T.C.
KASTAMONU ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÖZEL DURUMLARIN ORTAYA ÇIKMASINA VE GELİŞTİRME YAŞAM DÖNGÜSÜNÜN ERKEN SAFHALARINDA İSTİSNALARIN MODELLENMESİNE YÖNELİK BİR YAKLAŞIM

Mohamed Ali Salem HAGAL

Danışman	**Prof. Dr. Fatma KANDEMİRLİ**
Jüri Üyesi	**Prof. Dr. İdris KABALCI**
Jüri Üyesi	**Doç. Dr. Aybaba HANÇERLİOĞULLARI**
Jüri Üyesi	**Dr. Öğr. Üyesi Can Doğan VURDU**
Jüri Üyesi	**Dr. Öğr. Üyesi Javad RAHEBİ**

DOKTORA TEZİ
MALZEME BİLİMİ VE MÜHENDİSLİĞİ ANA BİLİM DALI

KASTAMONU – 2018

TEZ ONAYI

Mohamed Ali Salem HAGAL tarafından hazırlanan " **Özel Durumların Ortaya Çıkmasına Ve Geliştirme Yaşam Döngüsünün Erken Safhalarında İstisnaların Modellenmesine Yönelik Bir Yaklaşım**" adlı tez çalışması aşağıdaki jüri üyeleri önünde savunulmuş ve **oy birliği** ile Kastamonu Üniversitesi Fen Bilimleri Enstitüsü **Malzeme Bilimi Ve Mühendisliği Ana Bilim Dalı**' nda **DOKTORA TEZİ** olarak kabul edilmiştir.

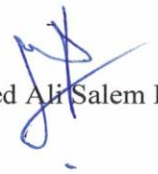
Danışman	Prof. Dr. Fatma KANDEMİRLİ Kastamonu Üniversitesi	.....
Jüri Üyesi	Prof. Dr. İdris KABALCI Karabük Üniversitesi	.....
Jüri Üyesi	Doç. Dr. Aybaba HANÇERLİOĞULLARI Kastamonu Üniversitesi	.....
Jüri Üyesi	Dr. Öğr. Üyesi Can Doğan VURDU Kastamonu Üniversitesi	.....
Jüri Üyesi	Dr. Öğr. Üyesi Javad RAHEBİ Türk Hava Kurumu Üniversitesi	.....

.../.../

Enstitü Müdür Prof. Dr. Hasbi YAPRAK .....

TAAHHÜTNAME

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildirir ve taahhüt ederim.



Mohamed Ali Salem HAGAL

ÖZET

Doktora Tezi

ÖZEL DURUMLARIN ORTAYA ÇIKMASINA VE GELİŞTİRME YAŞAM DÖNGÜSÜNÜN ERKEN SAFHALARINDA İSTİSNALARIN MODELLENMESİNE YÖNELİK BİR YAKLAŞIM

Mohamed Ali Salem HAGAL
Kastamonu Üniversitesi
Fen Bilimleri Enstitüsü
Malzeme Bilimi ve Mühendisliği Ana Bilim Dalı

Danışman: Prof. Dr. Fatma KANDEMİRLİ

Yazılım güvenilirliği, yazılımın istenen hedeflerine ulaşmak için dikkate alınması gereken en önemli faktördür.

Paydaşlarının gereksinimlerini etkin bir şekilde karşılayan bir yazılım oluşturmak için yazılım geliştiricilerine yardımcı olma yollarını ve araçlarını tanımlamak için pek çok çaba gösterilmiştir.
Yazılım mühendisliği organize bir şekilde birçok aşamayı içeren her biri geliştirme sürecinin geri kalanında etkileri olan sistematik bir yaklaşım sunar.
Bu nedenle, bu aşamaların başarısı, yazılım bütünlüğü açısından önemlidir. Gereksinim mühendisliği (GM) aşaması bu aşamalardaki en önemli aşamadır ve geliştirilecek yazılım sisteminin ne anlama geldiğine dayanmaktadır, ancak birçok yazılım sistemi görevlerini tatmin edici bir şekilde yerine getirmeye devam etmemektedir. Bu tür başarısızlıkların ortaya çıkmasına katkıda bulunan en önemli problemler, bu tür sistemlerin gerekliliklerinin oluşturulmasındaki zayıflık veya bunların uygulanması sırasında sorunların ortaya çıkmasıdır.
Bu aksaklıklar, hedeflere ulaşmada sistemlerin başarısız olmasına neden olmaktadır.
Bu genellikle, gelişim sürecinin ilk aşamalarından kendileriyle ilgilenmeyen ya da ileri aşamalarda bile onları ihmal etmiş olabilen geliştiricilere bağlı olan istisnadır.
Bu tez çalışmasında, geliştirme sürecinde daha sonra tespit edilen kullanım istisnalarının yükünü azaltmak için istisnai durumlarla başa çıkmada RE aşamasını geliştirerek, SDLC'nin erken aşamalarında istisnaları keşfetmeye yardımcı olmayı amaçlayan birkaç adım içeren bir yaklaşım sunulmuştur. Bu adımlar: istisnaların sınıflandırılması, keşif ve modellemedir.

Eksik gereksinimler konusu, ilk istisna sınıflaması olarak kabul edilir; Bu nedenle, ilk aşamada istisnalarla baş etme sürecini göz önünde bulundurarak gereksinimleri ortaya çıkarmaya yardımcı olacak bir yaklaşım önerilmiştir. Zayıf kullanıcıların sistemle etkileşimi ve zayıf sistem yanıtları, ikinci istisna sınıflandırması ve birinci sınıflamanın tamamlayıcısı olarak kabul edilir. İstisnalar tespit sürecini kolaylaştırmak için, kullanıcı gereksinimleri senaryolarını yazmak için kısıtlayıcı kurallar önerilmiştir. Ayrıca, bu istisna sınıflandırması için python programlama dili kullanılarak EDT adlı bir yazılım geliştirilmiştir.

Ek olarak, UCM'lerin görsel senaryoları, normal senaryo adımlarının açıklığa kavuşturulmasında kullanıcı gereksinimlerinin modellenmesinde uyarlanmıştır ve ayrıca önerilen EDT, tespit edilen istisnaları da içerecek şekilde geliştirilmiştir. Yaklaşımımızın, yazılım geliştiricilerinin SDLC'nin erken aşamasında istisnalara odaklanmasını teşvik edeceğine inanıyoruz.

Anahtar Kelimeler: Yazılım mühendisliği, istisnalar, doğal dil işleme, modelleme, yazılım testi.

2018, 118 sayfa
Bilim Kodu: 91

ABSTRACT

Ph. D. Thesis

AN APPROACH OF ELICITING EXCEPTIONS AND MODELING EXCEPTIONS HANDLING AT EARLY STAGE OF THE DEVELOPMENT LIFECYCLE

Mohamed Ali Salem Hagal

Kastamonu University
Institute of Science
Department of Materials Science and Engineering

Supervisor: Prof. Dr. Fatma KANDEMİRLİ

Software reliability is the most important factor that should be considered in order to achieve the desired objectives of the software. Many efforts have been made to identify ways and means of helping software developers to build software that effectively meets the requirements of their stakeholders. Software engineering offers a systematic approach that contains many stages in an organized manner, each of which has output that effects on the rest of the development process. Thus, the success of these stages is therefore essential for software rigidity. Requirements engineering (RE) stage is the most important stage in these stages, and is based on the definition of what the software system to be developed should do, however many software systems failing to continue to perform their duties satisfactorily. The most important problems contributing to the emergence of such failures are the weakness in building the requirements of such systems or the emergence of problems during their implementation and delivery to the applicants. These failings result in systems failing to achieve their objectives. This is the exception, which is usually due to developers, who are not interested in them from the early stages of the development process, or may have ignored them even in the advanced stages.

In this thesis, we presented an approach that contains several steps aimed at trying to help discover exceptions at an early stage of the SDLC, through improving the RE stage in dealing with exceptions early, in order to reduce the burden of handling exceptions if detected later in the development process.These steps are: classification of exceptions, their discovery and modeling.

Missing requirements issue is considered to be the first exceptions classification; therefore we suggested an approach to help elicit the requirements in a manner that took into consideration the process of dealing with exceptions at an early stage. Poor users' interaction with the system and weak system responses is considered as the second exception classification, and the complement to the first classification. To facilitate the process of exceptions detection, restrictive rules are proposed for writing the scenarios of user requirements. Also, a tool named EDT was developed using python programming language to address this classification of exception.

In addition, UCMs visual notations were adapted in the modeling of user requirements in the clarification of normal scenario steps, and also improed to include the exceptions detected by the proposed tool. We belief our approach will encourage software developers focus on exceptions at early stage of SDLC.

Key Words: Software engineering, exceptions, natural language processing, modeling, software testing

2018, 118 pages
Science Code: 91

TEŞEKKÜR

Danışmanım Prof. Dr. Fatma KANDEMİRLİ' nin çalışmamın en zor zamanlarında verdiği sonsuz destek ve teşvik için gösterdiği çabalara çok teşekkür ediyorum. Tüm çalışmam boyunca gösterdiği sabrı ve hoşgörüyü takdir ediyorum. Bu çalışmayı doğru ve zamanında geri bildirim ile değerlendirirken bana değerli zaman ve kaynaklarından yararlanma konusunda yeterince fırsat ve laboratuvar imkanlarını sağladığından dolayı ayrıca teşekkür ederim.

Ayrıca, tez çalışmam sürecince önerileri ve destekleri için Dr. Öğr. Üyesi Can Doğan VURDU ve Dr. Öğr. Üyesi Javad RAHEBİ'ye teşekkürlerimi sunarım. Kastamonu Üniversitesindeki arkadaşlarıma ve Fen Bilimleri Enstitüsü çalışanlarına, bu çalışmanın hazırlık aşamasında beni sürekli olarak cesaretlendirdikleri ve destekledikleri için teşekkür ediyorum

Bu çalışmayı tamamlayana kadar yanımda duran aile üyelerimin sonsuz desteğini ve sabrını asla unutamam.

Mohamed Ali Salem HAGAL
Kastamonu, Ekim, 2018

İÇİNDEKİLER

Sayfa

SİMGELER ve KISALTMALAR DİZİNİ

SDLC	Yazılım geliştirme yaşam döngüsü
UML	Birleştirilmiş modelleme dili
UCMs	Durum haritalarını kullanma
NLP	Doğal dil işleme
POS	Konuşmanın bir bölümüne
NLTK	Doğal dil araç seti
EDT	İstisna algılama yazılımı

ŞEKİLLER DİZİNİ

Sayfa

TABLOLAR DİZİNİ

Sayfa

1. GİRİŞ

Yazılım; bilgisayar programlarını, uygulama programlarını, yazılım geliştiricileri tarafından bir dizi talimatı uygulamak için hazırlanan ve bir görevi veya nesneyi tanımlamak için kullanılan genel bir terimdir. Yazılım sistemleri karmaşık bir şekilde büyümüştür. Bunun nedeni, yoğunlaşmanın sadece bu sistemlerin sağladığı hizmetlerde değil, daha sonra ortaya çıkabilecek anormal durumlar üzerinde de olması gerektiğidir.

Sağlam bir yazılım sisteminin geliştirilmesi öncelikle altyapısına ve gerekli ihtiyaçların net bir şekilde tanımlanmasına dayanır. Dolayısıyla, geliştiricilerin gelişim sürecinin sonraki aşamalarında göz önüne alması gereken potansiyel durumların çoğunu veya tümünü içeren gereksinimleri üzerinde güçlerini inşa ettikleri için, aşamalardan geri kalanlar daha az sorunludur.

Yazılım geliştirme aşamaları organize bir süreçtir, ancak bu aşamalar genellikle geliştiricilerin deneyimine ve verimliliğine ve bu sistemlerin paydaşlarının derecesine bağlıdır.

Sistem kullanıcıları ve geliştiriciler arasındaki uçurumu azaltmak için kullanılan çeşitli teknikler vardır. Yazılım mühendisliği, yüksek kaliteli yazılımların oluşturulmasına katkıda bulunmak için birçok aşamada disiplinli bir metodu içermektedir. Bu aşamalar şunları içerir: bir sistemin gereksinimleri, tasarımı, uygulanması, testi ve bakımı. Bununla birlikte, net ve çelişkili olmayan şartlar yaratmaya odaklanılmalıdır.

Gereksinim mühendisliği (SD), SDLC'deki ilk ve en kritik faz; çünkü bir yazılım sisteminin ne yapması gerektiğine, ihtiyaçların ortaya çıkartılmasını içeren faaliyetleri yoluyla yoğunlaşmaktadır. Analiz; şartname; doğrulama ve yönetim faaliyetleri (Konaté, Sahraoui, & Kolfschoten, 2014) ve herhangi bir yazılımı yeniden geliştirme süreci pahalıdır ve bazen yanlış veya eksik gereksinimler üzerine inşa edilmiş ise sistemin yeniden yapılandırılmasına neden olabilir. Dahası, bu sistemlerin birçoğu başarısız olabilir veya bu hataları düzeltme işlemi daha fazla

zaman ve emek gerektirebilir veya maliyet nedeniyle alternatif sistemlerin araştırılması olabilir

Ek olarak, gereksinimlerin ortaya çıkartılması hala zor bir faaliyettir ve paydaş ihtiyaçlarına en iyi şekilde ulaşılmasına ilişkin genel bir fikir birliği bulunmamaktadır; çünkü çeşitli beceriler gerektirir (sosyal ve uzman); Bu nedenle bu faaliyet sırasında göz önüne alınması gereken bir takım güçlükler bulunmaktadır. Zorluklar şu noktalarda özetlenebilir: (Davey & Parker, 2015; Pressman, 2005):

- Sistem kapsamının zayıf tanımlanması;
- Paydaşların ihtiyaçları konusunda eksik bir anlayışa sahip olması;
- Menfaat sahipleri, bilgisayar yetenekleri ve sınırlamaları hakkında yeterince bilgiye sahip olmamaları;
- Geliştiriciler problem alanıyla ilgili bilgi sahibi olmamaları;
- Kullanıcılar ve geliştiriciler birbirlerini anlamıyor olması;
- Menfaat sahiplerinin çelişki görüşlere sahip olması;
- Kullanıcıların, kapsamın sızmasına neden olan gereksiz taleplerinin olması;
- Şartların belirsiz olması ve test edilebilir olmaması;
- Kararsız gereksinimlerin olması.

Gereksiz ve eksik gereksinimler bir sistem hatasına neden olan ve sistemin istisnalar olarak adlandırılan anormal durumlara eğilimli olmasına sebep olan önemli bir konu olarak görülebilir (Cailliau & van Lamsweerde, 2014a). Her iş sürecinin (kullanıcı gereksinimi) bir hedefi vardır ve hedefine ulaşamaması durumunda; bir istisna oluştuğu anlamına gelir (Casati, Fugini, & Mirbel, 1999). Yukarıda belirtilen zorlukların üstesinden gelmek için, şartlar aşağıdaki özellikleri sağlamalıdır (Pressman, 2005).

- Doğru
- Tutarlı
- Doğrulanabilir
- İzlenebilir

Bir diğer önemli konu, yazılım sistemlerinin hatalı şekilde durdurulmasına veya çalışmasına neden olabilecek hataların ortaya çıkmasıdır ve bu hataları işlemek çok masraflı olacaktır. Dolayısıyla, hatalarla erken başa çıkma, erken dönem kanser tedavisine benzer ve son aşamalarda saptanırsa pek çok hastanın yaşamını risk altına sokar. Özel durum kayıt yazılımı, ilk adımda görünür hatalar olarak sınıflandırılabilir. Çünkü birçok geliştiricinin farklı görüşleri vardır: (1) Bazıları zaman aldığını varsaymaktadır; (2) bazıları odaklamanın SDLC'nin test aşamasında olması gerektiğini varsaymaktadır ve (3) bazıları ise, programlama dili tarafından dikte edilmesi gerektiğine inanmaktadır. Bununla birlikte, şartlar aşamasındaki istisnalara odaklanma yoktur ya da zayıf kalmaktadır.

Sistem belgelerini geliştirecek tüm durumlara ve ilgili faaliyetlere de önemli bir dikkat gösterilmelidir. İstisna, bu anormal durumlardan birinin daha fazla dikkat gerektirmesi ve eksik bir gereksinim nedeniyle daha sonra ortaya çıkabilir (Cailliau & van Lamsweerde, 2013). Ayrıca, geliştirilmesi gereken sistemin iş kurallarını ihmal etmesi nedeniyle ortaya çıkabilir.

Bu tür durumlara odaklanmak için derin bir içgörü gereklidir ve bu durumları erken tahmin etme girişimini ve olumsuz etkileri daha sonra önlemeye yardımcı olacaktır. İnanıyoruz ki ihmal edilirse daha fazla geliştirme masrafı veya sistem hatası ortaya çıkabilir.

Kullanıcıların ihtiyaçlarının yanlış anlaşılmasından kaynaklanan gereksinimlerin sıkça değişmesi veya dalgalanması genellikle sistemin başarısızlık nedenlerinden biridir. Toplanan gereklilikler ve yeterli paydaşların katılımı yazılım başarısının önemli faktörleridir.

Eksik ihtiyaçların ortaya çıkması, paydaşların ve kullanıcıların memnuniyetsizliğine yol açacak pek çok soruna neden olabilir. Daha sonraki aşamalarda bu sorunu gidermek pahalı olacaktır; çünkü bu işlem için daha fazla gayret ve maliyet getirebilir

Uygulama aşamasında, bir istisna meydana geldiğinde, bir işlemin veya bir programın normal akışı anormal şekilde sonlanır. Bu istisnalar birçok farklı

nedenden dolayı meydana gelebilir, bazıları kullanıcı hatalarıdır ve bazıları programcı hataları veya fiziksel hatalardır (donanım hataları ve kaynak uygunluğu gibi). Hataların çoğu, kötü yazılım tasarımından ve eksik gereksinimlerden kaynaklanmaktadır. Tüm bunlar paydaşların ihtiyaçlarını karşılamayan bir yazılım sistemi sunmaya katkıda bulunabilir.

Kontrol edilen istisnalar derleme sırasında yakalanabilir ve sistem geliştiricileri uygulama aşamasında bu istisnaları işlemelidir (bu istisnalar SDLC'nin önceki aşamalarından devralınmadıkça). Denetlenmeyen istisnalar (örneğin, eksik gereksinimlerin neden olduğu istisnalar) derleyici tarafından yakalanamaz ve genellikle kontrol edilen istisnalardan daha fazla zaman ve çaba gerektirir; çünkü yeniden ortaya çıkarma ve müzakere süreçleri gerekli olabilir. Bu, bir sistemin karmaşıklığını arttıracaktır, çünkü gereksinimlerdeki değişiklik daha sonra SDLC'nin sonraki aşamalarındaki modifikasyonları gerektirmektedir.
Literatüre ek olarak, erken aşamalardaki istisnalarla uğraşmak, geliştiricilerin SDLC'nin erken aşamasında istisnaları dikkate almalarına yol açan bir yaklaşım ve mekanizma eksikliğinden kaynaklanmaktadır.

Ayrıca, mevcut tekniklerin bazıları özel alanlar için özel olarak tanımlanmıştır ve bazıları ise yüksek düzeyde soyutlama teknikleridir. Buna göre, bu teknikler çeşitli alanların modellenmesinde kullanılması için uygun değildir. Ayrıca, hataların ve istisnaların düzeltilmesi, yazılım geliştirme maliyetinin yanı sıra belirlenen geliştirme süresini de aşabilir.

Bu sorunların öngörülmesi hala zor bir süreç olmakla birlikte, geliştirme sürecini iyileştirme ve bu etkilerden kaçınmaya yardımcı olacak çözümler bulunmasına çalışılmalıdır.

Yazılım sisteminin yüksek güvenilirlik ve sürekli kullanılabilirliğini sağlamak için SDLC'nin ilk aşamalarında istisna işlemlerinin ele alınması özellikle tavsiye edilmiştir. Bu, geliştiricilerin istisnaları erken aşamada tespit edip geliştirme yaşam döngüsü boyunca bunları ele almasına yardımcı olan bir mekanizmanın kurulmasını gerektirir.

Bu çalışmanın önemini anlamak için, araştırmaya büyük motivasyon sağlayan araştırma soruları şunlardır:

S1. İstisnaların başlıca nedenleri nelerdir?
S2. Geliştiricilerin çoğu, gelişimin erken evresinde istisnaları neden göz ardı ediyor?
S3. SDLC'nin erken safhasındaki istisnaların tespit edilmesinin faydaları nelerdir?
S4. Geliştiricilere, istisnaları açıkça belirtmeleri ve bunları nasıl ele alması gerektiği konusunda yardımcı olmak için ne yapmalıdır?

Bu çalışmanın kapsamında, sistemin menfaat sahiplerinin ihtiyaçlarını iyi ifade etmelerine yardım ederek, bu gereksinimlerin tam olarak anlaşılmasına yol açacak bir kullanıcı gereksinimi yaratmaya yol açan sistematik bir yaklaşım tanımlamaktır. Bu mekanizma, sistem geliştiricileri ve potansiyel paydaşlar arasında, sistemin yapması gereken ortak bir tanımlamaya ulaşma ve daha sonra bir sistemi anormal durumlara eğilimli hale getiren olası istisnalarla nasıl başa çıkılacağı konusundaki müzakere sürecini geliştirecektir.

