Pointcuts in AspectJ" (PDF). Proceedings of the 21st Annual International Computer Software and Applications Conference 2: 5–10. doi:10.1109/ COMPSAC.2007.196. ISBN 0-7695-2870-8, **2010**

[23] Durkin, J. 1994. Expert system: Design and development. New York: Prentice Hall. Eriksson, H.-E., and M. Penker. 1998. UML toolkit. Chichester, UK: John Wiley & Sons. Forselius, P, **2006**

[24] LogSequencer Application. To get more info, http://www.logsequencer.com/ , [seen 1-Ekim-2014]

[25] Abran, A. et. al.: The COSMIC Functional Size Measurement Method – Version 3.0 – Guideline for Sizing Business Application Software 1.1, http://www.cosmicon.com/ [seen 2-May-2014],**2008**

[26] A. J. Albrecht, "Measuring Application Development Productivity," Proceedings of the Joint SHARE, GUIDE, and IBM Application Dev. Symposium, Monterey, IBM Corporation, pp. 83–92, **1979**

[27] Fenton N. E. And Pfleeger S. L., Software Metrics “A Rigorious & Practical Approcach”, Second Edition, PWS Publishing Company, **1997**

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

[29] Fetcke, T., Abran, A., & Dumke, R.,“A Generalized Representation for Selected Functional Size Measurement Methods”, 11th International Workshop on Software Measurement, Montreal, Canada, **2001**

[30] H.Soubra, A.Abrain, Verifying the Accuracy of Automation Tools for the measurement of software with COSMIC – ISO 19761, **2014**

[31] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard: Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, **1992**

[32] Ramnivas Laddad. *AspectJ in Action: Enterprise AOP with Spring Applications* (2nd ed.). Manning Publications Co., Greenwich, CT, USA., **2009**

[33] Habela P., Glowacki E., Serafinski T., Adapting Use Marry Model for COSMIC-FFP based Measurement, in the 15th International Workshop on Software Measurement,Montreal, Canada,Shaker-Verlag, **2005**

[34] Vilus L, Levesque G., 2004, Une méthode efficace pour l’extraction des instances de concepts dans une spécification UML aux fins de mesure de la taille fonc-tionnelle de logiciels. ICSSEA, **2004**

[35] K.v.d. Berg, D. Ton, and O. Rogier, “Functional Size Measurement Applied to UML-base User Requirements”, Retrievable from doc.utwente.nl, **2005**

[36] Norman Fenton, Software Measurement: A Necessary Scientific Basis, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. VOL. 20, NO. 3, **1994**

[37] R.K. Yin, Case Study Research: Design and Methods, Applied Social Research Methods Series, vol. 5, SAGE Publications, **2003**.

# EK-1: “COSMIC CALCULATOR” Kullanım Kılavuzu

## “COSMIC CALCULATOR”

## Kullanım Kılavuzu

### Giriş

Yazılım kullanılarak COSMIC büyüklüğün ölçülebilmesi üç adımda gerçekleştirilmektedir. Adımlardan ikisi opsiyonel olarak görülebilir; bu adımlar için uygulamayı kullanmak zorunlu değildir. Manuel olarak yapılabilir. Fakat girdileri kullanarak hesaplamayı gerçekleştiren üçüncü adım-modüldür. Yazılımın adımlarını oluşturan modüller;

1. PointCuts Export (Tracer)
2. Compile & Run (Derle ve Çalıştır)
3. Cosmic Calculation (COSMIC Hesaplama)

Uygulamayı çalıştırabilmek için gereklilikler;

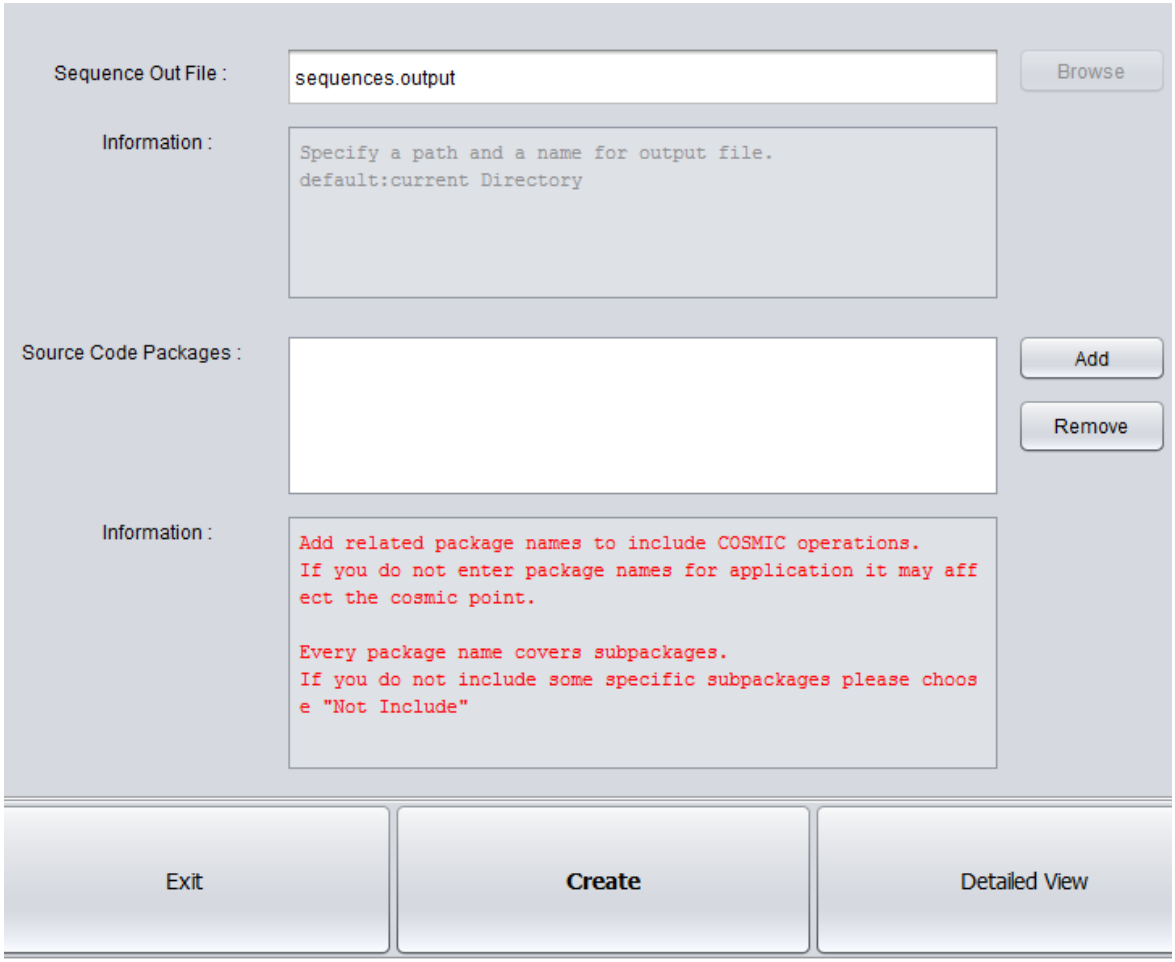
- Java SE 1.6+ yazılımı sistemde yüklü olmalı ve JAVA_HOME çevre değişkeni doğru şekilde set edilmiş olmalıdır.
- AspectJ 1.7+ sistemde kurulu olmalıdır.

Herhangi bir uygulamanın COSMIC işlevsel büyüklük puanını hesaplamak için üç adım gereklidir.

- “Pointcuts Export” Adımı
- “Compile & Run” Adımı
- “COSMIC Calculation” Adımı

### “Pointcuts Export” Adımı

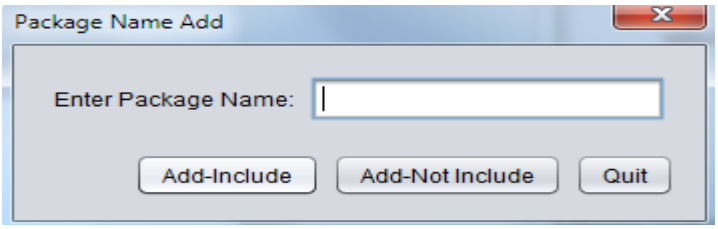
“Tracer” bileşenini oluşturan bu adım; hedef yazılımdan gerekli bilgileri alabilmek için gerekli tavsiye yordam ve kesme noktalarını içeren sınıfları oluşturmaktadır. Oldukça basit bir kullanımı olan bu bileşen aşağıda belirtilen uygulama paketlerinin adları ve çıktı dosyasının adı bilgilerinin girilmesi ile tamamlanmaktadır.