Bu çalışmanın kapsamı aşağıdaki gibi özetlenebilir:

1. SDLC'nin erken safhasında yoğunlaştırılmış istisnaların sınıflandırılması;

2. İstisnaların tespit edilmesi istisnaların modellenmesi ve ele alınması için ön koşul niteliğinde tutarlı ve eksiksiz bir girdi gerektirir. Bu süreci organize eden ve çözen bir yaklaşımın olmaması kalkınma sürecinin karmaşıklığını artırabilir daha sonra bu sürecin tekrar çalışmasına ve zaman alan bir çabaya götürebilir. Dolayısıyla, proaktif bir adım olarak, eksik gereksinimler sistem arızalarına neden olup, istisnaların ortaya çıkmasının etki faktörlerinden biri olarak düşünülebilir. Dolayısıyla, bu sorunun etkisini azaltmak için mükemmel gereksinimleri ortaya koyma ve açıklama yaklaşımı önerilmiştir.
   - Kullanıcı ihtiyaçlarını ortaya çıkarmak için organize bir yaklaşım sağlamak gerekir. UML kullanım senaryoları bu aşamada tanımlanmıştır.
   - Kullanıcıların geri bildirimlerini takip etmek için eksiksiz ve kesin şartlar üretmek ve kullanım senaryolarını açıklığa kavuşturmak için bir kağıt prototip

yaklaşımı tanımlamak gerekir. Bu nedenle, UCM'ler bu senaryoları netleştirmek için bir grafik araç olarak kullanılır.

3. Sınırlı bir doğal dilde yazılmış kullanım senaryoları olgularından olası istisnaları ortaya çıkarmak için bir mekanizma ve her kullanım durumunun içerebileceği olası istisnaları çıkarmak için doğal bir dil işleme bilgisi kullanılır. Python programlama dili kullanan bir program geliştirilmiştir.

4. Olası çözümler mekanizması ile istisnaların ele alınması.

   - UCM'leri kullanarak göreli kullanıcı gereksinimleriyle istisna yayılımını eşleştirme.

Bu çalışmanın ana amacı, yazılım geliştiricilerinin ve sistem paydaşlarının bir sistemin daha sonra karşılaşabileceği anormal durumlara yoğunlaşmasına ve bu durumların etkilerini azaltacak uygun bir işleme mekanizmasını teklif etmesine yardımcı olacak bir yaklaşımı tanımlamaktır. Bu hedefler aşağıdaki gibi özetlenebilir:

- İstisna türleri ve nedenleri hakkında kapsamlı bir çalışma. Bu, sorunu azaltmak için kullanılan tekniklerin incelenmesi ve bu yöntemlerin bu etkilerin azaltılmasına nasıl katkıda bulunduğunun araştırılması
- Müşterilerin ihtiyaçlarını açıkça ifade etmelerine ve tamamen geri bildirim sağlamasına yardımcı olacak bir yaklaşım tanımlanması
- İhtiyaç arama esnasında müşterinin girdilerinin uyarılması, gerekliliklerde hatalar ve ihmaller açığa çıkarılması
- Geliştiricilere istisnalar üzerine erken başlamak için daha sonraki aşamalarda onlara odaklanmak yerine ortak bir temel sağlayan bir yaklaşım uygulanması
- İstisnaları erken keşfetmek ve belirlemek ve bu istisnayla nasıl başa çıkılacağınnı ve işleme konulmasının sağlanması.
- Bir vaka çalışması kullanarak önerilen yaklaşımın değerlendirilmesi.

Bu çalışmanın en büyük katkısı, yazılım geliştiricileri ve paydaşlar arasındaki iletişimi geliştiren etkileşimli bir yaklaşım tanımlayarak ve eksik gereksinim sorununu azaltan bir şekilde kullanıcı gereksinimlerinin nasıl ortaya çıkaracağının bir yolunu içeren yeni bir yaklaşım tanımlamak ve kullanıcı gereksinimlerini kesin bir şekilde üretmeye teşvik etmektir. Bu, erken safhalardaki istisnalara odaklanmayı (yani iş kurallarının ve eksik gerekliliklerin ihlal edilmesinden kaynaklanabilecek birincil anormal durumlar) hesaba katacaktır. Ayrıca, kullanıcı gereksinimlerinin içerebileceği istisnaları yakalamak için bir yazılım geliştirilmiştir. Geliştirilen program, tokenizasyon gibi bazı doğal dil kavramlarını ve bazı konuşma etiketlerini kullanır. Başka bir katkı, sınıflandırma mekanizmamıza göre istisnaların işlenmesinin modellenmesi için bir görsel aracı (UCM'ler) uyarlamaktır.

Bu çalışmanın diğer kalan bölümleri aşağıdaki gibi düzenlenmiştir:

Bölüm I de tezin tanımı yapılmıştır. SDLC'ye genel bir bakış ve sağlam yazılım sistemlerinin oluşturulmasında gereksinim mühendisliğinin önemi açıklanmıştır. Ayrıca, bu tür sistemlerin geliştiricilerini ve paydaşlarını tatmin etmek için sorunlara neden olan en önemli zorluklar gösterilmiştir. Ayrıca, tezin kapsamı, tezin amaçları ve tezin içeriğinin ana hatları açıklanmıştır.

Bölüm 2, İki ana bölümden oluşmaktadır, genel bilgi ve önceki çalışmaların bir özetidir. Öncelikle, gereksinim mühendisliği ve faaliyetlerinin genel bir arka planı sunulmuştur. Ayrıca istisnalar, istisnalar ve hatalar arasındaki farkın nedeni, istisnaların neden sistem hatalarının nedenleri olarak görülebileceği, istisnalar üzerine geliştiricilerin neler olması ve istisnaların erken tespitinin önemi üzerinde durulmuştur. İkinci kısımda, literatür araştırması verilmiştir. Bu araştırma, istisna durumlarının azaltılmasına katkıda bulunan çabaları ve son zamanlarda yapılan araştırmaları .içermektedir.

Bölüm 3, SDLC'nin erken aşamasında istisna tespiti ve ele alınması için önerilen yaklaşımı sunmaktadır. Bu yaklaşım, yazılım geliştiricilerinin ve ilgili sistemin menfaat sahiplerinin, eksik gereksinimlerin etkisini azaltılmasını sağlayacak şekilde kullanıcı gereksinimlerinin oluştutulmasına nasıl yardımcı olacağı konusunda

organize bir aşamayı içermektedir. Bu yaklaşım aynı zamanda, bir yazılım sistemi ve kullanıcısı arasındaki etkileşimden veya sistemin kendisinden kaynaklanabilecek istisnaları türetmek amacıyla net tanımlanmış kurallara göre çalışan bir program da içerir. Ayrıca, kullanıcı gereksinimlerini ve istisna modellemeyi de göstermek için görsel tanımlama aracı (UCM'ler) kullanılmıştır. Ayrıca, bu yaklaşımı göstermek için ayrıntılı bir vaka çalışması yapılmıştır.

Bölüm 4 de, bulguların ve şartların ortaya çıkartılması ve istisnaların tespiti sürecindeki çalışmaların tartışılmasının bir özeti verilmiştir. Bu çalışmalara SDLC'nin erken safhalarında, bu konulara odaklanarak istisnaların zorluklarının nasıl azaltılabileceği de dahil edilmiştir.

Bölüm 5 de tez çalışmasının sonuçları ve elde edilen sonuçların özeti verilmiştir. Ayrıca, önerilen yaklaşımın aşamaları kısaca gösterilmiş ve kendi hedeflerinin ne olduğu konusunda net bir fikir verilmiştir.

Bölüm 6 da önerilen yaklaşımdaki ayrıntıları içeren öneriler sunulmuş ve gelecekteki çalışmalar için yeni eğilimler açıklanmıştır.

2. TEORİK PRENSİPLER

2.1. Genel Bilgi

2.1.1 Yazılım süreç modelleri

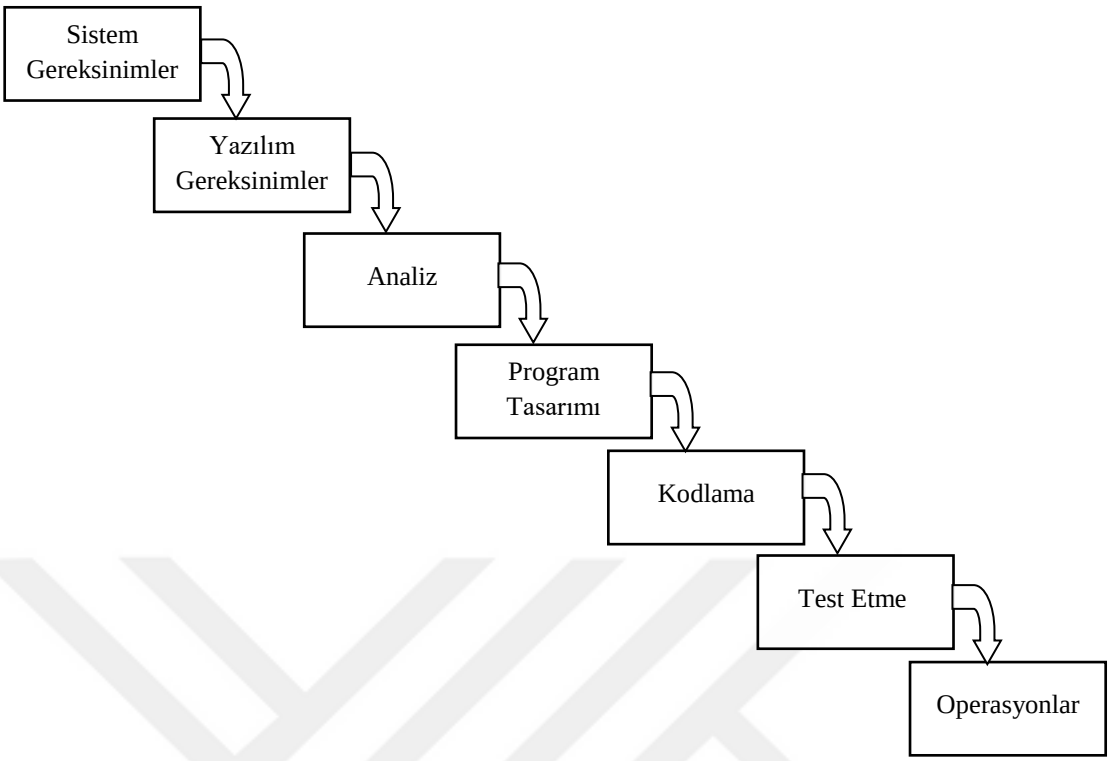
Herhangi bir yazılım sisteminin geliştirilmesi, çeşitli organizasyonel adımlar gerektirir. Paydaşlarının ihtiyaçlarını karşılayarak gerekli hedefleri elde etmeyi amaçlar. Süreç geliştirmenin çeşitli modelleri ve metodolojileri vardır. Bu modeller doğaları gereği çeşitli faktörlere göre değişir: projenin doğası, müşteri ve geliştiricilerin deneyimi.

Süreç modellerini anlamak ve sürdürmek zorlaşmaktadır; özellikle projelerin boyutunun artmasıyla uyum süreci zorlaşmaktadır (Weber, Reijers, Zugal, & Wild, 2009).

Bu modellerin her birinin avantajları ve dezavantajları vardır. Onlarla ilgili daha fazla bilgi edinmek için, bu faktörlerin bazıları özetlenmeye çalışılmıştır.

2.1.1.1. Şelale modeli

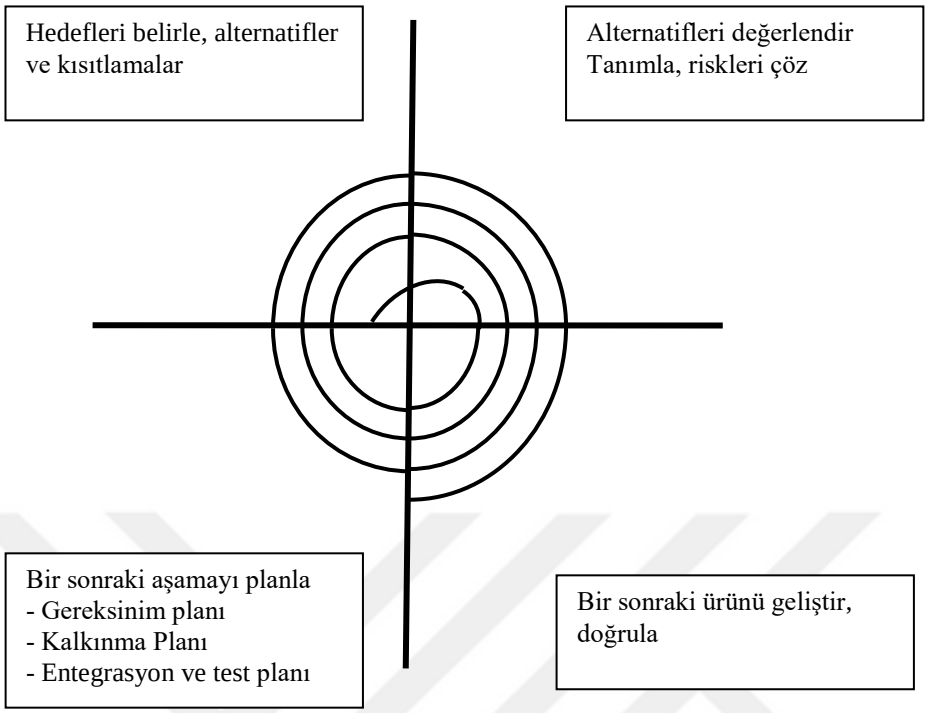
1970 yılında yazılım geliştirme sürecini organize etmek için Royce tarafından geliştirilmiştir (Şekil 2.1). Bu, klasik, doğrusal ve soyut bir modeldir, birçok gelişme aşamasından oluşur; gereksinim aşamasından başlar, test ve çalışma aşaması ile sona erer. Bu modelin doğası, bir sonraki aşamaya geçmeden önce her aşamadaki çıktıların doğru ve net olması gerektiğini varsayar. Önceki aşamaya dönmek, bu modelin maliyetli ve zaman alıcı olduğunun düşünüldüğü hipotezlerden biridir; bu genellikle paydaşların erken bir tarihte ihtiyaçlarını ifade etmediklerinden kaynaklanır. Aynı zamanda geliştirme sürecinin yeniden çalışmasına da yol açabilir ve bu modelin en önemli dezavantajları ve riskleri olarak düşünülebilir (Balaji & Murugaiyan, 2012; Guimarães ve Souza Vilela, 2005; Hijazi, Khdour, & Alarabeyyat, 2012).



Şekil 2.1 Şelale modeli (Guimarães ve Souza Vilela, 2005)

2.1.1.2. Spiral modeli

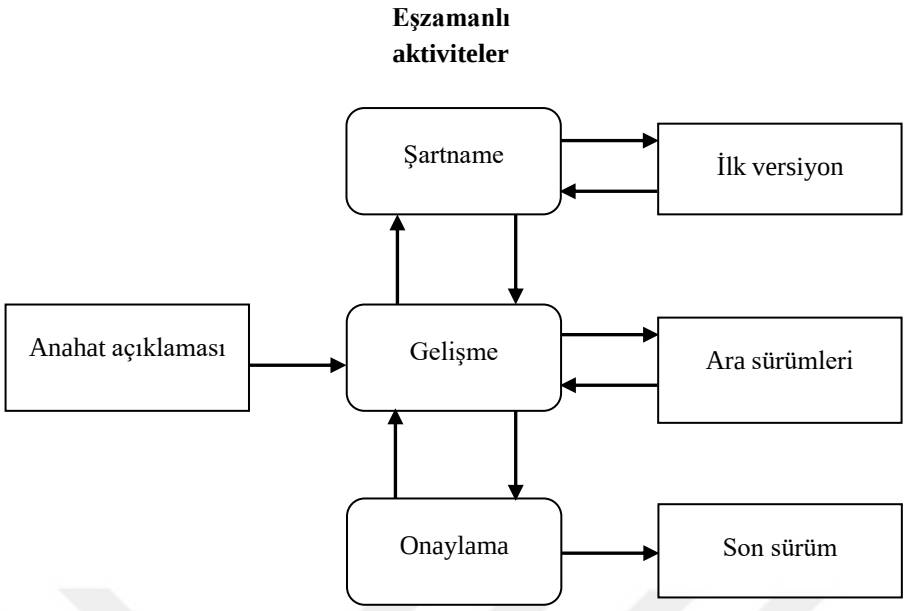
Bu model, 1988 yılında Boehm tarafından yazılım geliştirmeyi evrim yönünde organize etmek için sunulmuştur (Şekil 2.2). Şelale modelinin aşamaları her bir artışta uygulandığı ve müşterinin geribildiriminin tanımlandığı ve dikkate alındığı her artımda risk analizi ve yönetimine odaklanmaktadır. Modelin aşamalarına şunlar dahildir: (i) ihtiyaçların toplanmasına odaklanan planlama; (ii) alternatif çözümlerin risk analizi ve muafiyeti; ve (iii) nihai sürüme ulaşana kadar prototip mühendisliği ve değerlendirmesi. Her bir aşamada riskler analiz edilir ve azaltılır. Bu model risk analizi ve ele alma derecesine bağlı olduğundan, özellikle zaman ve bütçe sınırlı olduğunda, pahalı ve zaman alıcı bir model olarak kabul edilir (Munassar & Govardhan, 2010)



Şekil 2.2. Spiral modeli (Munassar & Govardhan, 2010)

2.1.1.3. Artımlı gelişim

Her bir versiyonun (artış) oluşturulması fikri, önemli müşteri ihtiyaçlarının önceliklerine odaklanmaktadır (Şekil 2.3). Bu süreç, müşterilerin ihtiyaçlarını önceden karşılayabilmelerini sağlar; ve riskin azaltılmasına katkıda bulunabilir. İş dağıtımı, bir sistemin farklı parçalarının farklı ekipler tarafından oluşturulduğu anlamına gelir. Bu, süreci düzenleyen açık bir çerçevenin yokluğunda tutarsız entegrasyona neden olabilir. Dahası, hızlı gelişmenin doğası, sistemi doğru bir şekilde belgeleme sürecini olumsuz şekilde etkiler ve aynı zamanda, müşteri taleplerinde kesintisiz değişimden kaynaklanabilecek zaman ve maliyeti de etkiler (Sommerville, 2010).



Şekil 2.3 Artan geliştirme modeli (Sommerville, 2010)

İyi bir sonuç için, bu tür modellerin bazı özelliklerinin birleştirilmesi yani gelişmiş yaklaşımların oluşturulması gereklidir.

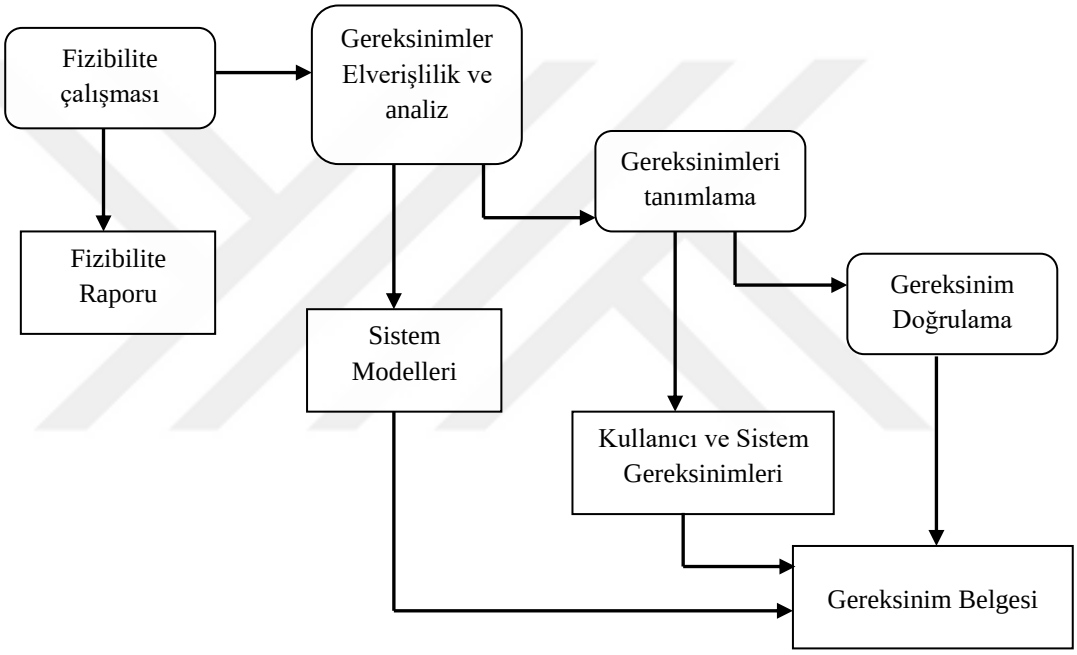
2.1.2. Gereksinim ne demektir

Hassas gereksinimler, başarılı herhangi bir yazılım sisteminin oluşturulmasında temel oluşturmaktadır. Gereksinim, yazılım gelişiminde önemli bir sözcüktür ve bunun anlamı gereksinim mühendisliği ve faaliyetleri alanında navigasyonun önce bilinmesi gerekir; özellikle gereksinimlerin ortaya çıkartılması faaliyetidir. Bu kelimenin birçok tanımları vardır ve bu çalışmada IEEE, 1990 standart sözlükte verilen tanımlar kullanılmıştır. Bu tanımlar şunlardır (Yang & Tang, 2003):

"(1) Bir kullanıcının bir sorunu çözmesi veya bir hedefe ulaşması için ihtiyaç duyduğu bir durum ya da yetenek, (2) bir sözleşmeyi, standardı, şartnameyi ya da bir sözleşmeyi yerine getirmek için bir sistem ya da sistem bileşeni tarafından yerine getirilmesi gereken bir durum yeteneği ya da diğer resmi olarak dayatılan belgeler, 1 veya 2'de belirtilen koşul veya kabiliyetin belgelenmiş bir temsili "

2.1.3. Gereksinim mühendisliği (GM)

GM, yazılım mühendisliği alanında daha kritik bir aşamadır ve yazılım geliştirme sürecinin temelini oluşturur, bir yazılım sisteminin ne yapması gerektiğine odaklanır ve belirsizlik ve hatalar içermeyen açık bir şekilde gereksinimlerin sunulmasından sorumludur. SDLC sonraki safhaların temelidir (Fu, Bastani, & Yen, 2007; Van Lamsweerde & Letier, 2000) GM, gereksinimlerin ortaya çıkartılması, analizi, şartnamesi, gerekliliklerin geçerliliği ve yönetimi olmak üzere beş faaliyetten oluşur (şekil 2.4).



Şekil 2.4 Gereksinimler Mühendisliği süreci (Sommerville, 2010)

2.1.3.1. Şartların ortaya çıkartılması

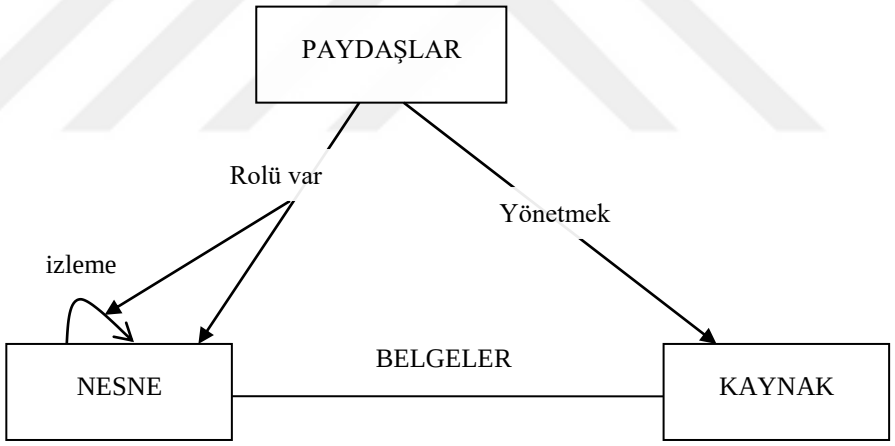
İhtiyaçların açıkça ifade edilememesinden dolayı gereksinim çıkarma, geliştirilecek yazılım sisteminin gerekliliklerini keşfetmeye odaklanan en kritik etkinlik olup paydaşların memnuniyeti üzerinde büyük etkiye sahiptir ve SDLC'nin diğer safhalarının başarısı üzerinde büyük bir etkisi vardır (Joseph; Sharma & Pandey, 2013).

Hatasız yazılım geliştirilmesi, geliştiricileri ile yüz yüze kalmak için büyük bir mücadele gerektirir. Bunun nedeni, paydaşlar ve sistem geliştiricileri arasında zayıf

bir iletişimin olmasıdır. Bu, paydaşların veya sistem kullanıcısının farklı öngörülerine sahip olduğu ve ihtiyaçlarını tam olarak ifade edemediği birçok nedenden kaynaklanır; bu, eksik veya belirsiz gerekliliklere ve daha sonra bir sistem hatasına yol açar (Joseph, Kelly, 2007; Stone & Sawyer, 2006).

Paydaşlar, bir sistemden etkileyen veya etkilenen kişilerdir (Pacheco & Garcia, 2008). Şekil 2.5, SDLC'deki paydaş rollerinin meta modelini göstermektedir (Ramesh & Jarke, 2001).

Ek olarak, ortaya çıkarma faaliyeti, yazılım geliştiricileri ile ilgili kişiler (paydaşlar / kullanıcılar) arasında ve kullanıcıların bir sistemden neye ihtiyaç duyduklarının tam olarak anlaşılabilmesi için yüksek bir işbirliği gerektirir (Pressman, 2005). Dolayısıyla, etkin iletişim, geliştiriciler ve sistem kullanıcıları arasında ortak bir anlayış ve müzakere yoluyla sağlanacaktır (Coughlan, Lycett, & Macredie, 2003).



Şekil 2.5 Paydaşların izlenebilirlik modeli (Ramesh & Jarke, 2001)

Bu şekilde, "Nesne" kelimesi, SDLC aşamalarının kavramsal girdileri ve çıktılarını (tek gereksinimler, alternatif vb.), "KAYNAK" kelimesi bilgi kaynağını, "İzler arası bağlantılar" bu nesneler arasındaki izlenebilirliği ifade etmektedir (Ramesh & Jarke, 2001). Gereksinim kaynakları çalışma belgeleri müşteriler; politikaları; kurallar ve miras sistemleri olabilir.

Eksik veya eksik taleplerin etkisi, gelişme maliyetini, yazılım geliştirme çabasını ve nihayet yazılım sistemindeki kullanıcıların memnuniyetsizliğini ve sistem arızalarını arttırmayı içerir (Pa & Zin, 2011).

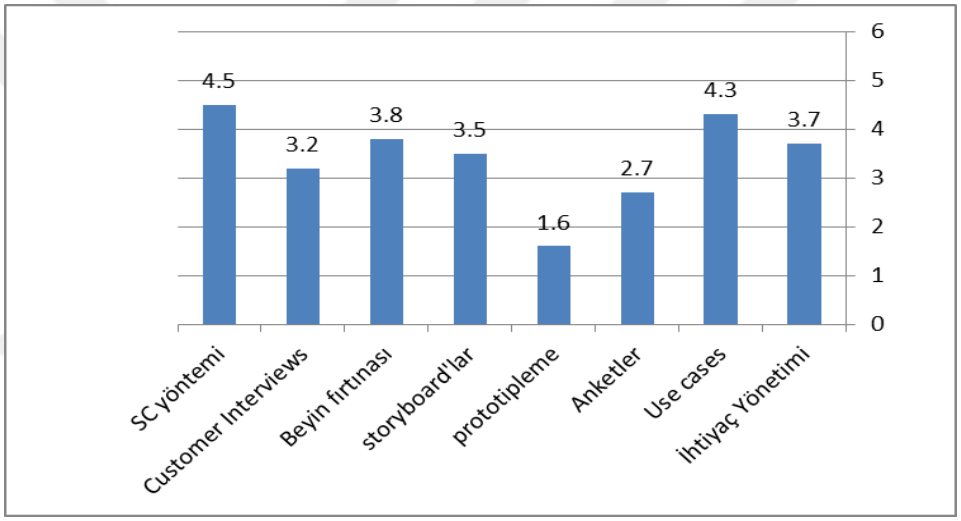
Röportaj, beyin fırtınası, prototipleme, senaryolar ve modelleme gibi yazılım gereksinimlerini ortaya çıkarmak için kullanılan çeşitli teknikler vardır (Zowghi & Coulin, 2005). Röportaj klasik bir teknik olup halen en çok kullanılan çıkarma tekniğidir ve bir geliştirici ile potansiyel kullanıcılar / paydaşlar arasında doğrudan bir etkileşimdir. Bu teknik, geleneksel ve yetersiz bir tekniktir, çünkü bir geliştiricinin kullanıcıların gereksinimlerini anlamak için sosyal becerilerin üretilmesi konusunda üst düzey becerilere sahip olması gerekir ve paydaşları kendi ihtiyaçlarını kendi sistemlerinden tanımlamaları ve pazarlık etmeleri için iyi sonuçların kazanılması geliştiricilerin deneyimine bağlıdır (Carrizo, Dieste, & Juristo, 2014; Robertson & Robertson, 2012) Yapılandırılmamış mülakatlar, şartların ortaya çıkması için her zaman uygun olmayabilir; çünkü gereksiz ayrıntılar toplanabilir, maliyet etkin olur ve oturumlarını yönetmek için üst düzey bir beceri gerektirir, oysa yapılandırılmış görüşmeler genellikle daha iyidir; çünkü önceden tanımlanmış sorulara ve kullanıcı gereksinimlerini anlamayı amaçlayan sorgulara dayalı organize bir oturumda çalışmaktadır; Bu, acemi geliştiriciler için daha faydalı olmaktadır (Robertson & Robertson, 2012; Sharma & Pandey, 2013)

Prototip, sistem kullanıcılarının geribildirimlerinin elde edilmesi ve ihtiyaçlarını ifade etme yeteneklerini geliştirmesi için yardımcı olabilecekleri gibi bu ihtiyaçların derin bir şekilde anlaşılması ve geliştiriciler için de kullanılabilecek bir başka tekniktir (Iqbal & Suaib, 2014).

Senaryoya dayalı teknik, ihtiyaçların ortaya çıkartılması için kullanılır ve çalışma süreçlerini gösteren bir kullanıcı hikayesinin tanımından oluşur (Pa & Zin, 2011). Bu teknik yeterli olmayabilir ve tamamlanmamış senaryolara neden olabilir; çünkü kullanıcılar, çalışma hikayelerini eksiksiz bir biçimde açıklayamayabilirler. Bu, tamamlanmamış kullanıcı gereksinimlerinin yakalanacağı anlamına gelir

Beyin fırtınası tekniği, sistem geliştiricileri ve ilgili paydaşlar arasındaki etkileşimli bir tartışmada olabildiğince çok fikir üretmek üzere tasarlanmış gayri resmi tartışma tekniğidir. İki aşamadan oluşur: (1) Nesil Aşaması. Herhangi bir eleştiri olmaksızın fikir üretmeye odaklanır ve (2) konsolidasyon aşamasında bu fikirlerin filtrelenmesine odaklanır; bazı fikirlerin birleştiği yerlerde, bazıları rafine edilir ve bazıları göz ardı edilir. Bu avantajlara rağmen, önemli konuları çözmek için uygun bir teknik olarak düşünülemez (Sharma & Pandey, 2013).

Lloyd, Rosson ve Arthur (2002), hangi çıkarma tekniğinin daha etkili olduğunu belirlemek için bir araştırma yapmıştır. Şekil 2.6 da bu çalışmanın sonucu verilmiştir.



Şekil 2.6 Elifasyon tekniklerinin etkinliği (Lloyd ve diğ., 2002)

Eleme yönteminin seçilmesi, yazılım gereksinimleri üretiminin kalitesine etki edecek ve sonunda bir yazılım sisteminin başarısına katkıda bulunacaktır (Carrizo et al., 2014).

Bununla birlikte, bir sistem büyüdükçe temel ikilem arttığı için, gereksinim belirleme ve değişikliklerin kontrolü zorlaşmaktadır. Dolayısıyla, bu tezin amaçlarından biri olarak kabul edilen, ihtiyaçların karşılanması için sistematik bir yaklaşım gerekmektedir.

2.1.3.1.1. Kullanıcı katılımının faydaları

SDLC sırasında kullanıcı katılımının birçok nedeni (Bano & Zowghi, 2015) aşağıda verilmiştir:

- geliştirme sürecini basitleştirir,
- Sistem ihtiyaçlarını karşılar,
- Geliştirme ekibine güvenlerini arttırır,
- Gereksinim değişikliği taleplerini azaltır.

2.1.3.1.2 Sorunları ortaya çıkarma

Gereksinimleri ortaya çıkarma, yazılım geliştirme için en önemli faaliyettir. Bir yazılım sistemininde ne yapılacağını anlamak için yazılım geliştiricileri ve ilgili paydaşlar arasında işbirliği olmalıdır. Tutarsız, belirsiz, eksik ve gereksiz gereksinimler yetersiz teknik özelliklere ve daha sonra başarısız bir yazılım sistemine yol açacağından kolay bir süreç değildir.

Davey ve Parker (2015), sistem hatalarının çoğunun yetersiz sistem belirtiminden kaynaklandığını göstermiştir. Belirsizlik, yetersiz gereksinimlere yol açar ve onarımı için daha fazla zaman gerektirir. Belirsizlik, birden fazla yoruma (anlamı) sahip tek bir ifade anlamına gelir. Haftanın sorun alanının haftalık olarak anlaşılması, gereksinimlerin ortaya çıkmasından ve yazılım geliştiricileri ile kullanıcıları arasındaki iletişim yetersizliğinden kaynaklanabilir (U.Sahah ve Jinwala, 2015). Benzer bir anlamda, belirsiz gereklsinim vakit kaybetmek anlamına gelir ve gereksinim belirsiz ve yanlış anlaşılmıştır (Glinz & Fricker, 2015).

Hastie ve Wojewoda (2015), 50.000 projenin analizine ilişkin "CHAOS Raporu" nun şu sonuçlara vardığını belirt: zamanında ve bütçe dahilinde tamamlanan başarılı projeler % 29 iken, % 52' sine itiraz edilmiş ve% 19 başarısız olmuştur.

Huzooree ve Ramdoo (2015), Şekil 2.7'de gösterildiği gibi, altı yazılım projesinin araştırılması ile ilgili olarak anketten elde edilen eleme sorununu bildirmişlerdir.

Şekil 2.7 Bir yazılım projesi örneği ile ilgili sorunlar (Huzooree & Ramdoo, 2015)

Eksik gereksinimler, projenin maliyetlerini ve gelişim sürelerini aşmasını sağlayan kritik bir konu olarak kalmaya devam etmektedir. Bir proje kapsamında gereksinimlerin ortaya çıkarılması ve analizler ve gereksiz ihtiyaçların toplanması önemlidir (Sutcliffe & Sawyer, 2013).

Eksik gereksinimler, projenin maliyetlerini ve gelişim sürelerini aşmasını sağlayan kritik bir konu olarak kalmaya devam etmektedir. Bu durum, bir projenin kapsamı ve gereksiz gereksinimlerin toplanması ile ilgili talep ve analiz gereksinimlerinin karşılanması gerçeğine dayanmaktadır.

Ayrıca, yetersiz gereksinimlerin ortaya çıkması, bu tür bir sorun yüzünden başarısız olan projelerin % 90'ında sistem hatalarına neden olan büyük bir sorundur (Davis, Fuller, Tremblay, & Berndt, 2006). Bu nedenle eksik gereksinimler, istenmeyen durumlara yol açan riskler olarak sınıflandırılabilir ve bunların etkileri tanımlanmalı ve değerlendirilmelidir (Cailliau & van Lamsweerde, 2014a)

Yazılım geliştiricileri ile potansiyel kullanıcılar arasındaki iletişimin zayıf olması, yazılımın arızalanmasına neden olan diğer bir zorluktur. 85 yayın içeren bir çalışmada, eksik taleplerin % 20'sinin müşterilerin istek değişkenliğinden ve % 40'nin eksik belge belgelerinden (Berenbach, 2006) kaynaklandığı belirtilmiştir.

Gereksinimlerdeki tutarsızlık başka bir kritik konudur. Menfaat sahiplerinin görüşleri, gereklilik değişiklikleri vb. çatışmadan veya çelişkiden kaynaklanabilir (Hadar & Zamansky, 2015).

Yukarıdaki sorunlara ek olarak, eksiklik gereksinimleri, istisnai durumların öngörülmemesi nedeniyle ortaya çıkan bir diğer önemli sorundur (Alrajeh, Kramer, Van Lamsweerde, Russo ve Uchitel, 2012; Fricker, Grau ve Zwingli, 2015).

2.1.3.2. Gereksinim analizi

Gereksinimlerin ortaya çıkarılmasından sonraki etkinlik, gereksinim analizidir. Gereksinim analizi, belirsizliklerin araştırılmasına, gereksinimlerin organizasyonuna bütünlük; ve çelişkilere odaklanır (Pressman, 2005).

Menfaat sahipleri farklı perspektiflere sahip olabilir, bu nedenle sistem paydaşları ve yazılım mühendisi arasındaki müzakere riskleri çözmek için önemlidir; Burada ortaya çıkarma ve analizde uygun bir teknik kullanılarak bazı gereklilikler gereksiz olduğundan kaldırılır, bazıları değiştirilir ve bazıları birleştirilebilir (Pressman, 2005).

"Gereksinimlerin "ortaya çıkışı, dokümantasyonu ve müzakere" süreci tekrarlamalı olarak yapılır ve iyi gereksinimlerin ortaya çıkması için temeli oluşturur. Bu nedenle çözülmemiş sorunların çözümünde etkili olacağı açıktır (Pohl, 2010).

2.1.3.3 Gereksinim Şartnamesi

Bir sistemin işlevselliğini ve performansını organize bir yapıda gösteren, toplanmış ve ortaya çıkartılan gereksinimleri belgelemek önemli bir faaliyettir ve yazılım geliştirme sürecinin temelini oluşturur ve bu dokümantasyon yazılı doküman gibi farklı formatlara sahip olabilir; matematiksel modeller ve benzeri; veya bunların birleşimi ve "doğal dil ile grafik modellerin kombinasyonu büyük sistemler için en iyi yaklaşım olabilir" (Pressman, 2005).

2.1.3.4. Gereksinim doğrulaması

Gereksinim Şartnamesi'nin geçerliliği, gereksinim mühendisliği aşamasında bir başka önemli faaliyettir. Sistem gereksinimlerinin paydaşları ihtiyaçlarını karşılamasını sağlamaya odaklanır. Bu aktivite esnasında aşağıdaki özellikler incelenir (Pressman, 2005):

- Belirsizlikler
- Tutarlılık
- Eksiksiz
- Fizibil
- Gerekli ve diğerleri.

Ayrıca, gereksinimdeki hatalar, SDLC'nin alt-dizin aşamalarında olumsuz bir etki yaratacaktır, bu nedenle yazılım geliştiricileri ve paydaşları arasındaki sözleşmenin temelini oluşturduğu düşünülürse, uygun dokümantasyon gereklidir (Pohl, 2010).

2.1.3.5.Gereksinim yönetimi

Bu etkinliğin rolü bir sistemin SDLC'nin tüm aşamaları boyunca izlenebilmesi ve gereksinimlerinin zaman içinde izlenebilirliğine odaklanması ve bu nedenle de değişim ve yapılandırma yönetiminin bu faaliyetin amacı olmasıdır (Pohl, 2010). Bu değişiklikleri ispatlamadan önce maliyetlerin ve emek harcamalarının değişimi tahmin edilmelidir, bu nedenle ilgili paydaşlarla daha fazla müzakere edilmesi gerekebilir (Pandey, Suman, & Ramani, 2010).

Buna ek olarak, bu etkinlik sırasında; yeni gereksinimlerin ortaya çıkması veya önceki gereksinimlerin güncellenmesi gerekebilir.

2.1.4. İstisnalar arka plan

2.1.4.1 İstisnalar tanımı

Bir istisna işleme hatasını nelerin oluşturduğuna dair bir tanım yaygın şekilde kabul görmemektedir (Ebert, Castor, & Serebrenik, 2015). Literatürde istisnaların birçok

tanımları vardır. İstisna normal bir senaryoyu temsil etmeyen bir yol akışıdır (Castor Filho, Guerra, Pagano ve Rubira, 2005; Dijkman, van IJzendoorn, Türetken, & de Vries, 2015; N Russell & van der Aalst, 2006).

Chen ve diğ. (2015) istisnaları "bir adımın uygulanmasının anormal sonucu " olarak tanımlamıştır. Cailliau ve van Lamsweerde (2014a) istisnaları engel olarak görmüşler ve engellerin, analiz edilmesi ve çözülmesi gereken riskler olduğunu ifade etmişlerdir.

Kayıp ve eksik gereksinimler, sistem hatalarına neden olan kritik konulardan biridir (Cailliau & van Lamsweerde, 2014a). Örneğin, UML, kullanım örnekleri adında popüler bir mekanizma sunmaktadır. Bu mekanizma yazılım sistemlerinin davranışsal gereksinimlerini keşfetmek ve kaydetmek için yaygın biçimde kullanılan biçimciliktir (Shui, Mustafiz, Kienzle, & Dony, 2005). Bu bir sistemin ve mevcut aktörlerin gereksinimlerini karşılamak için potansiyel aktörler arasındaki etkileşimleri tanımlayan hikayeleri temsil eder.

Buna ek olarak, istisnalar atıldığında, bu işlemin normal dizisinin sonlandırıldığı anlamına gelir ve erken keşfedilmemiş (yani gereksinimlerin toplanması sırasında) bulunmayan açıklanan istisnai durum, tamamlanmamış bir sistem belirtimine yol açacaktır. Kullanım örnekleri başlıca başarılı senaryoları içerir ve aktörleri hedeflerine ulaşmalarına yönlendiren alternatif senaryolar içerebilir. Bir kullanım durumu kesilir ve amacına ulaşamıyorsa (ör. Anormal durum) bu durum bir istisna olarak sınıflandırılabilir.

Yazılım geliştirme aşamaları arasındaki izlenebilirlik haritalama gerektirir. Bu nedenle, gereksinimler evresi herhangi bir yazılım projesinin başarılı bir şekilde geliştirilmesinin temel dayanağıdır.