Şekil 33 "Pointcut Export (Tracer)" bileşeni ekran görüntüsü

**Sequence Out File(Belirtim: zorunlu):** Varsayılan değeri “sequence.output” olan bu değişken dizge çıktılarının yazdırılacağı dosya adının belirtilmesinde kullanılmaktadır. Eğer oluşturulacak dosyanın tam yolu belirtilmez ise çıktılar, uygulamanın çalıştırıldığı dizine belirtilen isimle oluşturulan dosya içine yazılacaktır.

**Source Code Packages (Belirtim: zorunlu):** Opsiyonel olmayan bu bilgi sistemin düzgün çalışabilmesi için gereklidir. Uygulamada yer alan ve sınıfların içinde yer aldığı ölçümü düşünülen tüm paket isimleri bu listeye girilmelidir. “Add” butonunu tıkladığımızda Şekil 34’de gösterilen “Package Name Add” adlı diyalog kutusu oluşturulmaktadır. Bu diyalog kutusu yardımıyla uygulamanın paket isimleri girilerek sisteme dahil edilecek ya da edilmeyecek paketleri belirtilmektedir.




Şekil 34 Paket Adı Ekle diyalog kutusu

*Add-Include:* Belirtilen paketi ekleyin.

*Add-Not Include:* Eğer uygulamada dizge diyagramları oluşturmak için izlemeye dahil etmek istemediğiniz paketler var ise sisteme belirtmek için kullanın.

*Quit:* Diyalog kutusunu kapat

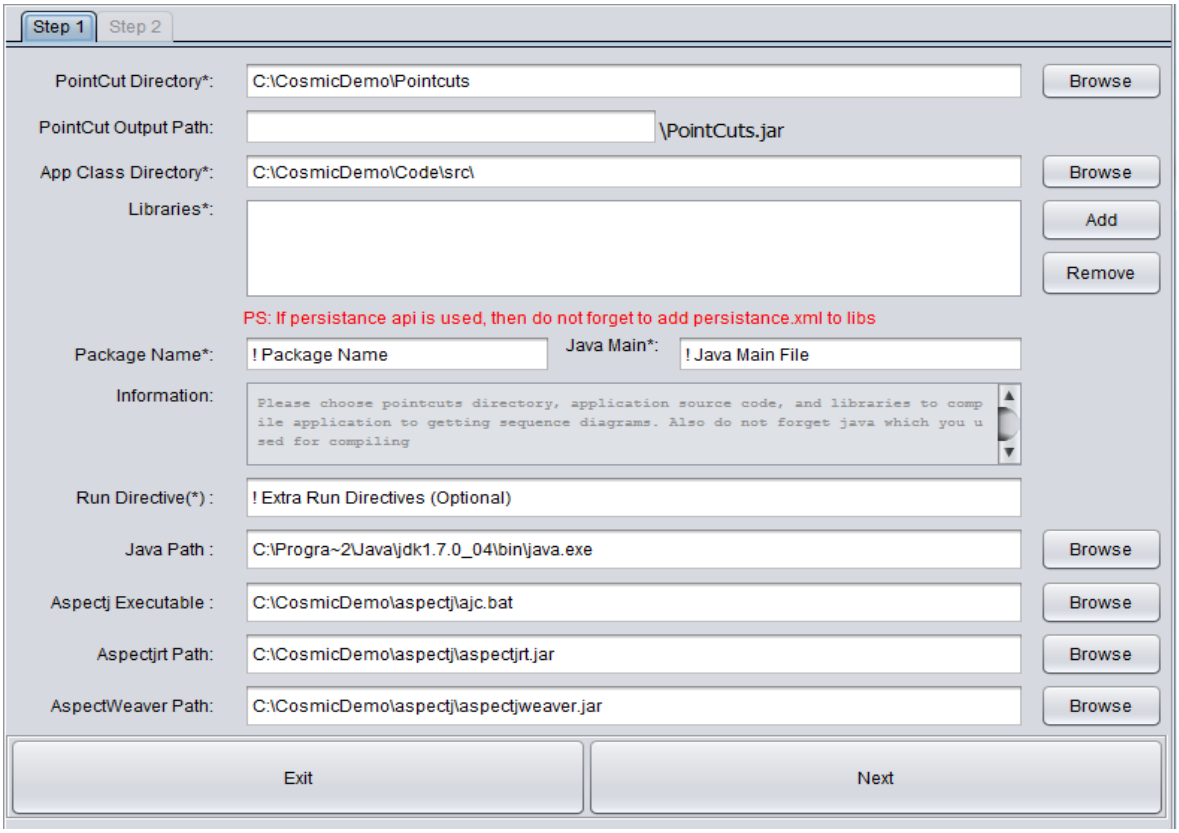
Belirtilen gerekli iki bilgi girildikten sonra “Create” butonu ile kesme noktaları (poincut) ve tavsiye yordamları (advice) içeren, “.aj” uzantısına sahip aspectj sınıfları “PointCuts” dizinine oluşturulacaktır. “PointCuts” dizini uygulamanın çalıştırıldığı yerde oluşturulacaktır.

**Not:** “Detailed View” işlevsel değildir.

**“Compile & Run” (Derle & Çalıştır) Adımı**

Derleme ve çalıştırma adımında yardımcı olarak geliştirilen bu bileşen kullanıcılara kolaylık sağlamaktan öte bir işlevi yoktur. Bu bileşeni kullanarak, kesme noktalarını derleyebilir ve uygulamayı kolayca çalıştırabilecek betikleri oluşturabilir. Bu eylemler tamamıyla el ile de gerçekleştirilebilmektedir. İki alt adım içeren bu bileşende birinci adımda gerekli bilgiler girilir ikinci adımda çıktılar oluşturulur.

**Adım 1:**




Şekil 35 Derle & Çalıştır bileşeni ekranı birinci adım görüntüsü

**Pointcut Directory**: “PointCut Export” bileşeniyle oluşturulan “.aj” uzantılı dosyaları içeren klasör

**Pointcut Output Path**: “PointCuts” dizininde yer alan aspectj sınıflarının derlendikten sonra “PointCuts.jar” dosyası olarak oluşturulacağı konum bilgisi.

**App Class Directory**: Hedef uygulama kod dizini konum bilgisi.

**App Libraries Directory**: Hedef uygulamanın ihtiyaç duyduğu kütüphaneler

**Package Name / Java Main**: “main” yordamını içeren sınıfın adı – uygulamayı çalıştıran java sınıfının içinde yer aldığı paket adı