(Romanovsky, 2007) tespit edilen istisnaların çoğunun daha sonra gereksiz gereksinimlerden kaynaklandığı sonucuna varmıştır. Bu nedenle erken aşamadaki iyi odaklanma, uygulama aşamasındaki istisnaları asgariye indirgemekte ve gelişim süreci üzerinde güçlü bir olumsuz etkisi olmayabilir. Bu gereksinim düzeyindeki istisnaların, uygulama seviyesindeki istisnalarla karıştırılmaması gerektiği anlamına

gelir. Bununla birlikte, uygulama aşamasında istisnaların tespit edilmesine yönelik rolü ortadan kaldırılmaması gerekir, ancak negatif etkinin en aza indirgendiğini ve bu nedenle de daha kolay ele alınabileceğini vurgulamışlardır.

Bilinen istisna örneklerinden biri Ariane 5'in ilk uçuşunda kaza olası nedeni olup uçuş sırasında ortaya çıkabilecek anormal durumları çözecek olan taşıma mekanizmasının bulunmaması ile ilişkilidir (Bruntink, Van Deursen, & Tourwé, 2006).

Hatalar ve istisna arasındaki fark, istisnalar çözülebilirken hataların çözülemiyeceğidir. İstisnalar iki şekilde sınıflandırılabilir: Kontrol edilen ve kontrol edilmemiş istisnalar. Kontrol edilen istisnalar, bir derleyicinin yakalayabileceği istisnalardır ve denetlenmeyen istisnalar bir derleyici tarafından yakalanamaz (örneğin çalışma zamanı istisnaları) ve dolaylı olarak çoğaltılamaz (ör., İşlenmemiş istisna) (VFD Dooren & Steegmans, 2005). Bir "FileNotFoundException" birçok programlama dilinde tanımlanan kontrol edilmiş istisnalara bir örnektir. Bu istisna, bir programın çalışma zamanında uzak bir dosya konumundan veri okumayı gerektirdiği ve bu dosyanın mevcut olmadığı durumlarda algılanır. Öte yandan, kullanıcının neden olduğu istisnalar da yakalanmamış istisna olarak kabul edilebilir.

Yukarıdakilerden istisna işlemenin zor bir görev olduğunu ve bu nedenle geliştiricilerin çoğunun SDLC'nin erken safhalarında bunu önlemeye çalıştıkları söylenebilir Bu yazılımın daha başarısız olmasına neden olacaktır. Bu nedenle, bu istisnaların erken bir aşamada ele alınması zaman ve çaba tasarrufuna büyük ölçüde yol açabilir.

2.1.4.2. İstisnalar başarısızlığa neden olabilir

İstisnalar, yazılım sisteminden diğerine farklıdır. İstisna kullanımı bir sistemin güvenilir olduğunu gösteren önemli faktörlerden biridir. Örneğin Ariana 5'in ilk uçuşunda meydana gelebilecek kazanın olası nedeni uçuş sırasında ortaya çıkabilecek anormal durumları ele alacak olan taşıma mekanizmasının bulunmamasıyla ilgilidir (Bruntink ve diğerleri, 2006). Bununla birlikte, birçok farklı nedenden ötürü istisnalar olabilir, bazıları eksik gereksinimler, kullanıcı hataları ve

bazıları programcıların hataları veya donanım hataları ve kaynak yetersizliği gibi fiziksel hatalardır (Cailliau & van Lamsweerde, 2014b).

İstisna kullanımı çoğunlukla SDLC'nin sonraki aşamasında gerçekleştirilir, bu bazı önemli bilgiler kaybolabildiğinden verimli olmayabilir ve bu konuyla ilgilenmek daha sonra daha fazla zaman ve emek gerektirir ve kurtarma eylemleri daha pahalı olur (Van Lamsweerde & Letier, 2000)

Dahası, özellikle kritik sistemler için uygun bir istisna işleminin yapılmaması, gelişim süreci boyunca dikkate alınmayan durumlar için sistemin başarısızlığa uğramasına neden olacaktır (Hecht, 2008).

2.1.4.3. İhlal istisnaları

SDLC'nin erken aşamalarındaki istisnai durumlara odaklanmak zor bir görev olarak kabul edilir ve birçok yazılım geliştiricisi bunları göz ardı eder. Bir araştırmada 154 geliştirici, örgütlerin ve geliştiricilerin% 73'ünün politikalarındaki istisnai durumlara odaklanmadığını gösterir (Ebert & Castor, 2013).

İstisnalarla uğraşmanın ihmalinin ardındaki başlıca nedenler, aşağıdaki noktalarda özetlenebilir (Harrold, 2010; H. Shah, Görg, & Harrold, 2008):

- Bazı geliştiriciler hata ayıklama amacıyla istisnalarla uğraşmaya dayanır; bu da istisnalar oluştuğunda onları düzeltir;
- Bazı geliştiriciler zaman kaybına uğradığını varsayar (yani zaman harcamaya değmez);
- Bazı geliştiriciler kullanılan bir programlama dili tarafından empoze edilen şeyleri takip eder ve onların işleme sürecini dil gereksinimlerine dayandırır; ve
- Diğer geliştiriciler ana işlevlerinin bir parçasını oluşturmadığını varsayar.

Yukarıdakilere ek olarak ve farklı geliştirme aşamalarındaki istisnalarla ilgilenme konusundaki sürekli ihmali vurgulamak için, modern programlama dillerinde farklı uzmanlığa sahip birçok geliştiricinin görüşlerini araştıran bir anket hazırlanmıştır. Bu

inceleme, istisnalarla Program yazma aşamasıdır. Bu husus istisnalar üzerinde vurgu yapılması gereken sahnedeki görüşlerin dalgalanmasına ek olarak halen devam etmektedir. Bu nedenle, geliştiricilere bu ikilemi hesaba katmak için kılavuz yaklaşımın gerekliliği çok önemlidir; bu, çalışmamızın ana hedeflerinden biridir

2.1.4.4. Beklenen ve beklenmedik istisnalar

İstisnalar çoğu çalışma sisteminde önemli sorunlardır ve bu istisnaları işleme koymak için daha fazla zaman ve çaba gerektirir. Normal durumlar (başarılı yollar), herhangi bir iş sapması olmadığı anlamına gelir ve istenen hedefe ulaşılır. Beklenen istisnalar, beklenmedik durumla ilgili bilgi anlamına gelir ve işleme süreci tanımlanır. Tersine bir istisna beklenmedik olduğunda, istisna ile ilgili önceden bilinmeyen bir bilginin bilinmesi anlamına gelir ve böyle bir durumun dikkate alınmaması, sistemin düzeltme gayretini ve maliyetini artırır (N. Russell, W. van der Aalst, & A. ter Hofstede, 2006a; N. Russell, WM van der Aalst, & AH Ter Hofstede, 2006b)

Önemli istisna tiplerinin, gösterilen türlere dayandığına inandıklarını belirtmişlerdir (Nick Russell ve diğerleri, 2006a):

- İş öğesi hatası: bir işlemi tamamlayamıyor.
- Son teslim tarihi: Bir zaman aşımı veya bir çalışma öğesi, gerekli sürede tamamlanmadı.
- Kaynak yokluğu: bir çalışma öğesi bir veri kaynağına erişemez.
- Harici tetikleyici: Harici bir olay, bir işlem kesintisine neden olur.
- Kısıtlama ihlali.

Bu nedenle, bu durumları SDLC'nin erken aşamalarında değerlendirmek için bir yaklaşıma ihtiyaç vardır.

2.1.4.5. İstisna yayılımı

Yayılma, uygulamanın kontrol akışı istisnanın işleyicisine taşınacağı anlamına gelir. İki türde atma ve devam ettirme yayılımları sınıflandırılabilir. Döküm yayılımında yürütme akışının kontrolü, istisnanın oluştuğu noktaya dönmeyecektir. Bununla

birlikte, yürütme denetimi, özgeçmiş çoğaltma türünün istisnanın yükseltilmiş noktasına dönecektir; bu tür programlama dili tarafından desteklenmelidir (P.A. Buhr & Mok, 2000). Dolayısıyla, gereksinimlerin değiştirilmesi, bağımlı olduğu diğer koşullarda bir değişikliğe neden olabilir. Bu nedenle, istisnalar olabilir ve bunların yayılımı beklenir

Dahası, hataların ters yönde yayılması olabilir. Örneğin gereksinimin başlangıçta dayandığı daha önceki gereksinimleri olumsuz olarak etkileyen kesintiye ve tamamlanmamış işleme yol açan bağımlı bir gereksinimde hata ortaya çıkabilir. Bu nedenle bu feshin etkilenmesini önlemek ve uygulananların gerekli güncellemelerini veya çekilmelerini sağlamak için, aynı serideki önceki gereksinimlerin bildirilmesi gerekir (diğer bir deyişle bir istisna meydana gelir).

Bu durumu açıklamak, görsel bir görüş vermek ve bu sürecin anlaşılmasını kolaylaştırmak için UCMs adlı görsel aracın görsel bir bakış açısı önerilmiştir.

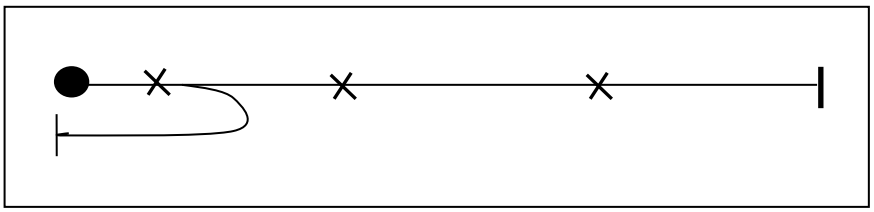
UCM'ler, Buhr ve Casselman tarafından, nedensel bir yol (lar) da yüksek seviyede bir sistemi temsil etmek için kullanılan görsel bir açıklama aracıdır (R. J. Buhr & Casselman, 1995). Her yol, uzun bir yolun sorumluluklarının sayısından ve bu sorumluluklar arasındaki nedensel ilişkiden oluşur. Her senaryo bir objektif veya bir görevi göstermek için sorumluluklar biçimindeki bileşenler arasındaki bir etkileşimi temsil eder.

Herhangi bir sistemi bileşenleri aracılığıyla görsel olarak canlandırmak, hedeflerin anlaşılmasını kolaylaştırır.

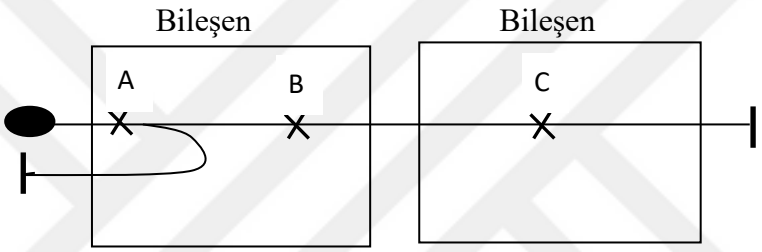
Her bileşenin sorumlulukları (kalın harf "X" ile gösterilir) vardır ve bu bileşenleri ilişkilendiren bir yol, bu sorumlulukların akışını ve bir objektif gerçekleştirmek veya bir görevi gerçekleştirmek için bu bileşenler arasındaki işbirliğini temsil eder.

Bu yazılımı basit bir şekilde açıklamak için, Şekil 2.8, temel bir bağlanmamış yol senaryosu gösterimi ve Şekil 2.9, iki bileşene bağlı UCM'nin basit bir örneği verilmiştir.

Buna ek olarak, her yol senaryosu bir başlangıç noktasına, senaryonun bir önkoşulunu (siyah dolu bir daire ile temsil edilir) son nokta da senaryonun (bir katı çubukla gösterilen) son durumunu temsil eder. UCM bileşenlerin adını açıklamadan kullanılabilir. Bu UCM, Unbound UCM olarak kabul edilir (Amyot ve diğerleri, 2000).

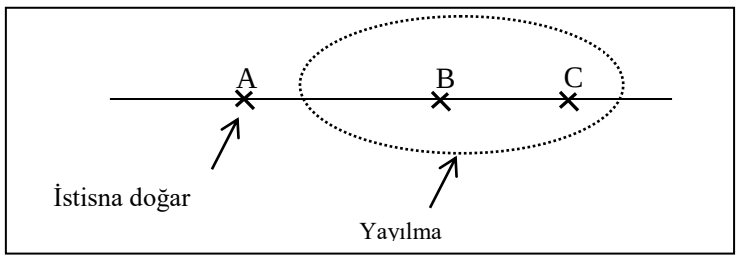


Şekil 2.8. Bağsız UCM'ye basit bir örnek



Şekil 2.9. Bağlı UCM'ye basit bir örnek

Tek bir kullanıcı gereksiniminde ileri istisna propagasyonu örneği şekil 2.10'da gösterilmiştir; "A" sorumluluğunda bir istisna olduğunda, bu istisna "B" sorumluluğuna, daha sonra "C" sorumluluğuna geçirilir; "B" veya "C" sorumlulukları bu istisnayı almazsa, aksi durumda istisna işlenmeyen soruna neden olur. Bu istisna türü, ileriye yönelik bir istisna olarak adlandırılabilir çünkü anlaşılıp ele alınmazsa, sonraki sorumlulukları olumsuz bir şekilde etkiler ve bu nedenle gereksinime bağlı daha sonraki kullanıcı gereksinimlerine etkileri olur.

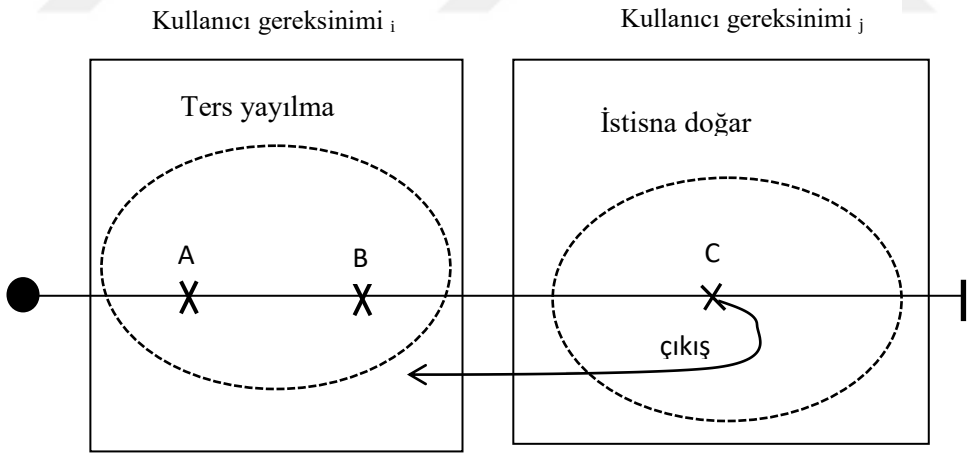


Şekil 2.10. İstisnanın ileri iletimi örneği

Öte yandan, tek bir kullanıcı gereksinimi içindeki sorumluluklardan birinde veya başka bir bağımlı kullanıcı gereksiniminin sorumluluğunda bir istisna ortaya çıkabilir

ve dolayısıyla sorumlulukları üzerinde önceliği etkileyebilir. Bu istisna ters istisna olarak adlandırılabilir; çünkü aynı zamanda önceki sorumlulukların izini de gerektirir. Dolayısıyla, bu istisnanın ortaya çıkışı, ister aynı gereksinim içinde isterse bu gereksinimin bağımlı olduğu önceki gereksinimde olsun, bu sorumluluğu takip etmeyi gerektirir.

Birden fazla kullanıcı gereksinimi olması durumunda istisna yayılımına bir örnek, Şekil 2.11'de gösterilmiştir; "C" sorumluluğunda istisnai bir hata ortaya çıkar. Bu, sürecin askıya alınmasına ve sonuç olarak, tamamlanamamasına yol açmıştır. Bu, bazı güncellemeleri veya geri yüklemeyi gerektirebilecek önceki sorumluluklarda dikkate alınmalıdır (bu durumda, işlem "A" veya "B" ya da her ikisinde de sorumluluklar içerebilir). Bu, bazı gereksinimlerin belirli bir hedef veya işlevi tamamlamak için birbiriyle bağımlı olduğunu açıklar. Bu nedenle istisna yakalanmadan gerekli işlem yapılırsa, bu daha sonraki bir aşamada tespit edilmesi durumunda çözülmesi güç bir probleme neden olur. Noktalı elips, istisnanın görünümünden etkilenen alanı gösterir (Şekil 2.11).



Şekil 2.11. İstisnanın Ters Dağılımına Bir Örnek

Bundan dolayı, dışlama noktası, gerekli ekleme işlemini kabul etmek için geliştirme ekibi ile birlikte geliştirilecek sistem kullanıcıları arasında görüşmeler yapılmasını gerektirebilir.

2.2. LİTERATÜR İNCELEMESİ

2.2.1. Geliştiriciler ve istisnalarla ilgili görüşleri

İstisnalar üzerine odaklanma konusunda birçok görüş var. Önceki çalışmaların birçoğunda gösterildiği gibi, istisnalar genellikle hata ayıklama / sınama aşamasında ve ayrıca sistem geliştiricileri tarafından kullanılan programlama diline göre düşünülür. Bu tamamlanmamış kabul edilebilir ve gelecekte yazılımın başarısını olumsuz yönde etkileyebilir; çünkü SDLC'nin önceki aşamalarını ihmal eden istisnalar olabilir. Buradan itibaren, bu konseptleri, eğilimleri ve gelişim sürecinin diğer aşamalarındaki istisnaların yükünü azaltmak gösterilen çabaları vurgulamak için araştırma yapılmıştır.

Örneğin, geliştirme sonraki aşamaları üzerinde odaklanma açısından, kod testinde ve özellikle de hata ayıklama döneminde istisnalarla uğraşan pek çok geliştirici, Schröter, Bettenburg ve Premraj (2010), geliştiricilerin kodlarını hata ayıklama yoluyla test etmelerine yardımcı olmak için yığın izlemenin rolünü incelediler ve çalışma hata raporlarının hata tespitinde faydalı olacağını gösterdiler. Kod testi, istisnaların testinin dahil edildiği anlamına gelmez ve ayrıca, hata ayıklama aktivitesi konusunda çalışmalar yeterli değildir çünkü SDLC aşamalardan bazılarının yeniden işlenmesini gerektirebilir.

Sinha, Orso ve Harrold (2004), bu yönüyle sunulan bir başka çalışmada (Kod testi), yazılım geliştiricilerin kodlarını test etmeleri ve bakımları için "prosedürel kontrol akış grafiği (ICFG)" yoluyla istisnaları analiz ederek otomatik bir yazılım sunmuşlardır, daha sonra tüm "throw" ve "catch" işlem blokları analiz edilir ve bu "*throw*" ve "*catch*" blokları arasındaki aralık, döngüsel grafikler kullanılarak hesaplanır. Bu araç test analizine odaklanır ve yine de oldukça soyut ve istisnaların test edilmesinin içerildiğine dair gösterge bulunur, ancak bu analiz yöntemi kullanıcının ihtiyaçlarını nasıl karşılayabileceğini gösterir.

Robillard ve Murphy (1999), istisnaların oluşabileceği olası noktaları çıkarmak için kullanılan kod anlama (java dilinde programlama) için bir istisna işleme geliştirme aracını sunmuşlar ve istisna işlemlerinin hangi yerlerde iyi olabileceğini

incelemişlerdir. İstisnalar akışı ile ilgili çıkarılan ayrıntılar yetersiz olabilir ve istisnaların nasıl test edilebileceğini açıklamıdığını rapor etmişlerdir.

Cornu, Seinturier ve Monperrus (2015), java programlama dilinde yazılan kod için "try-catch (Deneyin-Yakalamak)" bloklarında istisnalar enjeksiyonu için kısa devre test algoritması olarak adlandırılan bir yaklaşım geliştirmişlerdir; beklenmedik istisnalar söz konusu olduğunda test edilecek ve analiz edilecek kodun esnekliğini iyileştirmek için iki deneme anlaşması tanımladılar ve bu sözleşmelerin ihlal edilmesinin, ortalama istisnaların ortaya çıkmasına neden olduğunu varsaymışlardır. Benzer yaklaşım, kusur enjeksiyonlarını iyileştirmek için Natella, Cotroneo, Duraes ve Madeira (2013) tarafından sunulan başka bir test mekanizması olarak hata enjeksiyonlarını kullandılar, bir yazılımın içerdiği gerçek hatalarla, bazı enjekte edilen hataları simüle ettiler. Değerlendirme, enjekte edilen arızaların keşfedilmesi halinde, bu arızaların tespit edilmesi kolay olduğu anlamına gelir. Aksi halde zor ve tüm yazılımın arızalarının daha fazla denetlenmesi gerekebilir. Bunlar, hata ayıklama etkinliği testi için tamamlayıcı araçlar olarak kabul edilebilir; eğer bir sistem açık ve sağlam gereksinimler üzerine kurulmuşsa; Daha sonra bu araçlar test sürecini destekleyecektir.

Bazı programlama dilleri, geliştiricilerin bu dillerde yazılmış programlarda istisnalar kullanmalarını gerektirir; bu da, geliştirilecek programın gerekliliklerinin açık ve eksiksiz bir tanımı olursa iyi olur.