**Run Directive**: Hedef uygulamayı çalıştırmak için varsa gerekli direktif

**Java Path**: Sistemdeki java uygulaması konum bilgisi

**AspectJ Executable**: AspectJ kurulduğunda oluşturulan dizinde yer alan bin\ajc.bat file dosya konum bilgisi.

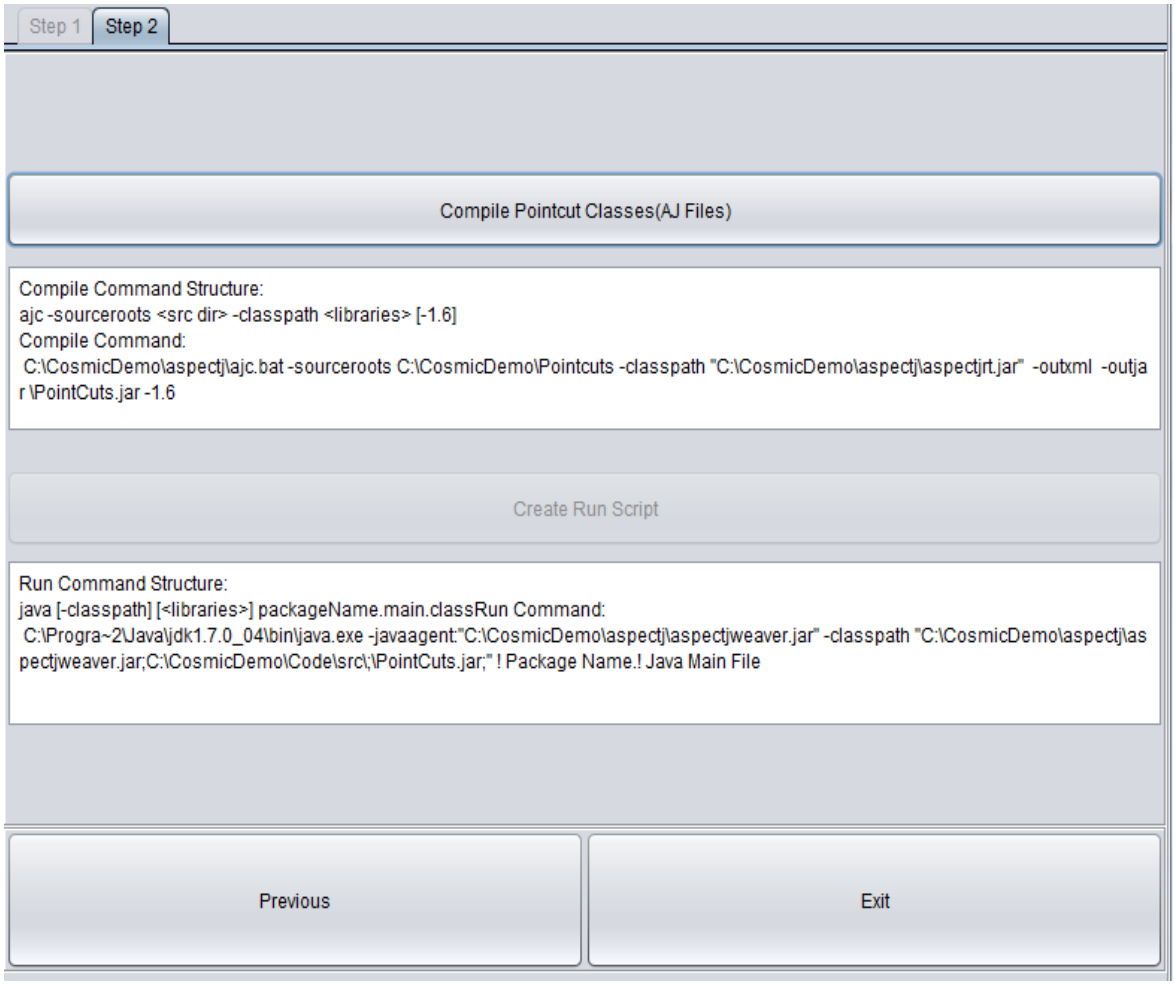
**Aspectjrt**: AspectJ dizini içinde yer alan kütüphanenin(aspectjrt.jar) konum bilgisi

**AspectjWeaver Path**: AspectJ dizini içinde yer alan Java ajanı ile uygulamayı başlatmak için gerekli kütüphanenin (aspectjweaver.jar) konum bilgisi

Tüm bilgiler girildikten sonra “next” butonu tıklanarak oluşturulan komutlara erişilir.

**Adım 2:**

Gerekli bilgiler girildiğinde oluşturulan yönlendirme amaçlı çıktılar Şekil 36’deki ekran görüntüsündeki gibidir. Compile(derle) butonu tıklanarak komut çalıştırılabileceği gibi ilgili betik kopyalanarak komut satırından da çalıştırılabilir.




Şekil 36 Derle & Çalıştır bileşeni ekranı ikinci adım görüntüsü

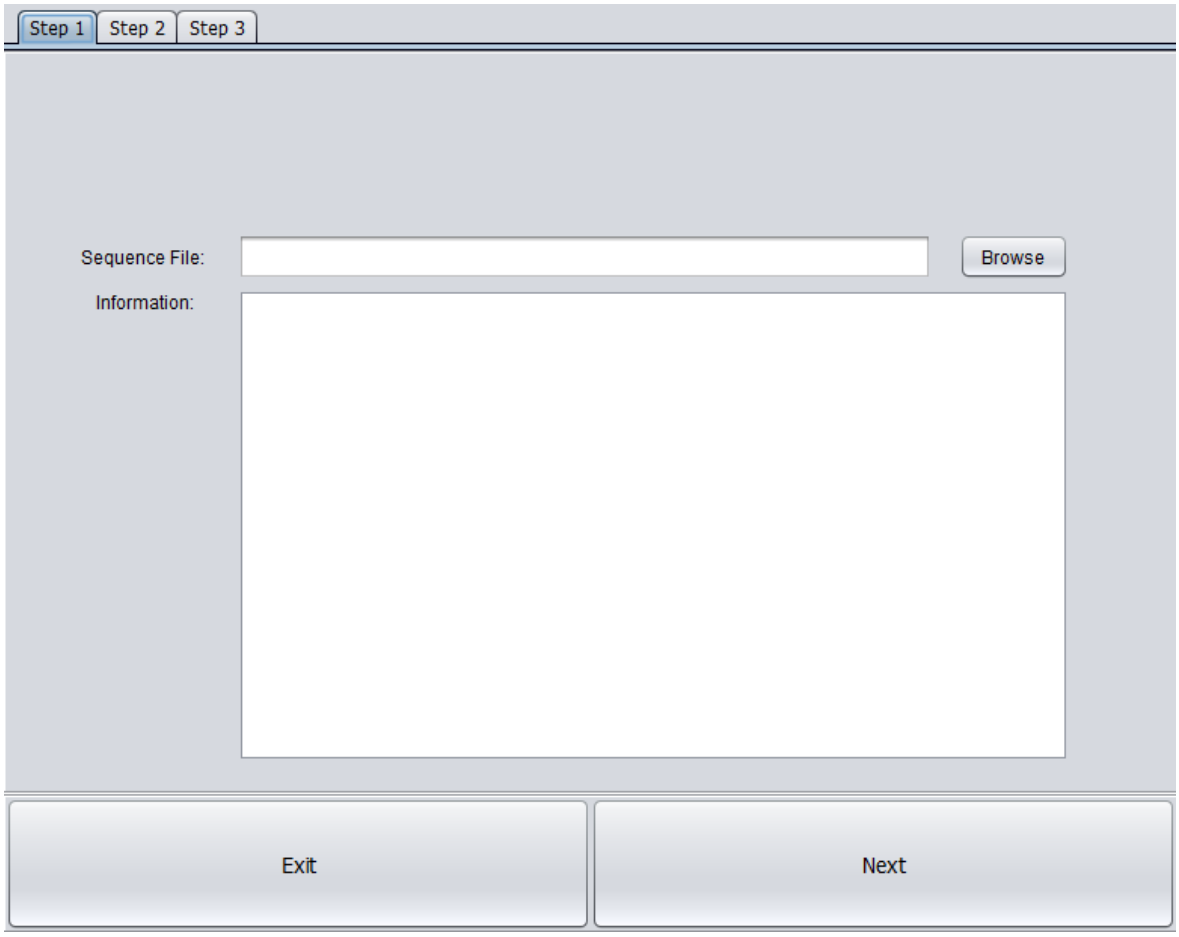
Java agent eşliğinde çalıştırılan hedef uygulama normal bir kullanıcı olarak kullanıldığında birinci modülde belirtilen dizge çıktıları dosyasına ölçüm için gerekli bilgileri yazacaktır. Üçüncü modülde bu dosya kullanılarak işlevsel büyüklük hesaplanabilecektir.

**“Cosmic Calculation” Adımı**

Hesaplama işlemi için metin formundaki dizge diyagramlarını içeren “sequence.output” dosyasına ihtiyaç vardır. İlk iki adımda bu dosyanın oluşturulabilmesi için gerekli eylemler belirtilmektedir.

**Adım 1:**

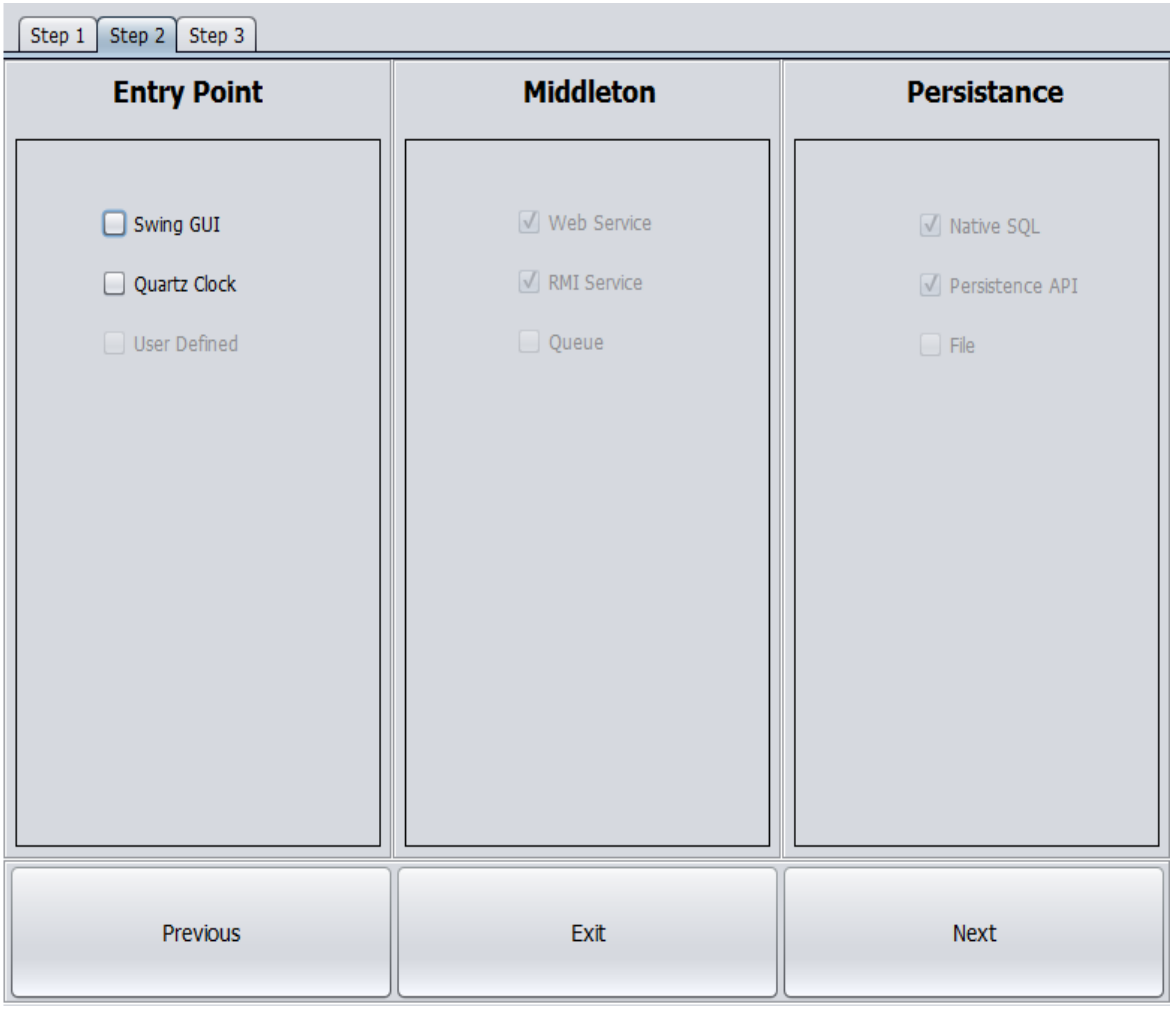
Hedef uygulamanın AspectJ ile dokunması ile çalıştırılmasının ardından, kullanım ile elde edilen metin formundaki dizge diyagramlarını içeren dosya konumu “Sequence File” olarak belirtilen alana girilir. Bu dosya sonraki adımlardaki bilgiler kullanılarak işlenecek ve çıktılar üretilecektir. “Browse” butonu kullanılarak dosya konumu belirtilebilir. Daha sonra “next” tuşu ile ikinci alt adıma geçilir.




Şekil 37 "Cosmic Calculator" birinci adım ekran görüntüsü

**Adım 2:**

İşlevsel kullanıcı gereksinimlerinin tetikleyicilerini belirleyebilmek için ön tanımlı uygulama öznitelikleri belirlenir. Prototip kapsamında grafiksel arayüzden tetiklenen işlevsel gereksinimlerin büyüklükleri ölçülebilmektedir. Orta katmanda, web ve RMI servisleri; depolama katmanında ise, JDBC ve JPA teknolojileri desteklenmektedir. “Entry Point” olarak “Swing GUI” seçilir ve “Next” butonu tıklanır.



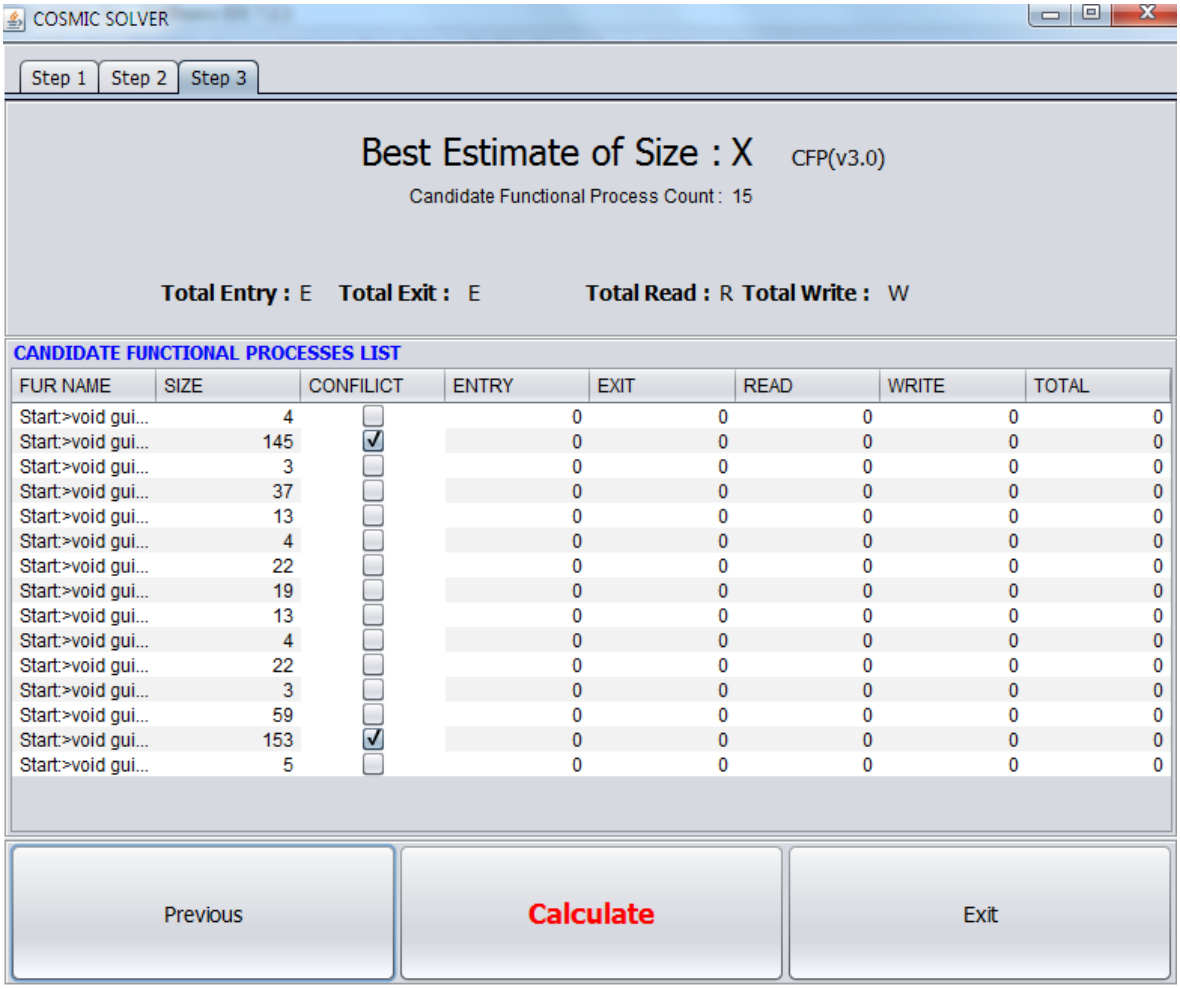

Şekil 38 "Cosmic Calculator" ikinci adım ekran görüntüsü

**Adım 3:**

Girdi olarak verilen dizge çıktıları dosyası işlenerek aday işlevsel süreçler listelenir.