Ayrıca, bazı programlama dilleri, geliştiricilerin bu dillerde yazılmış programlardaki istisnaları kullanmasını gerektirir. Örneğin, Cacho ve diğ. (2014), Java ve C # programlama dillerindeki istisna işlemlerini karşılaştırmış ve Java'nın istisna türlerine daha fazla odaklanmaya ihtiyaç duyduğunu göstermektedir. Bu nedenle, kullanıcı tarafından tanımlanan istisnalar birçok programlama dilinin kullanımını artırabilir. Bu, SDLC'nin erken evrelerinden istisna alınması üzerine odaklanılarak geliştirilebilir.

Zayıf yazılım gereksinimlerinden kaynaklanan istisnalar, tasarım sorunları veya programcılar, istisnalarla ilgili değerlendirmelerde bulunmazlar. Cabral ve Marques

(2008) aşağıdaki istisnalarla yazılım istisnalarını işleme talebi için bir model önermişlerdir: (1) Try-Catch ifadesi, istisnalara karşı korunan bir blok olarak programcılar tarafından beyan edilmelidir, (2) bir sistemin çalışma zamanı yakalamak veya istisnaları atma durumunda kurtarma eylemlerini tanımlamalıdır. Mükemmel bir yazılım gereksinimi üretme konusundaki yoğunlaşma bu sorunu azaltmak için gereklidir.

Castor Filho ve ark. (2005), geleneksel programlama dillerinin bu istisna işleme mekanizmasını sağlamadığı yerlerde, birçok nesne programlama dilinin istisnaları işleme mekanizmaları sağladığını rapor etmişlerdir. Bu mekanizmalar uygulama aşamalarının bir parçasıdır ve bu dillerin gelişmesine rağmen, istisnaların erken aşamada ihmal edilmesi, yüksek kaliteli yazılım oluşturmanın önünde bir engel teşkil etmektedir. Konsantrasyon tasarım ve uygulama aşamalarında olduğu ve SDLC'nin erken safhasındaki istisnai durumlara yoğunlaşmayı ihmal ettiği için bu yazılım sistemlerinin daha sonra güncellenmesi veya güçlendirilmesi için çok çaba gerektirebilir.

Yukarıdakilere ilaveten, birçok geliştiricinin istisnalarla uğraşmak açısından farklı görüşleri vardır, H. Shah ve ark. (2008), istisnalarla ilgili görüşlerini değerlendiren dokuz Java geliştiricisini içeren bir çalışma sunmuştur. Bu geliştiriciler, başvurularında istisna işleme konusundaki düşüncelerini anlamaları ve değerlendirmeleri için sorular sorup çalışmanın sonucu iki noktada özetlemişlerdir: (1) Katılımcıların çoğu kullandıkları programlama dili tarafından zorlanmadıkları sürece, görevin bir parçası olarak istisna ile ilgilenmeyi düşünmez ve (2) Çalışma, istisnaları SDLC'nin tüm aşamaları boyunca izlemek için "İstisna Mühendisi" ile SDLC'ye yeni rol verilmelidir.

Mao ve Lu (2005), kontrol akış grafiğini kullanarak kod analizi için bir prototip aracı önermişler ve yazılım sağlamlığını artırmak için bazı test durumlarında tüm kod yollarını kaydetmek için istatistiksel bir yapı tanımlamışlardır. Bu yaklaşımın daha fazla iyileştirmeye ihtiyaç duyduğunu ve geliştiricilerin bir katılımcının programın çalışmasını tamamlamak ve desteklemek için gerekli olduğunu özetlemişlerdir. Bu

yaklaşım, önceden tanımlanmış bazı test durumlarına dayanan kod testi için tamamlayıcı bir yaklaşım olarak düşünülebilir.

Hwang ve Tang (2004) "olay koşul eylemi" kuralı, kural tanımına (kapsam (bir süreç veya alt süreç, olay (istisna türü), durum (önerme)) dayalı bir fikir kullanarak istisnalar için bir mimari model önermişlerdir. Bir olay meydana gelir, ilgili istisna işleme ile ilişkili koşullar kontrol edilir) ve herhangi bir kural meydana gelen olayla karşılaşırsa, istisna işleme (eylemler) çağrılır aksi takdirde benzer önceki kural (yani benzer kapsamı içerir) kontrol edilir önceden tanımlanmış "istisna tutucu veritabanı" ve var ise, bazı değişiklikler içeren eylemleri alternatif bir işleme süreci olarak sunulabilir.Bu değişiklikler veya önceden tanımlanmış kurallara dahil olmayan beklenmedik olaylar, ortaya çıkabilecek aktivitenin pahalı olabileceği şekilde yeniden işlenmesini gerektirebilir.

İstisnalar, onları ele almak için gereken zaman ve çaba bakımından biraz korkutucu olur, bu nedenle bazı rehberlik noktalarının araştırılması, etkilerini azaltmak için önemli olabilir. Ferreira, Takai, Malkowski ve Pu (2010), istisnaların karmaşıklığını azaltmak için sunduğu rehber yaklaşımı üç adımdan oluşur: (1) Kritik yollara neden olan ana olaylar belirlenmeli, (2) Bu olayların tanımı ve ele alınması mekanizmaları tanımlanmalıdır (yani durumlar, her olayın durumu (koşulları) ve geçiş olayları tanımlanmalıdır) ve (3) iş akışları, olaylar ve veri akışı tetikleyicileri, unsurları ve olay durumlarını izlemek üzere tanımlanmalıdır. Bu tür yaklaşımlar gereklidir, ancak soyut kalırlar.

Xie ve Thummalapenta (2012) istisnaları tanımlamanın zorluklarını incelemişlerdir, Çalışma beklenen istisnaların ele alınmasına odaklanmıştır ve işleme süreci koşullu (Küçük Durumlar) veya istisna (Büyük Durumlar) olarak sınıflandırılmıştır. İşlemenin formal şartname ve ardından ilişki kurallarının ve koşullu modellerin analizi yoluyla gerçekleşebileceğini varsaymışlardır. Hangi durumların büyük ve hangilerinin küçük olduğunu belirlemek hâlâ soyutlama eğilimindedir; ancak resmi şartnamenin iyileştirilmesi daha sonra kod verimliliğini azaltacaktır.

İstisnaları göz ardı etmek kritik durumlara veya sistem hatalarına neden olabilir. Leopold ve diğ. (2012), özellikle süreçlerin işlenme süresi üzerinde kötü istisna işlemesine yol açan sorunları incelemişlerdir ve süreç modelinde dikkate alınmayan istisnaların, süreç modelinde yer alan istisnalardan daha uzun süreye sahip olduğuna karar vermişlerdir. Bu, kuruluşların politikalarının, yazılım geliştiricilerinin, gerekli kurtarma prosedürlerini hazırlayarak, beklenen istisnaları işleme koyacak özellikte odaklanmalarına izin veren bu düşünceyi göz önünde bulundurmasını sağlar. Bu kavram, organize bir süreçte benimsenmek istenilen konsepte çok yakındır.

de Lemos ve Romanovsky (2001), istisnaların sahne ile ilgili istisnalar olarak görülebileceğini (yani SDLC'de şartlar, tasarım ve uygulama aşamaları), ve aynı zamanda bu aşamalar arasındaki izlenebilirliğin gerekli olduğunu desteklediklerini rapor etmişlerdir.

Özellikle giderek artan yazılım boyutu ve karmaşıklığı ile, istisnalara önem vermemek yani önemsememek acil bir sorundur ve göz ardı edilmemesi gereken önemli bir ikilem olarak görülmelidir. Bu görüşten yola çıkarak, bazı araştırmacılar istisnalarla uğraşmanın yalnızca hata ayıklama veya sınama aşamasına bağlı olmamasına inanmaktadır. Kienzle (2008) SDLC'deki istisnalarla ilgili daha genel bir araştırma sunmuştur ve temel sonuç, istisnaların ele alınmasının her aşama ile ayrı ayrı ilişkili olması ve tüm yaşam döngüsü boyunca bu durumların belgelenmesi gerekir. Bu soyut bir öneri olarak algılanabilir ve sistematik bir yaklaşıma ihtiyaç duyulmaktadır

Bendraou ve diğ. (2010) analiz ve istisnaların işlenmesi süreci için bazı soyut desenler önermiş, bu modeller önerilen sorulara dayanmaktadır: "Sorunu derhal çözebilir miyiz? Sürecin sunması gereken başka bir alternatif var mı? Bu girdiyi tamamen reddedilecek mi?" tanımlanan model çerçevesinin temelini oluşturan bu öneri, istisnalara odaklanmanın önemli bir görev olduğu ve ihmal edilmemesi gerektiği sonucuna varan çalışma tecrübelerine dayanmaktadır. Bu bizi, gelişimin ilk aşamalarında İstisnaları hafifletmek ve ele almak için bir mekanizma kurmaya teşvik etme etkisine sahip olmuştur.

Sousa, Schilling ve Furtado (2007) yazılım geliştiricilerinin gelişmekte olan projelerinde istisna işleme konusuna odaklanmalarını desteklemek amacıyla yazılım kalitesinin önemini araştırmış ve istisna işleme kuralları için bir beyan edici şartname dili tanımlamışlardır. Barbosa, Garcia ve Mezini (2012), yazılım geliştiricilerin istisna işlemesine (ör. Istisna noktaları) ihtiyaç duyan kod parçalarını bulmasına yardımcı olmak için bir öneri sistemi oluşturmanın temelini oluşturan sezgisel stratejiler önermişlerdir. Bunlar, hata ayıklama işlemi sırasında yapılan bir iş olarak görülebilir.

Önceki çalışmalardan, istisnaları ele alma mekanizmalarının, geliştiricilerin deneyimlerine ve yazılımlarının daha sonra karşılaşabileceği istisnai durumların ciddiyetini anlamalarına bağlı olduğu açıktır. Birçoğu, istisnalara odaklanmanın kod oluşturma aşamasında olduğuna inanıyor. Modern programlama dillerinin çoğunda program yazarken istisnaları yakalama süreci vardır; bazıları ise, odağın tasarım aşamasında veya kod test aşamasında olduğunu hissetmiştir. Sonuç olarak, istisnalarla uğraşma sürecinde ve olması gereken aşamada ihmal vardır.

Kuruluş aşamasındaki yoğunlaşma, geri kalan aşamaların başarısının anahtarıdır. Daha sonra ortaya çıkabilecek anormal durumlar, gereksinimlerin kesin olarak tanımlanamamasından veya sistemle kötü etkileşimden kaynaklanabilir.

2.2.2. Eksik gereksinimler

Bütünlük, belirsizlik ve doğruluk açısından gerekliliklerin toplanması ve incelenmesi süreci, bu gereksinimlerin karşılaması gereken hedeflere yol açan derin bir anlayışa ulaşmayı amaçlayan yinelemeli bir süreçtir.

Bu nedenle, geliştirme süreci kolay bir iş değildir. Sistem gereksinimlerinin derinlemesine anlaşılmasını ve sistemin paydaşlarının ihtiyaçlarını anlamak için yeterli çaba gerektirmektedir. Böylece, sistem geliştiricileri ve paydaşlar arasındaki müzakere seviyesi, istenen hedeflere ulaşmak için önemli bir süreç olarak düşünülmelidir. Bu yüzden, süreç modellerinin iyileştirilmesi ve yeni teknolojilerin ortaya çıkması, gelişim sürecini iyileştirmeye katkıda bulunabilir.

Süreç modelleri, yazılım geliştirme süreci için önemlidir, çünkü yol haritasını düzenler. Yapılan çalışmaları her birinin amacı ve kullanım şartları vardır. Letsholo, Chioasca ve Zhao (2012), bazı süreç modelleri arasındaki boşlukları kapatmak için bir model sınıflandırma çerçevesi önermişlerdir. Bu boşluklar, iş hedeflerinin ve kaynakların belirlenmesi gibi önemli iş kavramlarının ihmalinden kaynaklanmaktadır. Bu çerçeve aynı zamanda geliştirme sürecinin herhangi bir seviyesinde istisnalarla uğraşmayı da ihmal etmektedir. Bilindiği gibi, genellikle projenin niteliğine ve geliştiricilerin deneyimine göre seçilen birçok süreç modeli vardır, Sampaio (2012) "iş süreci modelleme (BPM)" nin yeteneklerini değerlendirmek için bir çalışma sunmuştur ve sonuç olarak bu tür modellerin çoğu hala soyut, yetersiz ve aynı zamanda bir çok iş konseptini içermemektedir (örneğin iş hedefleri, davranışlar, vb.). Recker (2010) tarafından sunulan benzer bir araştırma, BPM'nin iyileştirmelere ihtiyaç duyduğu ve bu modellerde çalışmayı geliştirmek için gerekli destek araçlarına sahip olduğu sonucuna varmıştır. Bu nedenle, bu modeller soyuttur ve ikilem, gereksinimlerin ortaya çıkarılması ve istisnaların tespiti mekanizmasında giderek daha karmaşık hale gelir.

Gereksinimleri anlamak, birçok geliştiricinin yüz yüze kaldığı zorluktur çünkü birçok paydaş projelerinin başında ihtiyaçlarını belirleyememektedir. Damian, Chisan, Vaidyanathasamy ve Pal (2005), iyi gereksininimlerilerin etkisi hakkında bir araştırma çalışması sunmuştur ve diğer gelişme aşamalarının güçlenmesini artıran sistem gerçeklerini anlamada takım memnuniyetini arttırmanın önemli olduğunu rapor etmişlerdir.

Damian ve Chisan (2006) mühendisliğin etkili gereksinimlerinin olumlu ve önemli rolünü göstermek için ampirik bir çalışma gerçekleştirdiler. Sonuçlarını desteklediğimiz gereksinim mühendisliğinin proje başarısı ve kalitesi üzerinde önemli bir etkisi olduğunu rapor etmişlerdir. Bu, güçlü gereksinimlerin inşasının daha sonra beklenmedik istisnaları azaltmaya katkıda bulunacağına olan güveni artıracaktır

Kullanıcı gereksinimlerinin ortaya çıkması, bu kullanıcıların ihtiyaçlarını karşılamakta, böylece geliştirilecek sistemin istenen hedeflerine ulaşmada önemli bir rol oynamaktadır ve bu nedenle, bu talepleri gerçekleştirmek için birçok çaba sarf

edilmektedir. Rajagopal, Lee, Ahlswede, Chiang ve Karolak (2005), eleme sürecini geliştirmek için, paydaşların yazılım ve geliştiricilerin sınırlamalarını anlamalarını iyileştirmek için eğitimi, ihtiyaçların netleştirmesi için fonksiyonel davranış anahtar kelimelerini tanımlama ve kaydetmede paydaşlara yardımcı olma, paydaşların ihtiyaçlarının yanlış anlaşılmasını azaltmak için kalite işlevsel dağıtım tekniğini (QFD) uygulanması gibi birkaç adımdan oluşan bir yaklaşım önermişlerdir. Bu yaklaşım hala soyut olup gereksinimlerin ortaya çıkarma sürecinin nasıl yürütüldüğünü ifade etmemektedir.

Gereksinimleri çıkarmak için birçok teknik kullanılabilir. En sık kullanılan tekniklerden biri prototipleme modelidir. Vijayan ve Raju (2011), gereksinimleri ortaya çıkarmak için bir kağıt prototipi yaklaşımı önermişlerdir ve potansiyel paydaşların geri bildirimleri ile yaklaşımın etkili olduğunu düşünmüşlerdir. Bu yaklaşım, paydaşların deneyimini gerektirir ve bu, anormal durumları düşünmek için yeterli olmayabilir.

Herhangi bir yazılım projesinin inşası aşamaları birbirine bağlıdır ve daha erken bir aşamadaki herhangi bir etki, gelişim sürecinin sonraki aşamalarını olumsuz şekilde etkiler. Kukkanen, Väkeväinen, Kauppinen ve Uusitalo (2009), gereksinim mühendisliği ve test aşamalarının iyileştirilmesi hakkında öğrenilen dersler hakkında bir araştırma çalışması sunmuşlar ve bir gereksinim ve testin eşzamanlı olarak yapılmasının yazılım kalitesini artıracağı, riskleri azaltacağı ve müşteri ve geliştiricilerin memnuniyetlerini arttıracağını belirtmişlerdir. Bu tezde, birçok özel durumu erken yakalamayı dikkate alan kullanıcı gereksinimlerini katalitik bir şekilde toplamaya yönelik bir yaklaşım getirilmiştir.

Tamamlanmamış işlem demek sürecin bir kısmında birtakım hatalar veya belirsizlikler olduğunu anlamına gelir. Bu belki de belirsizlikten, yanlış anlaşılmadan veya işlemin tamamlanması için gerekliliklerin tanımlanamamasından dolayı olabilir. Örneğin, belirli bir hastanın tedavisinde eksiklik, hastanın durumunun bozulmasına ve sağlıkta olumsuz bir etkiye neden olabilir. Henneman ve ark. (2007), hasta bakım görevinde yetersiz eğitim ve politikaların zayıflığı nedeniyle her yıl bir dizi insanın eksik tedavi nedeniyle öldüğünü belirtmiştir ve kan transfüzyonu tedavisindeki

hatalar gibi tıbbi hataları azaltmak için resmi bir süreç tanımının gerekli olduğunu doğrulamışlardır. Kan nakli tedavisinde hatalar gibi tıbbi hataları azaltmak ve kusursuz bir sistem oluşturmak için çalışma sürecinin net bir şekilde tanımlanmasının gereklidir.

Liu, Li ve Xiao (2007), iş süreci yürütme dilini (BPEL) genişletmek amacıyla ECA kuralını kullanan bir deklarasyon yaklaşımı sunmuşlardır. Amaç, güvenilirlik derecesini arttırmanın yanı sıra, hata tespitlerinin de şansını arttırmaktır. Bunun, yaklaşımla çözülemediği için kurallarla ele alınamıyan sorunları azaltan kesin gereksinimlere ulaşması gerekir.

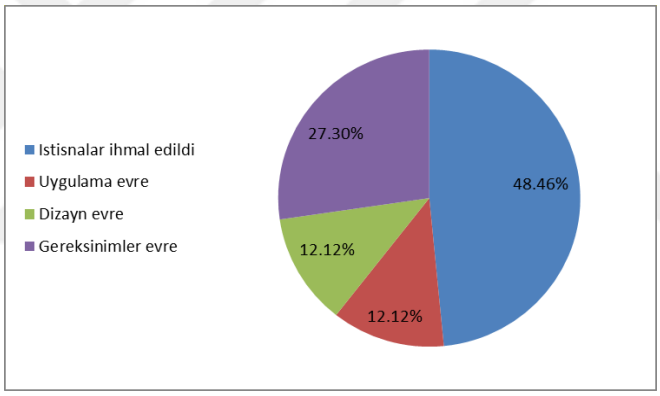
Yazılımın sağlamlığı, yazılımın başarılı olabilmesi için önemli bir faktördür ve gereksinim sağlamlığı, sonraki aşamaların tabanının sağlam olmasıdır. Shahrokni ve Feldt (2013) sağlamlık açısından yazılım gereksinimleri üzerine yapılan çalışmanın hala sınırlı olduğunu ve özellikle yazılım gereksinim düzeyinde yazılım sağlamlığını geliştiren sistematik bir yaklaşıma ihtiyaç olduğunu göstermiştir ve çalışmaların çoğunun tasarım ve test aşamalarında odaklandığını özetlemiştir. Bu öneri üzerinde kesinlikle hemfikiriz ve çalışmamızın hedeflerinden biridir.

(Mourão ve Antunes, 2005) beklenmedik istisnalarla başa çıkmayı destekleyecek işbirlikçi bir çerçeve önermiş ve bu tür istisnaların, bu durumlar hakkında gerekli bilgilerin eksikliğinden dolayı otomatik olarak ele alınamayacağı sonucuna varmışlardır ve çerçevenin bu istisnalarla nasıl baş edileceğine dair bir kılavuz olarak kullanılması tavsiye edilmiştir. Son olarak, kullanıcıların beklenmeyen istisnalar oluştuğunda yapılandırılmamış süreçlere geçebileceğini varsaymışlardır. Kullanıcıların katılımı yazılım geliştirmede önemlidir, ancak bu yaklaşım etkili bir geliştirme yönetimi olmaksızın bu durumlarda en uygun çözümleri tanımlamak için etkili olmayabilir.

3. YÖNTEM

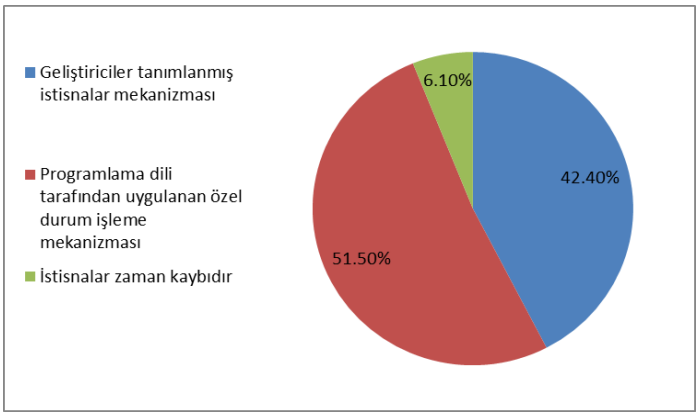
3.1. Geliştiricilerin Yazılım Geliştirmedeki İstisna İşlemleriyle İlgili Görüşleri

Daha ayrıntılı bir inceleme ile istisnalarla uğraşırken farklı deneyimlere sahip birçok geliştiricinin vizyonunu gösteren gelişimin hangi aşamada odaklanması gerektiğini gösteren bir çalışma sunulmuştur. Çalışma, hedef geliştiricilerin istisnalarla uğraştığı aşamayla ilgili önemli bir anketin cevabını içerir. Ve eğer istisnaları ele alırsa, hangi temelde ele alınıyor? İhmal mümkünse, bunun bir sonucu mu var, çünkü istisnalarla uğraşmak zor bir görevdir. Bu çalışmanın sonuçları Şekil 3.1, 3.2 ve 3.3'te gösterilmiştir (Hagal et al., 2017).



Şekil 3.1. Katılımcılar geribildirim ve istisna işleme

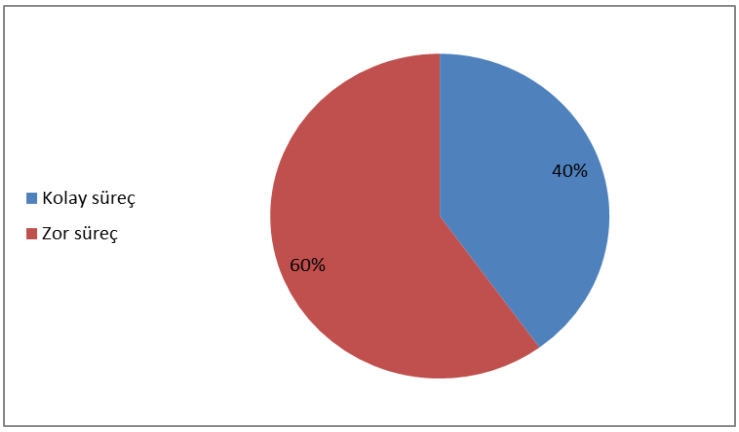
Farklı gelişme aşamalarında istisnaları göz ardı etmek, istisnaların ele alınmasını ihmal etmeye dayalı bir ikilemle karşı karşıya kalırsa, herhangi bir yazılım sisteminin başarısının devamı için bir engel olacaktır.



Şekil 3.2. Yazılım geliştiricilerinin istisnalarla ilgilenmesi

İstisnalarla uğraşmanın doğası, geliştiricilerin deneyimlerine ve vizyonlarına bağlıdır. Geliştiriciler, doğru ve daha kısa sürede çalışan bir sistem üretmekle ilgileniyorlar. Bu nedenle, sistem gereksinimlerini analiz etme sürecine yeterince dikkat edilmemesi, istisnalarla uğraşmak da dahil olmak üzere birçok öğenin ihlaline yol açabilir.

Buna ek olarak, düşüncelerin programlama dillerinin daha ilerisini düşünülemez (ör. Istisnalarla uğraşmak gibi) ve bu nedenle gözlemlenmesi gereken istisnalar üreten tüm olayları göz önünde bulundurmayabilir, yalnızca geliştiricilere uyulması gereken başka bir geliştirici kategorisinin görüşüne göre kullandıkları programlama dili, ancak istisnaların kendi yollarıyla tanımlanmasını tercih eder ve bu nedenle, bu açıdan programlama dilinin dayattığı kısıtlamalarla birleştirilirse daha iyi olabilir.



Şekil 3.3. İstisna işleme görevi ne kadar kolay

İstisnaları geliştirme sürecine dahil etmek için gerekli olabilecek çabadan dolayı istisnalarla uğraşmak zor bir süreçtir. Bu nedenle, istisnalarla başa çıkma sürecini kolaylaştıracak bir mekanizma ile geliştiricilerin teşvik edilmesi, bu ikilemin basitleştirilmesine ve böylece benimsenmesine katkıda bulunabilir.

3.2. İstisnaların sınıflandırma aşaması

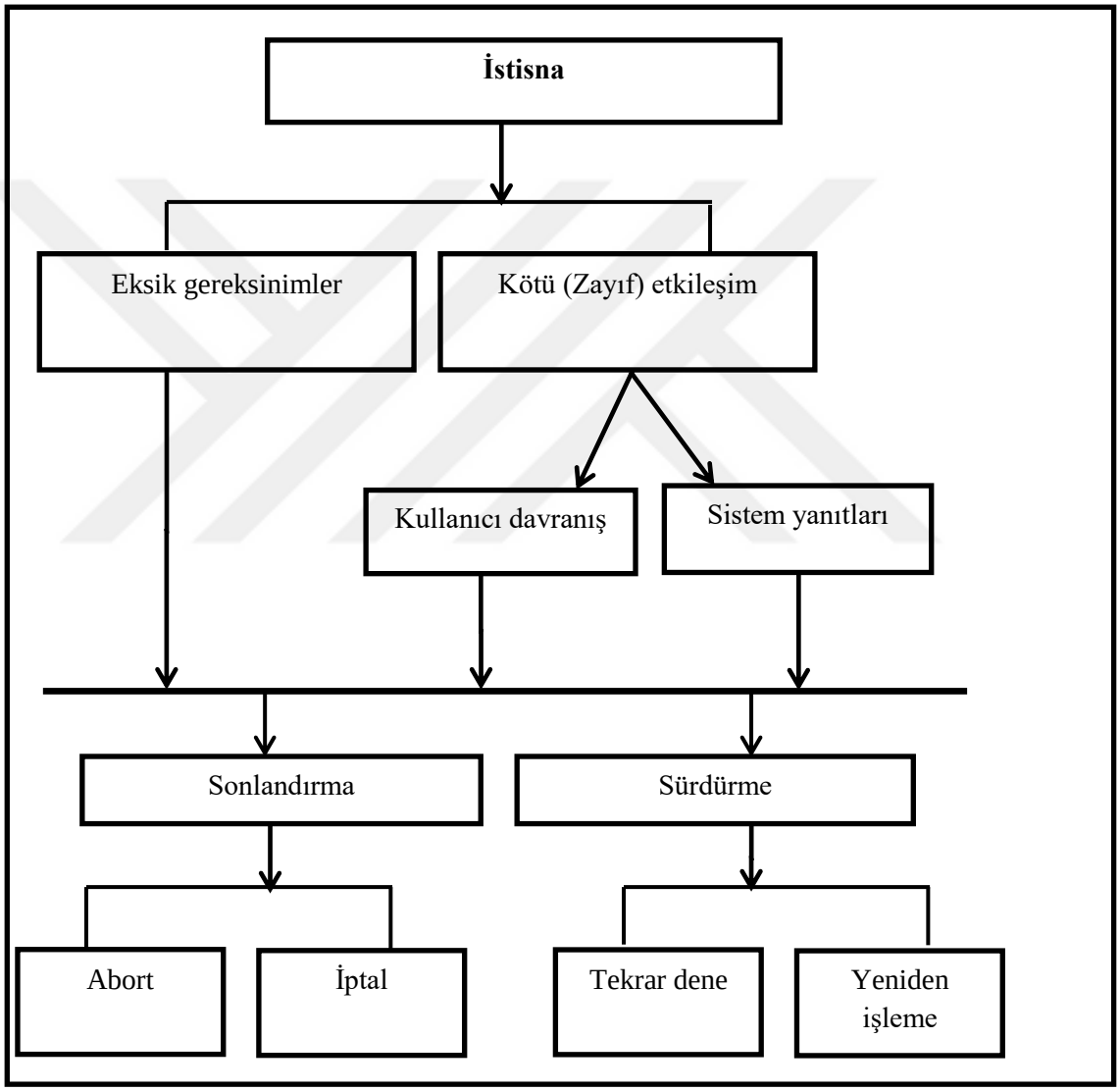
Bu aşamanın hedefleri; muhtemel potansiyel istisnaları anlamak ve durumları kullanmaya istisna oluşturabilecek durumları tanımak ve tüm olası istisnai durumları ortaya çıkarmak ve kaydetmektir. Bu sınıflandırmalar istisnaları sınıflandırmanın temelini oluşturacaktır (Şekil 3.4). Sonraki bölümlerde bu mekanizma daha ayrıntılı bir şekilde açıklanmıştır.

Bir sistemin sağlamlığı üzerinde etkili olan en önemli faktörlerden biri, sistemde hataların (veya istisnaların) bulunmamasıdır

Bu aşamada, her bir sistemin karşılaşabileceği istisnaların genel bir sınıflandırmasını tanımlamak için mekanizmada kullanılabilecek bir fikir önerilmiştir. Bu aşamada, genel istisna durumlarına aşina olmak için daha fazla kullanım senaryoları çalışması yer almaktadır.

Eğitim durumunun kullanım rolü ve istisnalar durumları tamamlayıcıdır; bunlardan biri geliştirilen sistem ile potansiyel kullanıcısı arasındaki gerekli tüm etkileşimleri ve diğeri de istisnalara neden olan durumları yakalamak ve anlamaktır.

Bu nedenle, potansiyel istisnaların çoğunu keşfetmeye yönelik ilk doğru adım, bu durumları anlayıp bunları sınıflandırmaktır. Bu sınıflandırma, her durum senaryosunun içerebileceği olası potansiyel istisnaları yakalamayı kolaylaştıracaktır. İstisnalar nedenlerine göre sınıflandırılır: Eksik gereksinimler ve zayıf etkileşim. Zayıf bir etkileşim, istisnaların neden olduğu iki sınıfta kategorize edilir: (1) Kötü kullanıcı davranışı, (2) Kötü sistem yanıtları.



Şekil 3.4. İstisnaların sınıflandırmasina kavramsal bakış

Yukarıdaki şekle ek olarak, istisnaların iki nedeni, geçişlerin iki ana durumuna yol açabilir: Sonlanma ve Devam Eden Durumlar. Bu durumlar normal bir süreçten anormal süreç durumuna geçişi ifade eder. Alt durumlar: Abort, İptal, Tekrar dene ve Yeniden İşleme büyük vakaların detaylarıdır (Sonlandırma ve Sürdürme).

3.2.1. İstisna sebeplerinden ötürü eksik gereksinimler

Hangi yazılım sisteminin olması gerektiğini anlamak, sistemlerin başarısızlıkları veya başarıları üzerindeki ana faaliyeti etkiler. Bazı hatalar, eksik gereksinimler nedeniyle sonraki aşamalarda ortaya çıkar ve bu, geliştirme çabalarını artıracaktır ve aynı zamanda bir sistemin başarısız olmasına neden olabilir, çünkü düzeltmenin daha fazla zamana ve çabaya ihtiyacı vardır.

Eksik gereksinimler, alanın kötü tanımlanması veya bir yazılım projesinin kapsamı ve sistem paydaşları ile kötü iletişim gibi birçok nedenden kaynaklanabilir. Ayrıca, kullanıcı gereksinimlerinin açık bir tanımının daha kritik bir adım olduğu ve paydaşların memnuniyetinin, sistemin ihtiyaçlarını karşılama dereceesine dayandığı bir Ön Sınıflandırma adımı olarak da düşünülebilir.

Bu sorunun üstesinden gelmek ve proaktif bir adım olarak, bu etkiyi azaltan kullanıcı gereksinimlerini üretmek için organize bir yaklaşım önerilmiştir.

3.2.2. İstisna sebebi olarak zayıf etkileşim

Bu istisna türü sistem kullanıcılarının davranışlarından (yani bir sistemin kullanılması) veya sistemin kendisinden kaynaklanır. Aşağıdaki alt bölümlerde (3.2.2.1 ve 3.2.2.2) kısaca bu istisna türleri açıklanmaktadır, bununla birlikte Kısım 3.4'te algılama mekanizması daha ayrıntılı olarak gösterilmiştir.

3.2.2.1. Kullanıcı davranışı istisnası

Bu tür bir istisna, bir sistemin zayıf bir etkileşiminin neden olduğu etkileşim istisnalarının bir alt türüdür. Bir kullanıcının yanlış bir eylem gerçekleştirdiği yerde optimal etkileşimden sapmadır.

Bir kullanıcı herhangi bir yerde ilerleyemezse, bu durum bir istisnaya neden olabilir. Kullanıcı hatalarını analiz etmek, SDLC'nin sonraki aşamalarında tespit edildiğinde yeniden işleme çabasını azaltabilir. Bu gibi hataları derinden incelemek, geliştiricilerin ve kullanıcıların bir sistemle olan yanlış etkileşiminden kaynaklanan

olumsuz yan etkileri önlemek için işleme süreçlerine odaklanmalarını teşvik edecektir. Bu tür istisnalar genellikle sistem hizmetlerinin yanlış anlaşılması ve sistemin amaçlarını gerçekleştirmek için nasıl kullanıldığı sonucunda ortaya çıkar.

3.2.2.2. Sistem yanıtları istisnası

Bir sistem, gerçekleştirilmesi gereken bazı işlemlerde yanıt alınması için anormal durumlarla karşılaşırsa bu durumlar istisnalara neden olur. Bu tür bir istisna örneği, bir sistem bir veri tabanında bilgi depolamaya çalıştığı ve veri tabanının kullanılamadığı durumdur. Bu örnekteki istisnayı işlemek için, bir kullanıcıya durdurma veya durumu düzeltmek için bazı işlemler yapma şansı verilir

3.2.2.3. İstisnai durumlar

Bazı senaryonun devamı bir nedenle iptal veya iptal gerektirdiğinde, istisna oluşur ve Yapılan sınıflandırma ile tanımlanan durumlardan birine geçişte çalışma durma noktasına gelir.

Bu nedenle, vakalar iki ana duruma ayrılır, istisnalar: Fesih ve Devam. Bu durumlar Abort, Iptal, Tekrar dene ve Yeniden işleme olarak adlandırılan dört alt duruma ayrılmıştır. Yeniden işleme durumu, süreci devam ettirmek için daha fazla eyleme ihtiyaç olabileceği anlamına gelir. Yeniden çalışma durumu, süreci devam ettirmek için daha fazla işlem yapılması gerekebileceği anlamına gelir.

Abort (anormal sonlanma) durumları, tüm işlemin tamamlanmasından önce sona erdirme anlamına gelir, İptal durumları ise, olayın ertelenmesi anlamına gelir (planlanan etkinlik gerçekleşmez), örneğin başlamadan önce seyahat iptali gibi. İptal ve İptal anahtar kelimelerinin anlamı, WordNet sözcüksel ingilizce veritabanı sözlüğünden alınmıştır. WordNet bir başka NLTK corpus okuyucusudur ve iso639 dil kodlarını kullanarak açık çok dilli WordNet'e erişim sağlar (Gillam, Tariq, & Ahmad, 2005).

"Tekrar deneme" ve "Tekrar işleme" durumları, "Devam Eden" durumun bir alt durumudur ve bu hatayı düzeltmek için bir şans verilmesi gereken bir hata

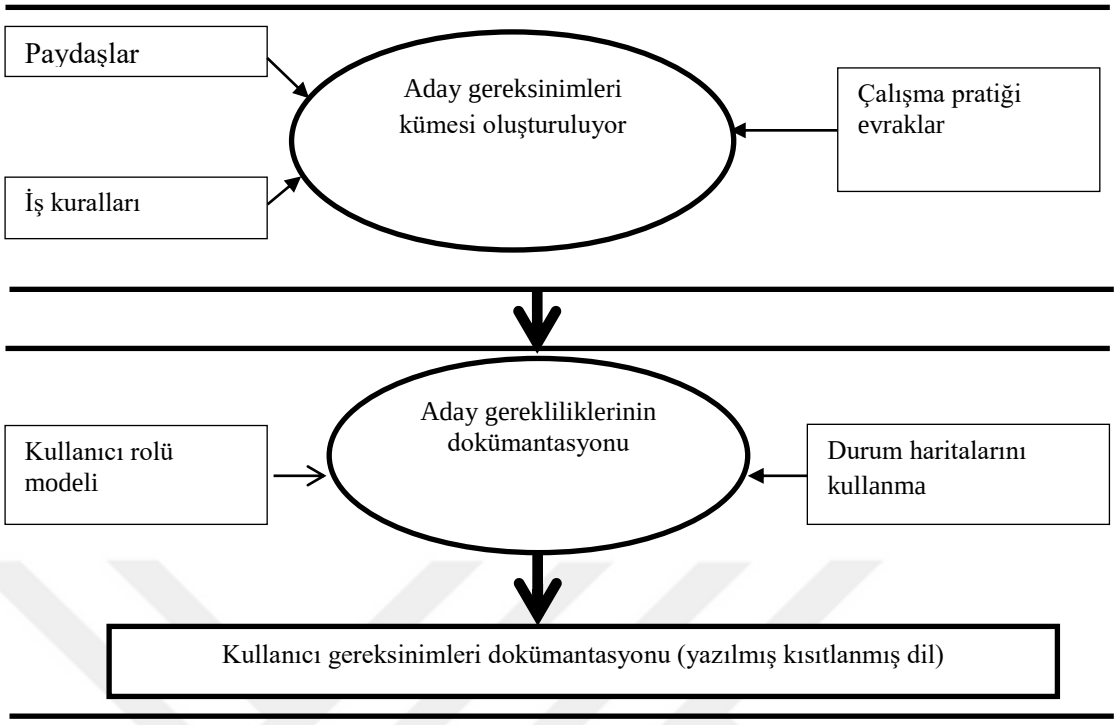
gerektiğinde ortaya çıkar. Örneğin, bir kullanıcı geçersiz bilgi girer, Bu durumda, kullanıcının süreci için bazı işlemler (ör. Süreci yeniden hazırlama) güncelleme veya ekleme veya işlemi sonlandırma (başka bir deyişle İptal veya İptal etme) şansı vardır. Bazı işlemlerin yapılmasını gerektiren "Yeniden Yapılanma" durumu hakkında daha ayrıntılı bir örnek, gerekli veritabanı bulunmadığında; Sistem, kullanıcıya bir veritabanının saklanabileceği başka bir depolama yerini tanımlama şansı verebilir.

Ek olarak, istisnalar, kullanıcı ve sistemler arasında paylaşılan bir sorumluluk olabilir. Bu, bazı sistem istisnalarının kullanıcılardan kaynaklandığı anlamına gelir. Örneğin, bir kullanıcı sıfır değerini girdiğinde ve işlemin sıfıra bölünmesi gerektiğinde. Bu, "sıfır istisnalarla bölünme" ile sonuçlanacak; ve sistem kullanıcıya "sıfır olmayan" bir değer girme şansı verebilir, aksi takdirde, herhangi bir elleçleme mekanizması bulunmadığında iptal sonlandırma durumu gerçekleşir.

3.3. Kullanıcı gereksinimleri belirleme yaklaşımı

Eksik veya belirsiz gereksinimler, yanlış ve eksik spesifikasyona neden olan kritik konulardır: Bu tür sorunlar, alan karmaşıklığı, müşteriler ihtiyaçlarını erken ifade edememesi, veya hatta geliştiricilerin müşterilerin ihtiyaçlarını yanlış anlaması nedeniyle olabilir. Sistemlerin gelişmelerinden kaynaklanan sorunlar, bazen müşteriler ile geliştiriciler arasında iletişim eksikliği bulunduğunu göstermektedir

Bu problemlerin üstesinden gelmek için, yazılım gereksinimlerinin açıklığa kavuşturulmasına yönelik bir yaklaşım önerilmektedir. Birkaç faaliyetten oluşur ve bu faaliyetler boyunca bazı belgeler oluşturulur. Bu faaliyetleri takip ederek gereksinimler doğrulanır ve müşterinin geri bildirimi iyi yönetilir. Önerilen yaklaşımın kavramsal genel görünümü Şekil 3.5' te gösterilmiştir.



Şekil 3.5. Çıkarma sürecinin kavramsal genel görünümü

3.3.1. Aday ihtiyaçları kümesi

Bu etkinliğin amacı, geliştirilecek sistemi anlamak ve geliştiriciyi problem alanına alıştırmak, paydaşlardan gelen aday ihtiyaçlarını toplamak ve kaydetmektir.

Potansiyel paydaşlarla yapılan bir röportaj ve çalışma belgeleri akışının gözlemleri, bu gereksinimleri ortaya çıkarmak için kullanılabilir. Mülakatlar, bir veya daha fazla paydaş içerebilir. Ayrıca, diğer potansiyel menfaat sahiplerini ve bu gereksinimlerler arasındaki herhangi bir çelişkiyi keşfetmek için kullanılan bir soru-cevap oturumunu içerebilir.

Mülakatlar paydaşların ihtiyaçları ve bunlarla ilgili değişiklikler konusunda onay vermeyi kolaylaştırır.

Menfaat sahiplerinin gereksinimlerini ortaya çıkarmadan önce geliştirici, kararların dayandığı iş kurallarını ve koşullarını ve sistemin iş yürütmesini kontrol eden ve tanımlayan prosedürleri bilmeli ve bunların geliştirilen iş üzerindeki etkilerini göz önünde bulundurmalıdır.

Bu aktivite sırasında elde edilen bilgilere dayanarak, geliştirici, daha sonraki faaliyetlerin temelini oluşturan bazı belgeler oluşturur.