Giriş noktaları aynı olan fakat dizge boyutu birbirinden farklı işlevsel süreçler "conflict" sütununda işaretli şekilde gösterilmektedir.

"Calculate" butonu tıklandığında COSMIC kuralları işletilerek işlevsel süreçlerin COSMIC puanları hesaplanır. Çeşitli ayrıntılarıyla ekranda görüntülenir. "En iyi kestirim" olacak şekilde sonuçlar ve işlevsel süreç sayısı kullanıcıya sunulur.




Şekil 39 "Cosmic Calculator" üçüncü adım ekran görüntüsü

## ÖRNEK ÖLÇÜM – DEMO UYGULAMA

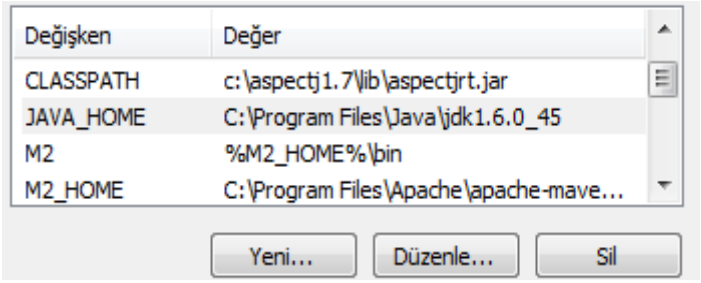
Makaleyi oluştururken kullanılan ve Örnek uygulamalar başlığı altında bahsedilen yazılımın ölçümünün nasıl gerçekleştirileceği adım adım anlatılacaktır.

Demo gösterim olarak kullanılacak yazılım Windows platformunu kullanmaktadır. Adımlar;

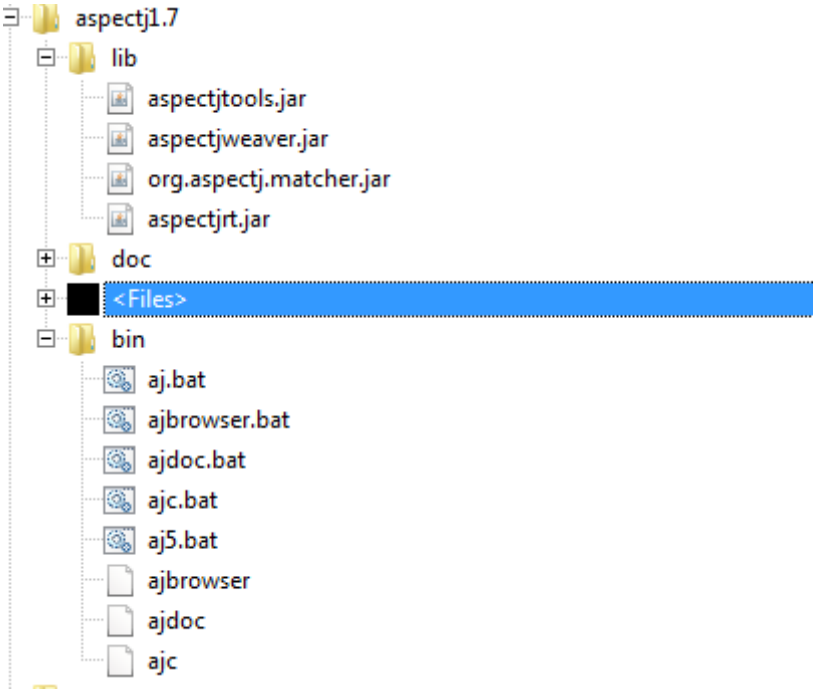
1. Java yazılımının kurulup olup olmadığını kontrol et. Komut satırında “java -version” komutu yazıldığında alttaki gibi bir çıktı elde edilecektir. Eğer 1.6+ sürümü yüklüyse çevre değişkenlerini kontrol et.

```
C:\Users\muhammets>java -version
java version "1.7.0_45"
Java(TM) SE Runtime Environment (build 1.7.0_45-b18)
Java HotSpot(TM) 64-Bit Server VM (build 24.45-b08, mixed mode)
```

   Computer Properties-> Advanced -> Environment variables




2. AspectJ paketlerini indir ve sisteme kur. Daha sonra “classpath” e ekle. Eğer varsayılan konfigürasyon ile kurulursa c:\aspectj1.7\ dizini olacaktır.



Bu demo için aspectj kurulumuyla gelen üç adet dosyaya ihtiyaç vardır. Bunlar;

- bin\ajc.bat
- bin\aspectjweaver.jar
- bin\arspectjrt.jar

3. Hedef uygulamamız alttaki klasör yapısına sahiptir. (ya da verilen uygulamayı APP01 dizine aç)

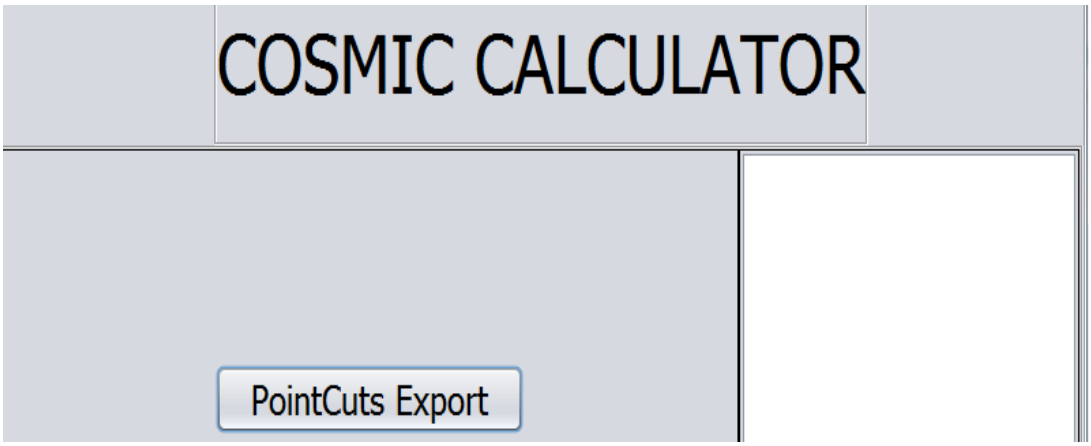

Lib: uygulama kütüphaneleri

Hsqld: Uygulama veritabanı

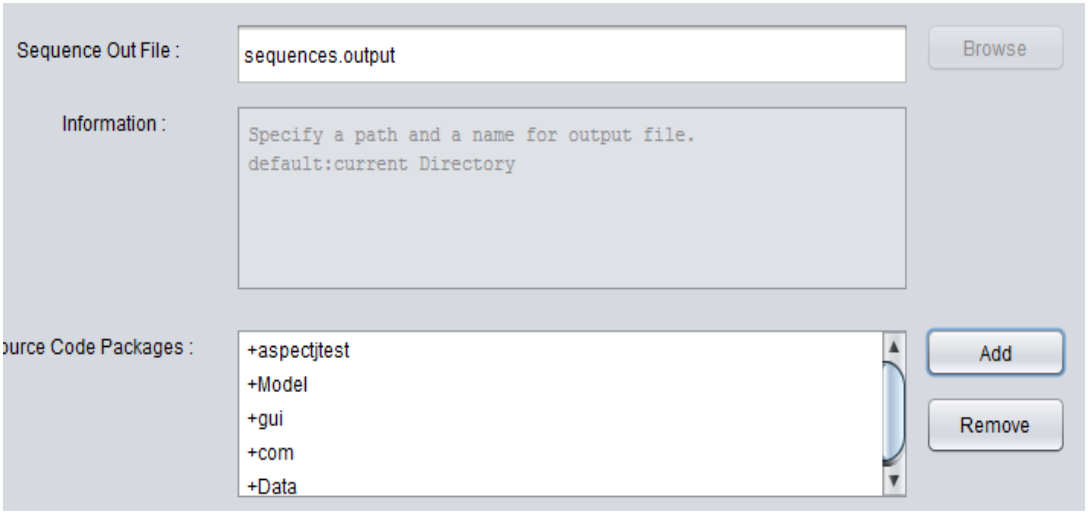
Src: Kaynak Kod(zorunlu değil)

Bin: Derlenmiş uygulama sınıfları

4. Java, aspectj ve uygulama hazırsa “COSMIC Calculator” u çalıştıralım.
5. Birinci adım olan “PointCuts Export” u çalıştıralım.