3.3.1.1. Paydaşların kimlikleri

Tüm Gereksinimleri keşfetmenin ilk adımı tüm paydaşların kim olduğunu ve hangi rolleri oynaması gerektiğini anlamaktır (yani projenin sosyolojisini anlamak zorundayız). Sistemin paydaşları, sistemle yakından ilgili kişilerdir, doğrudan kullanıcılar, kullanıcıların yöneticileri, gelişmiş sistemle etkileşime giren diğer sistemler ve üzerinde çalışanlar olabilirler. Bu, tüm paydaşların gereksinim süreci hakkında daha iyi bir anlayışa sahip olmalarını sağlar ve bunun sonucunda istikrarlı ve eksiksiz bir şart kümesi elde etme görevi daha da kolaylaşır. Ayrıca, paydaş ihtiyaçlarının eksik bir şekilde anlaşılması, eksik veya hatalı gerekliliklere neden olabilir, bu da yanlış bir yazılım çözümü geliştirir.

Ardından geliştirici, sistemin paydaşları hakkında isimleri, pozisyonları ve ilişkisel açıklamalar gibi sistemle ilgili gerekli bilgileri yazmalıdır.

Sağlık sistemindeki basit paydaş örnekleri Tablo 3.1'de gösterilmiştir. Doktor'un sistemle doğrudan etkileşimi vardır. Doktor hastaları tedavi etmek ve takip etmek üzerine yoğunlaşırken, eczane, hastalara doktor tarafından reçete edilen ilaçlara odaklanır.

Tablo 3.1. *Paydaşların tanım kayıtlarına bir örnek*

Paydaş adı	**Pozisyon**	**Sistem ile ilişki**
Doktor	Sağlık ekibi	Hastalara tıbbi tedaviden sorumlu
Eczacı	Tıbbi ekip veya özel eczane içinde	Tavsiye eder ve doktor tarafından reçete olarak ilaç verir

3.3.1.2. İşyeri uygulama belgelerinin denetimleri

Gereksinim çıkarma, paydaşların ihtiyaçlarını yakalamak için önemli bir süreçtir (kullanıcı gereksinimleri). Başlangıçta, müşterinin ihtiyaçlarını tam olarak

belirleyebilecek yeterli bir yeteneği olmayabilir. Bunu basitleştirmek için, şu anda müzakerede bir başlangıç noktası olarak oynadıkları rollere göre gerçekleştirilen kullanıcının görevleri tanımlanır; bu, müşteri ile geliştirici arasındaki etkileşimi basitleştirecektir. Ayrıca, bahsedilen ilk gereksinimlerle ilgili müşterinin geri bildiriminde yardımcı olur. Tablo 3.2 de, bir doktorun çalışma sorumluluk belgesine bir örnek verilmiştir.

Tablo 3.2. *İş sorumluluklarının bir örneği kayıt*

Kullanıcı Rolü Adı	**Doktor**
Rol sorumlulukları	**İş Kuralları ve Kısıtları**
Hastaları teşhis etme	Hasta öyküsünü kontrol et
Hastaları izleme	İlerleme raporu yaz
Ameliyat işlemleri yap	Gerekli tüm analizleri kontrol et İmza gerekli
Reçete yaz	----

Bu aktivitenin sonunda, geliştirici iş kurallarını ve çalışma kısıtlamalarını bilmiş, paydaşları, prosedürleri ve sorumlulukları tanımlamıştır.

3.3.2. Aday kullanıcı gereksinimleri belgeleri

3.3.2.1. Kullanıcı rolü doğrulaması

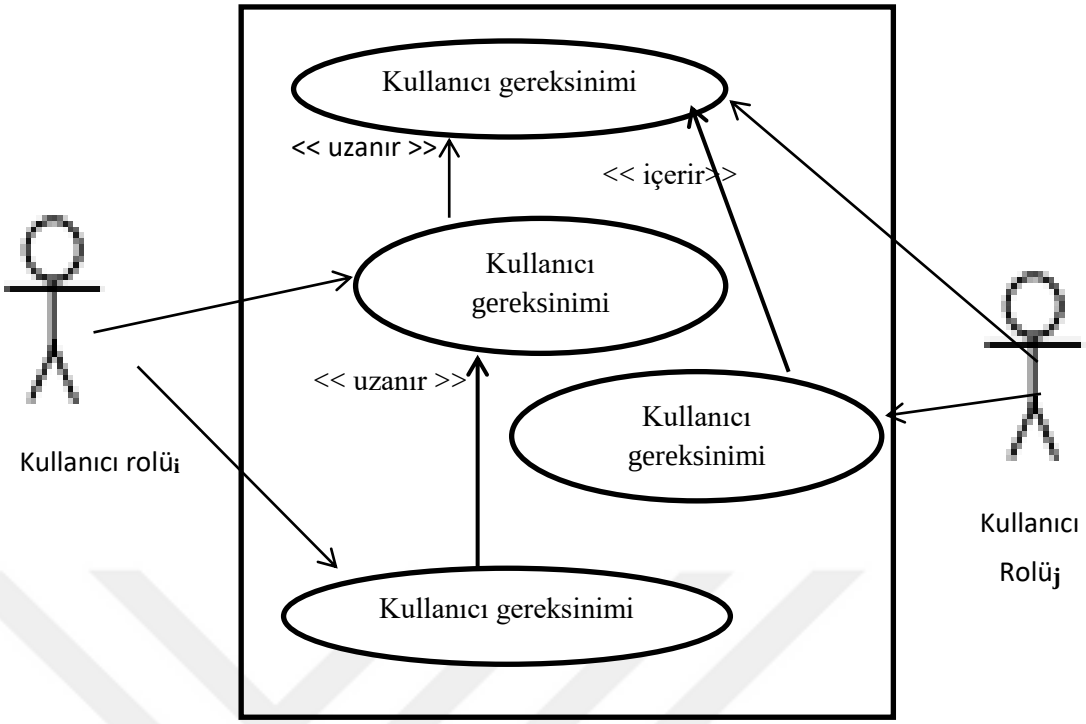
Bu aktivitede, aday gereksinimleri ilgili kullanıcı rollerine göre gerçekleştirilecektir. Her bir gereksinim, sistemin ve kullanıcının sorumluluğunu göstermek için UCM'ler biçiminde gösterilir ve ardından eşdeğer pencereler gezinti prototipi, gereksinimin içerebileceği durumların çoğunu göstermek için hazırlanır (adımlar). Menfaat sahiplerinin geri bildirimleri, güncelleme veya gereksinim değişikliği şeklinde veya belki de yeni gereklilikler eklenerek belgelenecek ve yönetilecek ve onaylanan her görev belgelenecektir.

Her bir aday gereksinim için bir prototip oluşturması, müşteri geri bildirimi almak için yararlıdır. Bu süreç, müşterinin ihtiyaç duyduğu yeni gereksinimleri güncellemeye, değiştirmeye ve hatta eklemeye yardımcı olur. Bu süreç, müşterinin ihtiyaç duyduğu yeni gereksinimleri güncellemek, değiştirmek veya hatta almak için birçok yönden yardımcı olur.

Bu süreci basitleştirmek için, her kullanıcı rolü tarafından yürütülen aday gereksinimlerini göstermek için UML kullanım durumu diyagramı ile başlanması önerilmiştir. Bu diyagram, kullanıcı sistemlerinin davranışını göstermek için kullanılabilir (Kösters, Six, & Winter, 2001). Kullanım örnekleri (eliptik şekiller) senaryolardan oluşur; her senaryo, kullanıcı gereksinimlerini tanımlayan bir dizi cümle veya adımdan oluşur. Aktörler ise bu senaryolarla etkileşim için sistem kullanıcıları tarafından oynanan rollerdir.

Bu diyagramı, önerilen kullanıcılar arasında sorumlu oldukları gereksinimlere göre (diğer bir deyişle bir veya daha fazla gereksinimden sorumlu olan her rolde) etkileşimleri göstermek için kullanılması kabul edilmiştir (Şekil 3.6). Bu, cevapsız olabilecek (belirtilmeyen) diğer kullanıcı gereksinimlerini doğrulamaya ve algılamaya neden olacaktır.

Buna ek olarak, kullanım vaka ilişkilendirme ilişkileri (örneğin, "genişletir" ve "içermektedir"), görev kavramını (kullanıcı gereksinimi) tamamlamak için kullanılacaktır; çünkü bir kullanıcı gereksiniminin eksik olması daha sonradan onarılması zor olan hatalar veya istisnalara neden olabilir.



Şekil 3.6: Kullanıcı rolü modeli

Gösterildiği gibi, aktörler rollerdir: "Kullanıcı rolü$_j$" iki kullanıcı gereksiniminden sorumludur ve "Kullanıcı rolü$_i$" iki Aday Gereksinimlerinden sorumludur. İlişkilendirme ilişkisi, bazı Aday Gereksinimleri arasındaki bağımlılığı gösterir. Ancak, şekilde gösterildiği gibi, "Kullanıcı rolü$_j$" Aday Gereksinimlerinden biri, diğer Aday Gereksinimlerine bağlı değildir.

3.3.2.2. Kullanıcı gereksinimlerini görselleştirme

Kullanıcı gereksinimlerini açıklığa kavuşturmak tam ve kesin gereksinimlerin üretilmesi için önemlidir. Müşterilerin genellikle geliştiricilerin yardımı olmadan kendi ihtiyaçlarını tam bir biçimde ifade edemediği bilinmektedir. Kullanıcı gereksinimlerinin tanımını açıklığa kavuşturmak için, kullanıcı gereksinimlerini görsel olarak açıklığa kavuşturmak ve doğrulamak için UCMs notasyonları önerilmektedir. Bu gösterimler, müşterilerin geri bildirimlerini netleştirmeye yardımcı olan daha fazla anlama ve eğitim tekniği için kullanılan bir teknik olarak düşünülebilir.

UCM'ler, Buhr ve Casselman tarafından ortaya çıkarılan, nedensel bir yol (lar) da yüksek seviyede bir sistemi temsil etmek için kullanılan görsel bir açıklama aracıdır (yani, rahat yolların bir biçiminde sistem işlevlerinin kuşbakışı görünümü) (RJ Buhr & Casselman, 1995). Her yol, uzun bir yolun sorumluluklarının sayısından ve bu sorumluluklar arasındaki nedensel ilişkiden oluşur. Her senaryo, bir objektif veya bir görevi göstermek için sorumluluklar biçimindeki bileşenler arasındaki bir etkileşimi temsil eder.

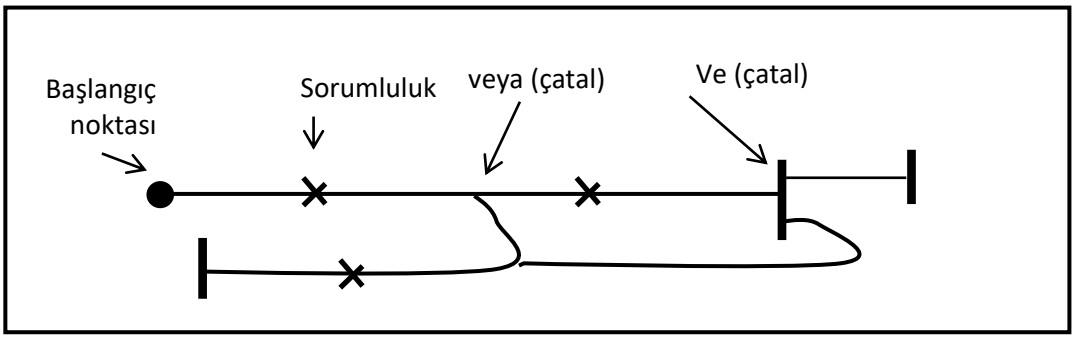
UCM elemanları, bir veya daha fazla bileşenin karışımı olabilir, her bir bileşen, içindeki eylemlerden sorumlu olan bir varlığı temsil eder (Amyot ve Mussbacher, 2000). Her bir bileşenin birtakım sorumlulukları vardır ("X" harfi ile temsil edilen) ve bu bileşenleri ilişkilendiren bir yol, bu sorumlulukların akışını temsil eder ve bu bileşenlerin bir nesneyi gerçekleştirmek veya bir görevi yerine getirmek için işbirliğini temsil eder.

Bir yol senaryosunda, başlangıç noktası senaryonun bir ön koşulunu (siyah dolu bir daire ile temsil edilir) ve bir bitiş noktası bir senaryonun (bir katı çubukla gösterilen) son durumunu gösterir. UCM'ler bağlanmamış olabilir, bu da bileşenleri göstermediği (yani bileşenlerin isteğe bağlı olduğunu gösterir) anlamına gelir (Amyot ve diğerleri, 2000).

Buna ek olarak UCM, fork ve fork gösterimleri gibi bazı ekstra gösterimler içerir. Çatal bir yolun bir çok alt yola bağlandığı anlamına gelse de, alt yollarda-veya-fork yolları devam eder. Şekil 3.7, UCM gösteriminin basit bir görünümü verilmiştir. Dolgulu daire, senaryoların ön koşullarını ve katı çubuklar senaryoların bitiş noktalarını (yani, son durumlar) gösterir.

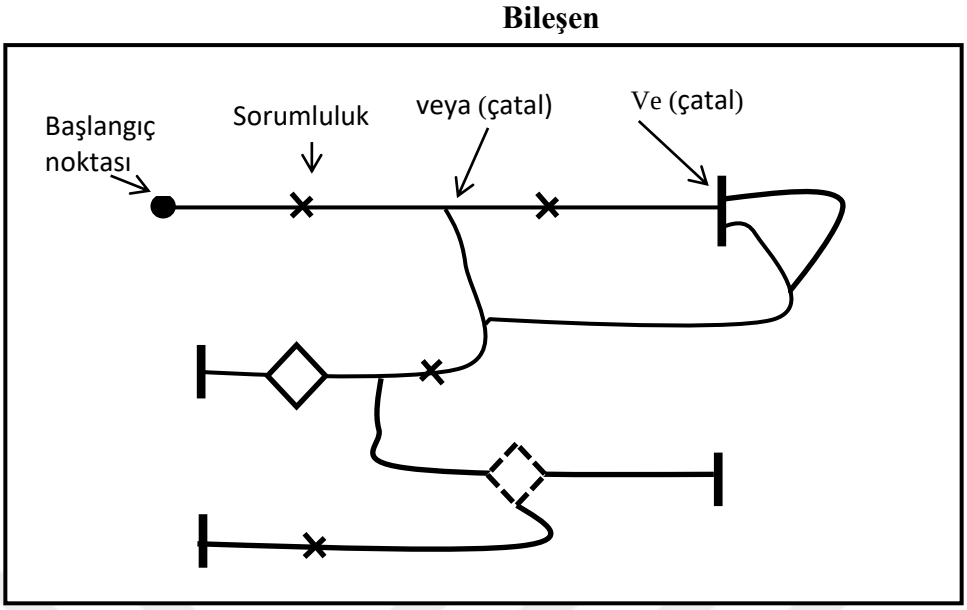
Bu tezde, iki bileşenli temsile odaklanılmıştır. Biri sistem, diğeri ise kullanıcıyı temsil eder.

BILEŞEN



Şekil 3.7: Basit UCM gösterimi

Karmaşık bir sorumluluk (görev) olması durumunda, ayrıntılı bir prototip bu görevi göstermek açısından önemlidir ve verilen senaryoda bir eklenti olarak düşünülür. UCM eklentileri kütük olarak adlandırılır. Bir kütük, alt senaryolardan (alt-haritalar) daha fazla olduğunda, bu kütük dinamik bir kütük olarak kabul edilir ve kesikli elmasla ifade edilir. Yalnızca bir takoz içeren bir kütük, statik kütük olarak kabul edilir ve sağlam bir elmas tarafından temsil edilir. Bu eklentiler, karmaşık görev akışlarını göstermek için kullanılabilir (bkz. Şekil 3.8). Her bir eklenti, bir prototip formundaki sistem bileşenleri arasındaki etkileşimi netleştirmek için kullanılır. Bu, ana kullanıcının işlevlerinin bölümlerini (senaryolarını) temsil eden alt gereksinimlerin (görevlerin) açıklığa kavuşturulması ve doğrulanmasına yardımcı olacaktır. UCM'nin daha fazla açıklaması (Mussbacher, Amyot, & Weiss, 2007) 'de bulunabilir.



Şekil 3.8: Kütüklü UCM'nin basit bir örneği

Şekil 3.8'den açıkça anlaşılmaktadır ki, UCM birkaç notasyon içermektedir: sorumluluklar, "veya çatal", "Ve çatal", statik ve dinamik kütükler. "Or-fork", senaryonun takip edilen yollarından birini seçmek için bir seçenek sunar, And-fork ise katı çubuğun bağlı olduğu takip edilen iki yolun eşzamanlı olarak gerçekleştirileceğini gösterir. Sonunda yol senaryosu planlandığı gibi devam edecektir.

Statik kütük ve dinamik kütük, bu eklentilerin içerdiği gizli yolları göstermek için eklenti senaryolarının gerekli olduğunu gösterir.

Geliştiricinin, kullanıcı gereksinimlerinin prototipini müşteriyle tartışması gerektiği önemlidir. Bu, müşterinin geri bildirimini yakalamayı kolaylaştıracaktır; Bu da geliştirme aşamasındaki sistemin davranışının müşterinin gereksinimlerini karşılayacağını onaylayan eylemler anlamına gelir. Bu geri bildirim, müşterinin diğer kullanıcı gereksinimlerini güncellemesine veya yakalamasına neden olabilir ve ayrıca geliştirici, normal olarak önemli veya kaçırılması beklenen bazı ek gereksinimler önerebilir. Bu değişiklikler veya eklemeler, önerilen prototip yaklaşımı kullanılarak açıklığa kavuşturulabilir ve doğrulanabilir.

Onaylanmış her kullanıcı gereksinimi için belirsizliği azaltmak ve önerilen araç çalışmasını daha sonra kolaylaştırmak için doğru belgeler gereklidir. Bu dokümantasyon, kullanıcı faaliyetleri ile sistem yanıtları arasındaki etkileşim şeklinde kısıtlanmış kurallardan bazılarını dikkate alarak yazılmalıdır. Daha kesin sonuçlar elde etmek için, (Yue, Briand, & Labiche, 2009) tartışılan kullanıcı gereksinimleri belirtimine ilişkin kısıtlama kurallarının bazıları benimsenir ve önerilir:

- Olay akışlarını doğru bir şekilde açıklamak
- Sistemin konusu sistem veya bir aktör (kullanıcı) olmalıdır.
- Sırayla olay akışını açıklayın. "U:" aktörü (kullanıcı), "S:" ise sistem yanıtlarını ifade eder.
- Cümle başında bir eylem tarif edilir (tercih edilir).
- Mevcut zamanın kullanılması tercih edilir.
- Model fiilleri kullanmayın (örn., Zorunlu, gerekir, ..etc.)
- Olumsuz zarf ve sıfat kullanmayın, örneğin PIN numarası hiç bir zaman doğrulanmadı.
- Sonunda, cümleler gramatik olarak doğru yazılmalıdır.,

3.3.2.3. Süreç hedefi modeli

Bu model, kullanıcı rolü modelinden türetilir. Her bir planın yerine getirilmesi için bir veya daha fazla kullanıcı gereksiniminden oluşan süreç tamamlama planını açıklar (Tablo 3.3). Bu model üç sütundan oluşur: Hedef sütunu, ulaşılacak hedefin adını gösterir; Kullanıcı gereksinimi sütunu, hedefin tamamlanması için gerekli olan kullanıcı gereksinimleri listesini ve son sütun, her kullanıcının gereksinimlerinden sorumlu olan kullanıcı rollerini belirtir.