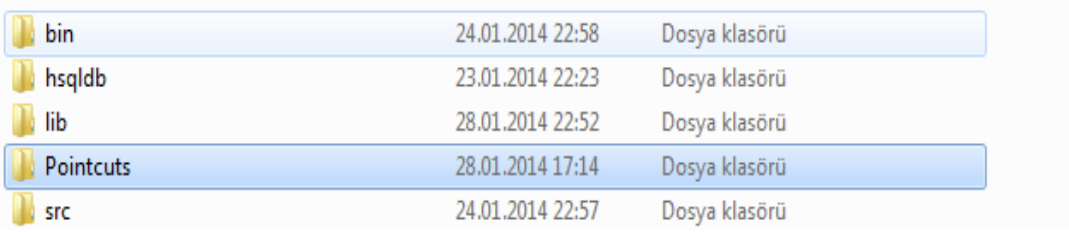



6. Uygulama derlenmiş sınıflarını içeren paketlerin isimlerini gir.

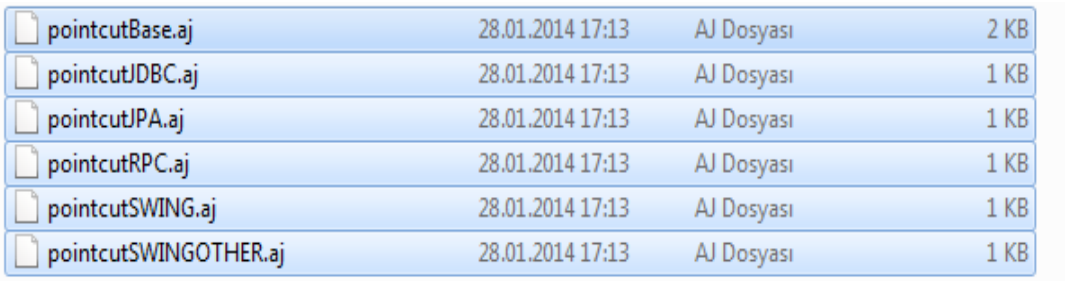



7. “Create” butonuna tıkla.

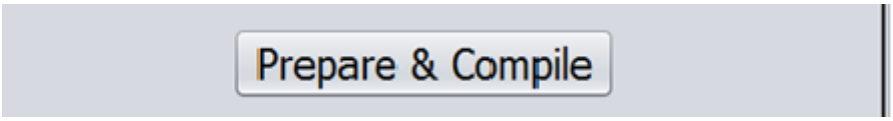
   “Cosmic Calculator” uygulamasını çalıştırdığınız konumda “Pointcuts” isimli bir dizin oluşturulacaktır.



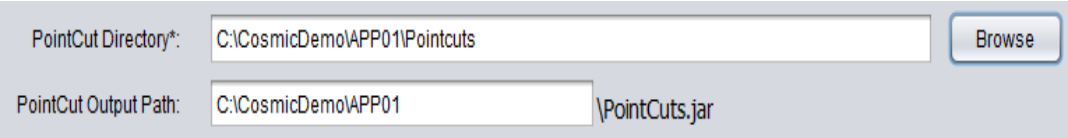

   Bu klasörün içerisinde temel tavsiye yordam ve kesme noktaları tanımlıdır. Henüz derlenmediği için bir java sınıfıymış gibi üzerinde işlem yapılabilir.



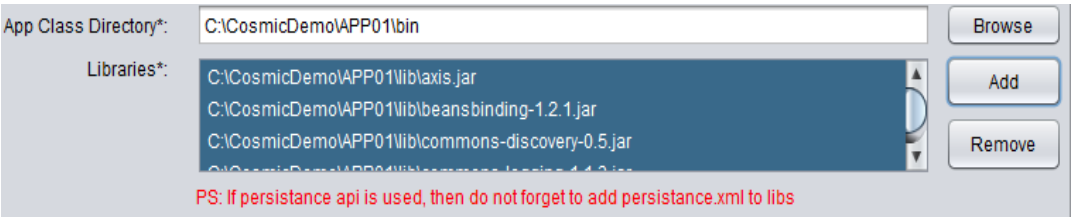

8. “Exit” butonunu tıklayarak birinci modülü kapatın.
9. Ana uygulama penceresinde yer alan “Prepare & Compile” butonunu tıklayarak ikinci modülü başlatın.



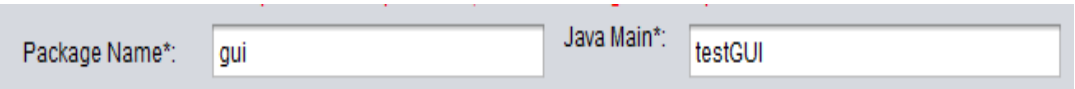

10. Derlenecek kesme noktalarının konumunu belirtin ve oluşturulacak “.jar” dosyası için konum belirtin.



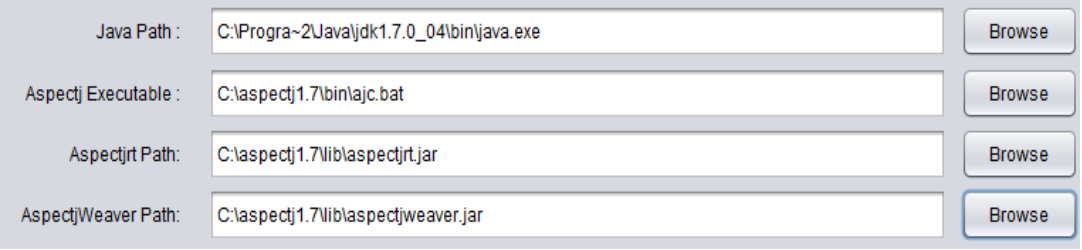

11. Uygulamanın derlenmiş sınıflarının bulunduğu dizini girin. Uygulamanın ihtiyaç duyduğu kütüphaneleri belirtin. (META-INF/persistence.xml dosyası varsa eklemeyi unutmayın).



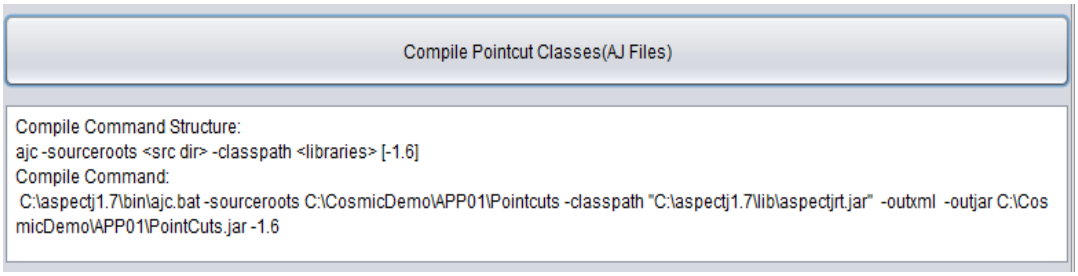

12. Hedef uygulamayı çalıştıracak olan “main” metodu içeren sınıfın paketinin adını ve sınıfın adını belirtin.




13. Java ve AspectJ programlarına ait ilgili dosyaların konumlarını belirtin.

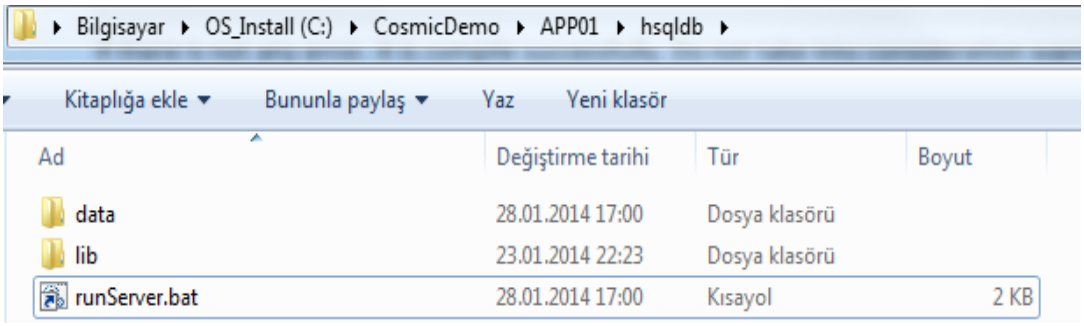



14. “Next” butonunu tıklayın.
15. Çıktı olarak alttaki resimde gösterildiği gibi çalıştırılabilir bir komut dizisi oluşturulacaktır. İsterseniz “Compile PointCut Classes” butonu tıklayın, ister komut satırına yapıştırıp istediğiniz düzenlemeyi yaparak çalıştırın. Sonucunda “.aj” uzantılı dosyalar derlenecek ve “PointCuts.jar” dosyası oluşacaktır.

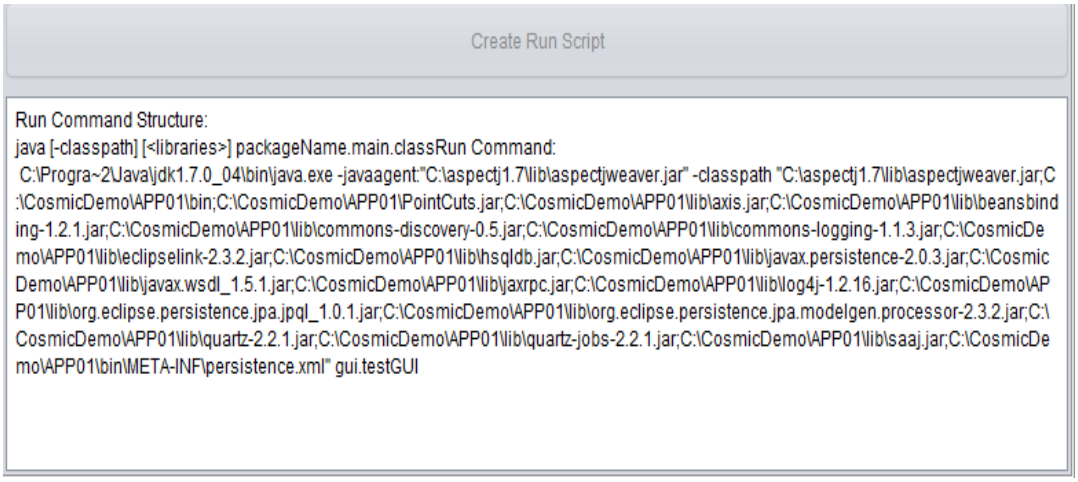



Eğer bir hata oluşmaz ise işlem başarıyla tamamlanacaktır. Derleme sırasında oluşturulan uyarı mesajlarını dikkate almanız gerekmemektedir.

16. APP01 in kullandığı “hsql database” ini çalıştırın. Bunun için hsqldb dizini altındaki “runServer.bat” ı çalıştırmanız yeterlidir.

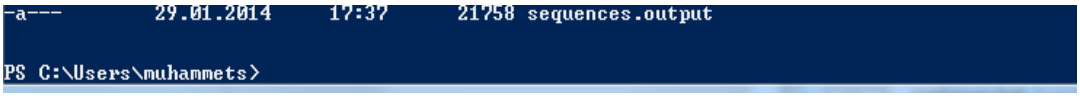



17. APP01 için oluşturulmuş olan yükleme zamanı dokuma işlemi eşliğinde çalıştırma için ikinci bölümdeki komut dizisini kopyalayın ve komut satırında çalıştırın.




*Dikkat*: java agent in çalıştırılabilmesi ve yükleme zamanı dokuma eylemi için aspectjweaver.jar dosyası ve pointcuts.jar dosyası komutta yer almalıdır.

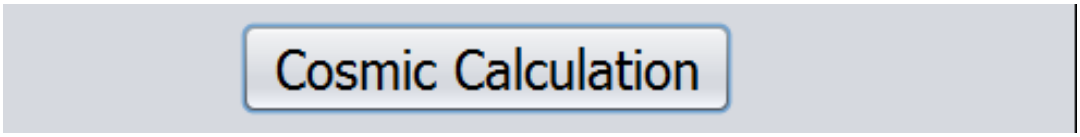
18. Normal bir aktör olarak uygulamayı kullanın ve metin formunda dizge diyagramlarını içeren çıktı dosyasını elde edin.
19. Bu kullanım sonucunda “sequences.output” adında bir dosya oluşacaktır.



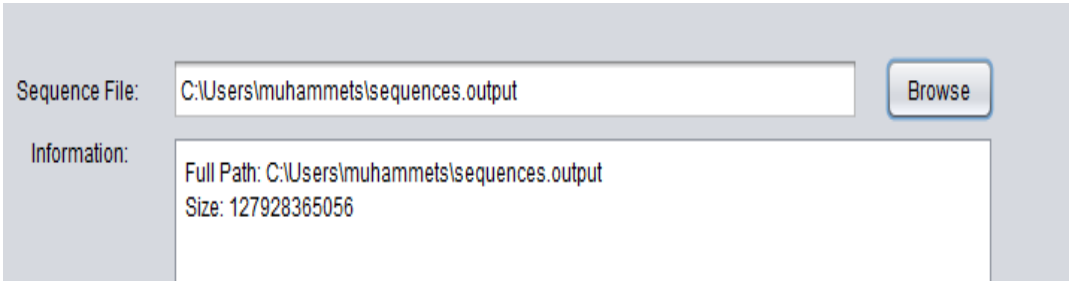

Bu dosyayı COSMIC büyüklüğü hesaplamak için girdi olarak kullanacağız. Dosya içeriğine bakıldığında alttaki gibi metinlerden oluştuğu görülecektir.

```
PS C:\Users\muhammets> cat .\sequences.output
Start:>void gui.testGUI.main(String[])
End:>void gui.testGUI.main(String[])
Start:JDBC>ResultSet org.hsqldb.jdbc.JDBCStatement.executeQuery(String)
 call database_version()
Start:JDBC>ResultSet org.hsqldb.jdbc.JDBCStatement.executeQuery(String)
 call database_version()
End:JDBC>ResultSet org.hsqldb.jdbc.JDBCStatement.executeQuery(String)
End:JDBC>ResultSet org.hsqldb.jdbc.JDBCStatement.executeQuery(String)
Start:>void gui.testGUI.initComponents()
Start:JPA>PreparedStatement java.sql.Connection.prepareStatement(String)
 SELECT ID, BORNPLACE, NAME, SURNAME FROM PERSON
Start:JPA>PreparedStatement org.hsqldb.jdbc.JDBCConnection.prepareStatement(String)
 SELECT ID, BORNPLACE, NAME, SURNAME FROM PERSON
End:JPA>PreparedStatement org.hsqldb.jdbc.JDBCConnection.prepareStatement(String)
End:JPA>PreparedStatement java.sql.Connection.prepareStatement(String)
End:>void gui.testGUI.initComponents()
Start:>Integer Data.Person.getId()
End:>Integer Data.Person.getId()
Start:>String Data.Person.getName()
End:>String Data.Person.getName()
Start:>String Data.Person.getSurname()
End:>String Data.Person.getSurname()
Start:>String Data.Person.getBornplace()
End:>String Data.Person.getBornplace()
Start:>Integer Data.Person.getId()
End:>Integer Data.Person.getId()
Start:>String Data.Person.getName()
End:>String Data.Person.getName()
Start:>String Data.Person.getSurname()
End:>String Data.Person.getSurname()
Start:>String Data.Person.getBornplace()
End:>String Data.Person.getBornplace()
```

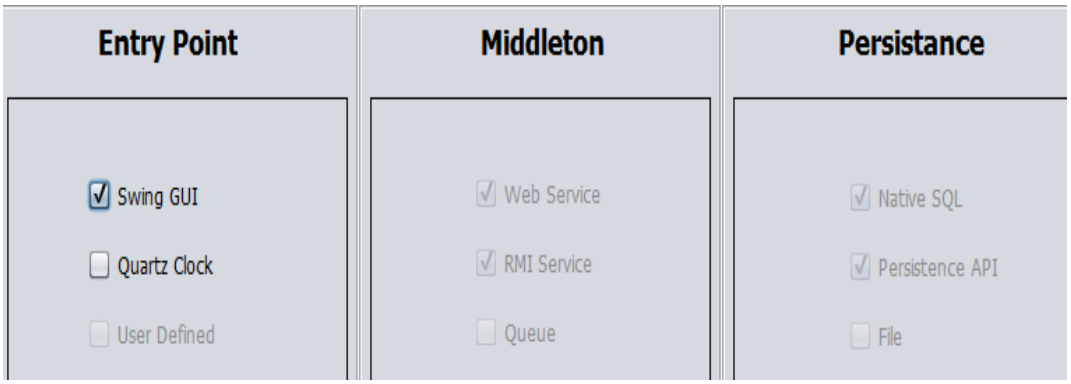
20. Üçüncü ve son adım olan “Cosmic Calculation” modülünü çalıştırın.



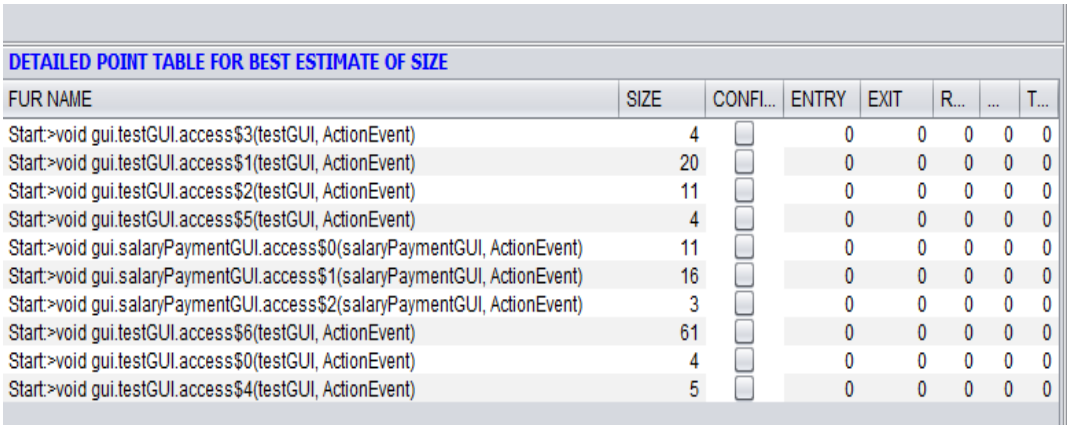

21. Uygulama kullanımıyla oluşan sequence.output dosyasını seçin. “Next” tuşunu tıklayın




22. Swing GUI giriş noktasını seçerek “Next” butonunu tıklayın.



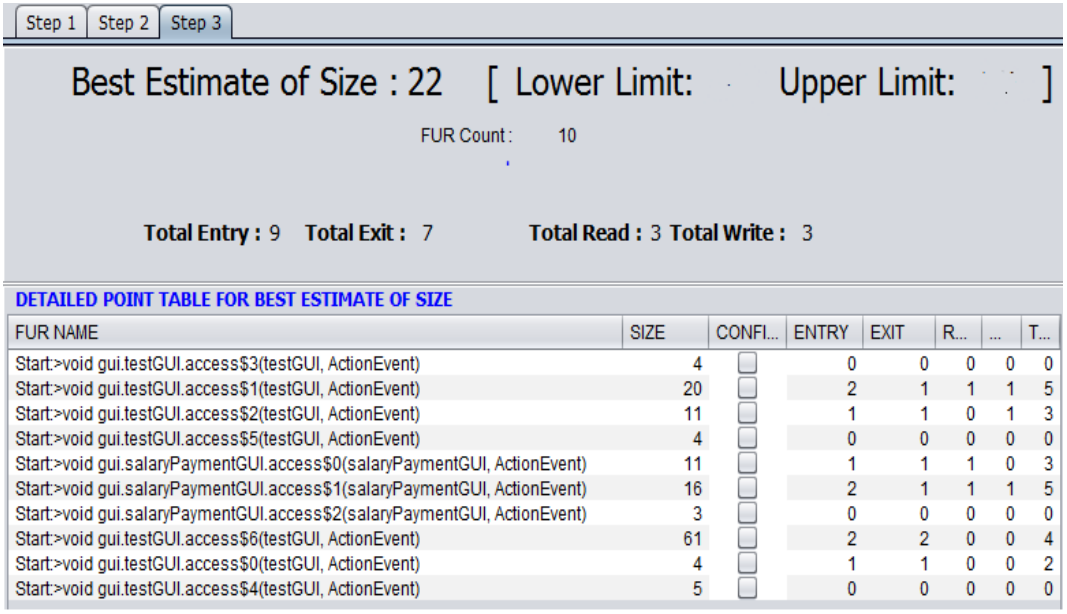
23. Aday işlevsel süreçlerin listesini gösteren ekran görüntülenecektir.



DETAILED POINT TABLE FOR BEST ESTIMATE OF SIZE

| FUR NAME | SIZE | CONFI... | ENTRY | EXIT | R... | ... | T... |
|---|---|---|---|---|---|---|---|
| Start:>void gui.testGUI.access$3(testGUI, ActionEvent) | 4 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$1(testGUI, ActionEvent) | 20 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$2(testGUI, ActionEvent) | 11 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$5(testGUI, ActionEvent) | 4 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.salaryPaymentGUI.access$0(salaryPaymentGUI, ActionEvent) | 11 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.salaryPaymentGUI.access$1(salaryPaymentGUI, ActionEvent) | 16 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.salaryPaymentGUI.access$2(salaryPaymentGUI, ActionEvent) | 3 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$6(testGUI, ActionEvent) | 61 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$0(testGUI, ActionEvent) | 4 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$4(testGUI, ActionEvent) | 5 | | 0 | 0 | 0 | 0 | 0 |

24. "Calculate" butonunu tıklayarak uygulamanın COSMIC büyüklüğünü hesaplayın.




DETAILED POINT TABLE FOR BEST ESTIMATE OF SIZE

| FUR NAME | SIZE | CONFI... | ENTRY | EXIT | R... | ... | T... |
|---|---|---|---|---|---|---|---|
| Start:>void gui.testGUI.access$3(testGUI, ActionEvent) | 4 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$1(testGUI, ActionEvent) | 20 | | 2 | 1 | 1 | 1 | 5 |
| Start:>void gui.testGUI.access$2(testGUI, ActionEvent) | 11 | | 1 | 1 | 0 | 1 | 3 |
| Start:>void gui.testGUI.access$5(testGUI, ActionEvent) | 4 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.salaryPaymentGUI.access$0(salaryPaymentGUI, ActionEvent) | 11 | | 1 | 1 | 1 | 0 | 3 |
| Start:>void gui.salaryPaymentGUI.access$1(salaryPaymentGUI, ActionEvent) | 16 | | 2 | 1 | 1 | 1 | 5 |
| Start:>void gui.salaryPaymentGUI.access$2(salaryPaymentGUI, ActionEvent) | 3 | | 0 | 0 | 0 | 0 | 0 |
| Start:>void gui.testGUI.access$6(testGUI, ActionEvent) | 61 | | 2 | 2 | 0 | 0 | 4 |
| Start:>void gui.testGUI.access$0(testGUI, ActionEvent) | 4 | | 1 | 1 | 0 | 0 | 2 |
| Start:>void gui.testGUI.access$4(testGUI, ActionEvent) | 5 | | 0 | 0 | 0 | 0 | 0 |

25. Böylece ölçme işlemi tamamlanmış olacaktır.

# ÖZGEÇMİŞ

**Kimlik Bilgileri**


| | | |
|---|---|---|
| Adı Soyadı | : | Muhammet Ali SAĞ |
| Doğum Yeri | : | Erzurum |
| Medeni Hali | : | Bekar |
| E-posta | : | muhammetalisag@gmail.com |
| Adresi | : | Altay Mah. 35. Sok. Nilüfer Apt. No:8 Eryaman / Ankara |


**Eğitim**

| | | |
|---|---|---|
| Lise | : | Erzurum Lisesi |
| Lisans | : | Hacettepe Üniversitesi Bilgisayar Mühendisliği |
| Yüksek Lisans | : | Hacettepe Üniversitesi Bilgisayar Mühendisliği |

**Yabancı Dil ve Düzeyi**

- İyi

**İş Deneyimi**

- 4 Yıl

**Deneyim Alanları**

- Yazılım Mühendisliği, Sistem Mühendisliği

**Tezden Üretilmiş Projeler ve Bütçesi**

-

**Tezden Üretilmiş Yayınlar**

-

**Tezden Üretilmiş Tebliğ ve/veya Poster Sunumu ile Katıldığı Toplantılar**

M.A.Sag and A.Tarhan. Measuring COSMIC Software Size from Functional Execution Traces of Java Business Applications, in proceedings of IWSM-Mensura 2014, IEEE CPS