Tablo 3.3. *Süreç hedefi modeli*

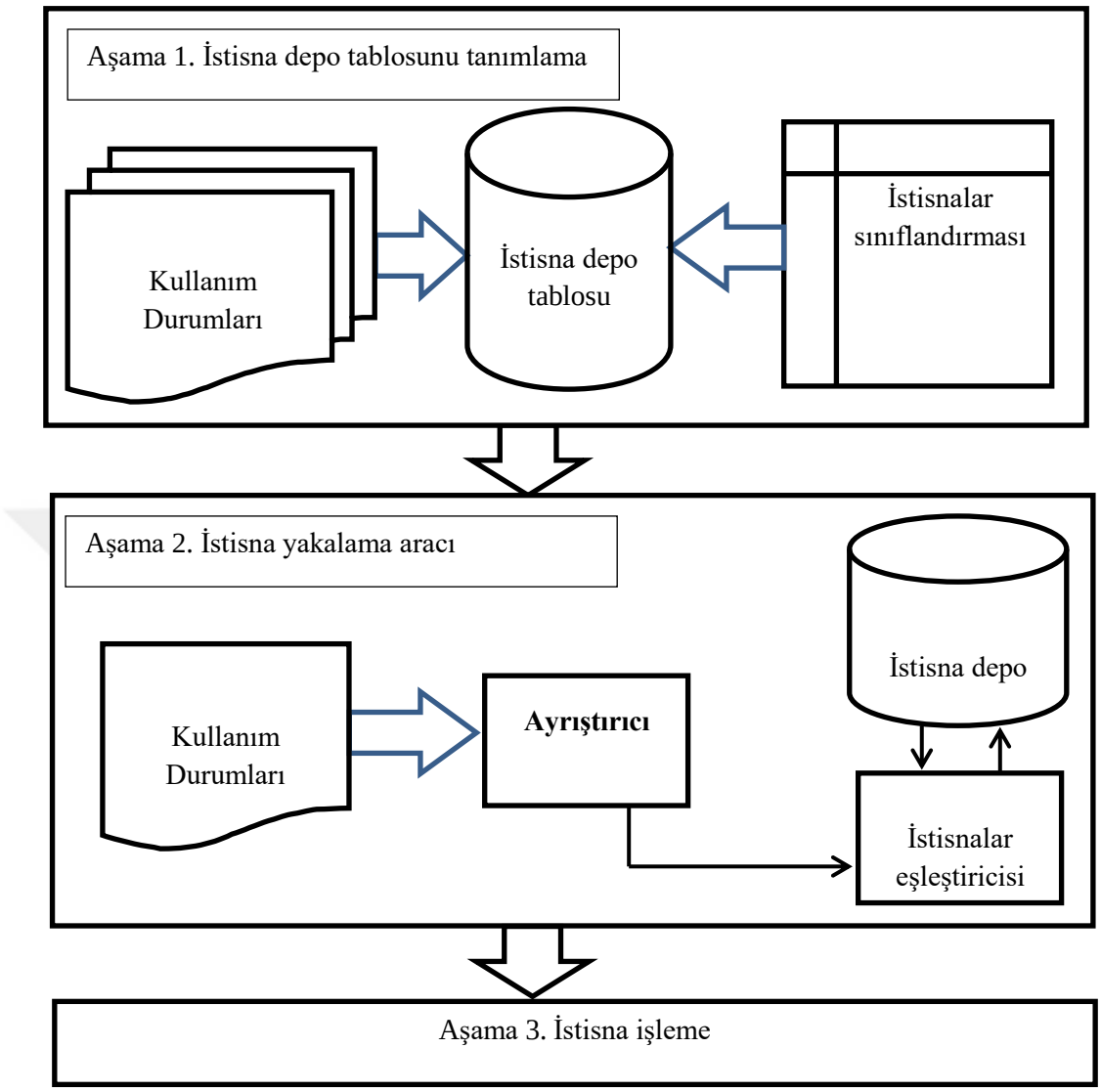
Hedef	Kullanıcı gereksinimleri	Kullanıcı rolü

3.4. İstisnaları algılama mekanizması

Daha önce de belirtildiği gibi, istisnalarla uğraşan birçok ampirik çalışma ve bunların taşıma teknikleri önerilmiştir. Bu çalışmalardan elde edilen sonuçlar, istisnalarla uğraşmanın zor bir görev olduğunu göstermiştir ve bir çok geliştirici, daha sonra istisnalara dayanan sorunlarla yüz yüze gelene kadar bunlarla başa çıkmayı ihmal etmiştir. Bu, sonraki aşamalarda baş etmek için daha fazla zaman ve çaba gerektirir.

Bu tez kapsamında, problemin üstesinden gelmek için, erken aşamada olası istisnaların çoğunun tespit edilmesi için bir mekanizma önerilmiştir (yani Gereksinimler aşaması). Mekanizma birkaç aşamadan oluşmaktadır. Bu aşamalar, hazırlık aşamasını temsil eden (önerilen istisnalar sınıflandırmasına dayanan) ve mekanizmanın son aşamasını temsil eden son aşamaya (elleçleme aşamasına) kadar uzanan tüm olası istisnaların bir havuzunu (sözlük) oluşturarak başlar. Önerilen aşamaları takip ederek, potansiyel istisnaların çoğu tespit edilir ve işlenir.

Son aşama (Elleçleme aşaması) mekanizmanın son aşamasını temsil eder. Önerilen aşamaları izleyerek, olası istisnaların çoğunu tespit eder ve bunların taşıma teknikleri açıklığa kavuşturulur. Bundan sonraki bölümlerde önerilen mekanizmanın aşamaları detaylı olarak açıklanmıştır (Şekil 3.9).



Şekil 3.9 İstisnalar algılama mekanizmasının kavramsal genel görünümü

3.4.1. İstisna depo tablosunu tanımla

Bu tablo, olası istisnaları (veri kümesi) tanımlayan temel kayıtlar olarak düşünülebilir. Kısıtlamaları ihlal eden ya da iyi performans göstermediğinde istisnalara neden olan istisna eylemlerinden oluşur ve bu, önerilen istisna sınıflandırmasına göre yapılır (istisnalar sınıflandırması aşamasında gösterilmiştir).

Başlangıç aşamasındaki istisnalar, geliştiriciler tarafından ele alınması gereken durak noktaları veya gereksinimleri karşılamak için ilgili kişilerle müzakereye ihtiyaç duymaları nedeniyle değerlendirilebilir. Örneğin, bir kullanıcı belirli bir seçeneği

belirleme veya "iptal işlemini onayla" gibi belirli bir eylemi onaylamalı ve kullanıcı bu istekleri yok sayar, Bu da istisnai bir durum yakalanmış demektir. Başka bir örnek, bir sistem veritabanına bilgi depolamaya çalışırken, veritabanı mevcut değilse bu istisnanın ele alınması gereken bir durumdur. Bu örneklerde, "Seç ve Sakla" iki anahtar kelime veya eylem (durdurma sözcükleri), istenen eylemler doğru yapılmadığında istisnalara neden olabilir.

İstisna tabloları, olası istisnaları ve açıklamaları kaydetmek için gereklidir. Bu çalışmada önerilen sınıflamaya göre, istisnalar iki temel hususa odaklanmıştır: bir sistemle zayıf kullanıcı etkileşimi ve sistemin kendisinden istisna. Birçok kullanıcı hatasının, yanlış girdi sürecinde yoğunlaştığını veya kullanıcının tüm süreci tamamlamak için belirli bir işlemi onaylamasını gerektiren bazı etkileşimlerin uygulanmasını vurgulamama sürecinde yoğunlaştığını belirtmek isteriz.

Ayrıca, senaryodaki istisnalar havuzda (istisnalar tablosu) bulunmadığı için senaryodaki istisnalar tespit edilemeyebilir, bu nedenle senaryo anahtar kelimelerinin eş anlamlıları kontrol edilecek anahtar kelimeler olarak düşünülebilir. Böylece senaryo, amacını yerine getirmeye devam eder. Bu işlemi kolaylaştırmak için, "WordNet" kullanımı, istisnalar tablosuna eklenen ek durdurma sözcükleri olarak bu eş anlamlıları göz önünde bulundurarak bu sorunu ele almaya yardımcı olacaktır. Kelime anlamını yakalamak için WordNet (İngilizce için sözcük bilgisel bir veritabanı) sözlüğü kullanılır. Bu tablo, kendi istisna anahtar kelimelerini tanımlamak için gerektiğinde geliştiricilerin isteği üzerine yetiştirilebilir. Tablo 3.4, zayıf kullanıcı etkileşimi istisna sözlük tablosu örneğini gösterir ve 3.5, bir sistem yanıtları istisna sözlük tablosu örneğini gösterir. Bu tablolar aşağıdaki öğelerden oluşmaktadır.

Durma noktası: İhlal edildiğinde bir istisnayı temsil eden bir sözleşme sözcüğü olarak düşünülebilir. Örneğin, "giren" bir kelime, kullanıcı bir geçersiz veri girdiğinde bir istisna oluşturabilecek aday kelimedir
İstisna açıklaması: geçersiz girişten kaynaklanabilecek bir istisnanın kısa bir açıklaması ("giren kelime" ye karşılık gelir).

Tablo 3.4. *Kötü kullanıcıların etkileşiminin neden olduğu istisna tablolarının bir örneği*

Durma noktas	**Istisna açıklama**
Insert, input, feed, enter , fill, specify	IOException
set, decide, deliver	IOException
Command, determine, require	ProcessTerminationException
Select, choose, accept, confirm, take, elect	ProcessTerminationException
Approve, okay, affirm, certify	ProcessTerminationException
Submit, state,	ProcessTerminationException

Tablo 3.5. *Sistem yanıtlarının neden olduğu İstisna Listeleri tablosunun bir örneği*

Durma noktası	**İstisna açıklama**
assign, allocate, appropriate, give, specify, allow	ResourceUnavailbilityException
set, determine, open, close, reserve, check, store	ResourceUnavailbilityException
search, catch, find, database, file, delete, remove	ResourceUnavailbilityException
arrange, put, order, dispose	ArrayIndexException
calculate, compute, estimate, count	MathOperationException
enumerate, index	ArrayIndexException
divide, split	DivisionbyZeroException
extract ,get, inquire, store, keep	ResourceUnavailbilityException

3.4.2. İstisna yakalama aracı

Doğal dil işlemek (NLP), doğal metni analiz etmek ve göstermek için bilgisayarlı bir tekniktir (Cambria & White, 2014)

İstisna yakalama yazılımı (Ek1) Durumları otomatik olarak analiz etmek ve tespit etmek. bir kullanım durumu senaryosunu ayrıştıran doğal bir dil çözümleyici içerir (yani, bir olay grubunu içerir ve her olay, bir kullanıcıdan bir sisteme veya bir mesajın tersine (yanıt) bir iletiyi uyaran bir eylemi temsil eder). Bu senaryo, bu tezde önerilen kurallara göre kullanıcı ve sistem etkileşimi şeklinde kısıtlı bir doğal dilde yazılmıştır (bkz. Bölüm 3.3.2).

Bu araç, hangi ifadelerin istisnalara neden olabileceğini ve istisnalara neden olabilecek anlamsal bilgileri çıkarır. Bu adım, yakalanan sözcüklerin (örneğin fiillerin) uygun biçimde dönüştürülmesini gerektirir.

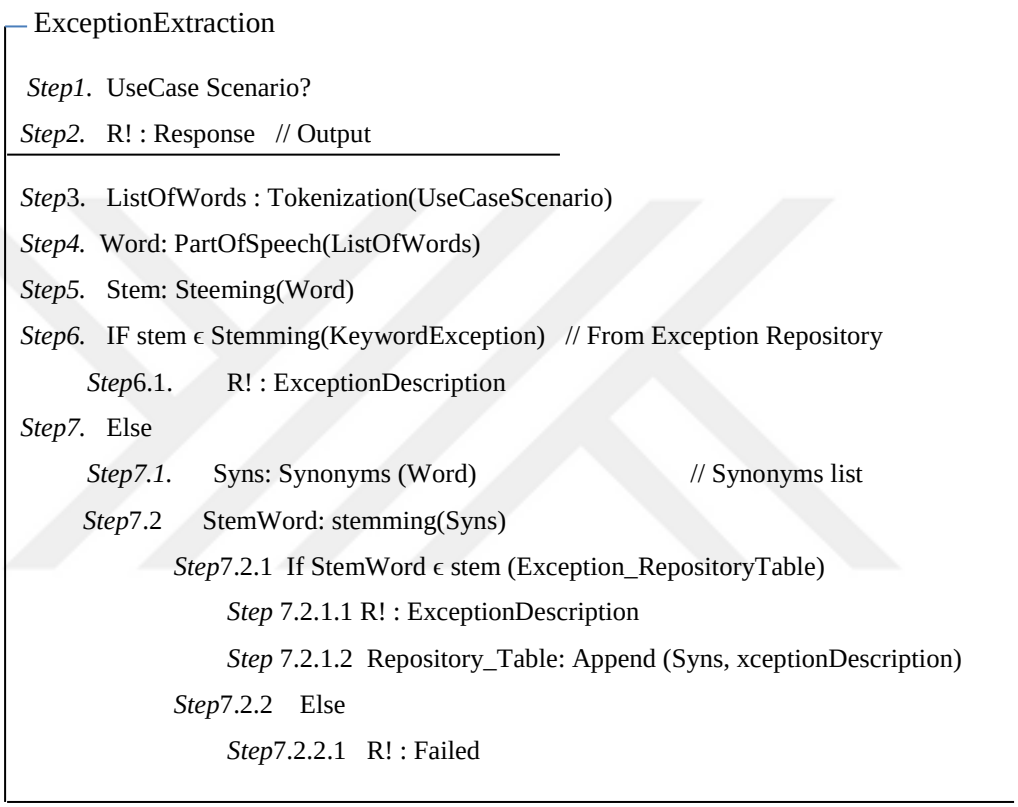
Korpus dilbiliminde, çoğu NLP, corpora olarak bilinen geniş dilsel verilerin toplanması üzerine kuruludur; corpus'un çoğulluğu (Baker, 2006) ve metnin

çözümlenmesindeki diğer bir aşama, her simgeyi gramer kategorisine ya da konuşmanın bir bölümüne (POS) bağlamaktır. Aşağıdaki adımlar, algılama mekanizmamızın fikrini açıklığa kavuşturmaktadır:

- İlk olarak, Tokenleştirme tekniği kullanılır. Bu, senaryodaki her bir bildirimin bazı parçalara (Tokenlere) bölüneceği anlamına gelir. "Tokenleştirme", "Tokenizasyon" işleminin kontrolüne izin veren daha gelişmiş metin bölünmesine ulaşmak için verilen düzenli ifadeler kullanılarak yapılır ve daha sonra, her bir metin parçası, karşılık gelen POS ile eşleştirilir (yani, her bir kelimeye bir POS atanacaktır)). Bu, sözcüklerin isim, fiil veya benzerlerine göre sınıflandırılması anlamına gelir. Dahası, kontrol çoğunlukla fiiller üzerinde yapılır, çünkü bazı kelimeler bu çalışmada düzensiz kelimeler olarak kabul edilebilir. Bu nedenle, bu çalışmada kullanılan POS etiketleri şunlardır: fiil için VBZ (üçüncü şahıs tekil), fiiller için VB (taban formu), fiil için VBD (geçmiş zaman), sıfat için JJ, ad için NN (tekil veya kütle), NNS için isim (çoğul). Bu nedenle, her kelimenin sınıflandırılması, her kelimenin sözcüksel anlamındaki belirsizlik derecesini azaltacaktır. Örneğin, "mağaza" kelimesi, tasarruflara eşdeğer olan sistem mağazaları gibi bir fiil olarak kabul edilebilir ve diğer durumlarda, bir isim olarak kabul edilebilir.
- İkincisi, stemming işlemi için bir porter algoritması (en yaygın İngiliz fermmer) kullanılır. Bu, arama yapmak üzere anahtar kelimeyi kök formuna döndürmek anlamına gelir (yani ekleri kaldırma). Stemming işlemi her iki tarafta da yapılacak; yakalanan anahtar kelime ve istisna tablosunda saklanan anahtar kelime. Örneğin, "mühendislik" sözcüğünden kaynaklanan "mühendis" kelimesi olacak ve "yürüdü yürüdü" kelimesi "yürümek" sözcüğü olacaktır .
- Son olarak, WordNet (ingilizce dili için bir sözcüksel veritabanı), çıkarılan belirteçlerin anlamlarını (ortak bir anlamı paylaşan eş anlamlıları) aramak için kullanılacaktır.

Şekil 3.10, İstisna Tespit İşleminin Z-Şematik göstermektedir.

İstisna eşleştiricisi, ayrıştırıcıdan döndürülen anahtar kelimelerle (anahtar anahtar kelimeler), depo istisna tablosunda saklanan anahtar kelimelerle aramak suretiyle olası tüm istisnaları bulmak için bir anahtar kelime araştırmacısına sahiptir. Arama anahtar kelimeleri tabloda yer almıyorsa, istisnalar tablosunda saklanan anahtar kelimeler yerine eşanlamlılar kullanılacaktır. Bu eş anlamlılar, sözlüğe istisnaların yeni anahtar kelimeleri olarak eklenecektir.



Şekil 3.10. İstisna Yakalama Mekanizmasının Z-Şeması

Aşağıdaki adımlar Şekil 3.10 daki şemayı daha ayrıntılı olarak açıklamak için verilmiştir.

Aşama 1. Bu açıklayıcı bir adım, önerilen aracın girdisinin, Bölüm (3.3.2) 'de tanımlanan kurallara göre kullanıcı eylemleri ve sistem yanıtları formunda yazılmış olan kullanım senaryosudur

Adım 2. Bu aynı zamanda açıklayıcı bir adımdır. Bu, yazılım çıktısının, verilen senaryoda içerebilecek beklenen istisnaların listesi olacağını gösterir. "R!", Algılama işleminin sonucunu temsil eder.

Aşama 3. Bu adımda, verilen senaryo birkaç parçaya bölünecektir. Bu Parçalar, NLP kavramlarında Tokens olarak temsil edilir.

Adım4. Bu adımda, her belirteç NLP'deki konuşmanın ilgili kısmına sınıflandırılacaktır (yani fiiller, isimler, vb.).

Adım 5. Kelimeler farklı formatlarda yazılabilir ve bu da sınıflandırılmış kelimeler için de geçerli olabilir, bu nedenle arama sürecini basitleştirmek için her kelime köküne geri döndürülür (bu mekanizma, NLP kavramlarında dalgalanma süreci olarak adlandırılır). Örneğin, sözcük kabul edilen, kabul veya kabul ederek yazımı kabul eder, bu nedenle sözdizimsel işlemden sonra "kabul" olur. Saptırılmış sözcüklerin her biri, ihlalinin senaryo sonlandırmasının nedenine (yani bir istisna oluşursa) sözleşme sözcüğü olarak addedilecektir.

Adım 6. Her Stemming kelimesi istisnalar tablosunda kontrol edilecektir. Bulunursa, ilgili istisna geri dönecektir (adım 6.1'de gösterilmiştir).

Adım 7. Bazı durumlarda, yakalanmış köklü sözcükler depo tablosunda dikkate alınmayabilir. Bu, verilen senaryoların istisnalar dışında olduğu anlamına gelmez ve bu nedenle doğrulama sürecini arttırmak için, bu sözcüklerin eşanlamları WORDNET veritabanından alınır (adım 7.1) ve sonra bu eşanlamlı kelimeler üzerinde stemleme işlemini gerçekleştirir (adım7.2)). Bu kaynaklı eşanlamların herhangi biri depo tablosuna (adım7.2.1) dahil edilirse, onun ilgili istisnası geri verilir (adım7.2.1.1) ve daha sonra depo tablosuna yeni bir sözleşme sözcüğü olarak eklenecektir (adım7.2.1 .2). Son olarak, depo tablolarında bu eşlemelerin hiçbiri mevcut değilse, bu yazılımın verilen senaryoda herhangi bir istisna yakalayamadığı anlamına gelir (adım 7.2.2 ve adım 7.2.2.1).

3.5. İstisna işlemlerinin modellenmesi

Başarıyla ilerleyemeyen bir süreç olduğunda, bunun bir istisna oluştuğu anlamına gelir. Herhangi bir sürecin modellemesinin amacı, görevlerinin akışını göstermektir. Bu, bir sürecin normal ve anormal durumları açıklığa kavuşturmaya yardımcı olur. Örneğin, bir kullanıcı yanlış bir veri girerse veya bilgi saklama dosyası

kullanılamıyorsa ne yapılmalıdır? Bu gibi durumlarda; normal senaryoların ötesinde yollara bazı işlemler veya görevler ekleyerek bir taşıma işlemi gereklidir.

Yazılım geliştirme spesifikasyonları, anormal durumları ve bu durumların nasıl işlendiğini anlamayı düşünmelidir. Bu, istisnaların oluşmasının nedenlerinin anlaşılmasını gerektirir.

Bu tür zorlukların sorununu çözmek için, Senaryo Modelleme sürecini kolaylaştırmak için UCM'ler yazılımı önerilmiştir; Bu istisnaların tespit sürecini de geliştirecektir.

Yukarıda belirtildiği üzere, önerilen algılama aracı, senaryolarda, kullanıcı gereksinimlerini temsil eden ve (Bölüm 3.3.2) de belirtilen kurallara göre formüle edilmiş istisnaları tespit etme girişimini daha da ileri götürmektir. Geliştiriciler ve potansiyel paydaşlar arasındaki görüşme fırsatlarını artırmak için, bu programın çıktılarını UCM'yi kullanarak senaryo modelleme süreciyle bütünleştirme süreci, bu istisnaların nasıl giderileceğine ilişkin bir plan geliştirmek için bazı netlik ve erken istisnaların tespit edilmesini sağlayacak ve gereksinimlerin tanımını güçlendirecek kullanım noktaları çizilir. Bu nedenle, özel durum işleme mekanizması şu şekilde özetlenebilir:

- Algılama yazılımı tarafından yakalanan istisnaların UCM'nin sınırları boyunca normal senaryolara genişletilmesi
- Yazılım geliştiricilerin istisnaların nerede oluşabileceğini ve yayılmalarını bilmek için gerekli yayılım noktalarını açıklama
- İstisnaları, istisnalarımız sınıflandırması tarafından tanımlanan durumlara göre eşlemek.

Yukarıdaki noktalardan yararlanmak için, her istisnai durum hakkında aşağıdaki bilgiler mevcut olmalıdır:

- Vakanın adı.
- Vakanın amacı - vaka kullanıldığında.
- Vakanın görsel temsili.

Bu nedenle, istisna durumları iki kategoride düzenlenmiştir: Sonlandırma ve Devam Etme. Her bir kategorinin doğasını ve vakaların açıklaması sunulmuştur. Bu vakalar, yazılım geliştiricileri ile ilgili paydaşlar arasındaki müzakere noktaları olarak düşünülebilir.

Buna ek olarak, işleme süreci hakkında bazı bilgileri keşfetmeleri gereken işleme süreci, geliştiriciler ve potansiyel paydaşlar arasında daha fazla müzakere gerektirebilir; Özellikle bu hususlar paydaşların ihtiyaçlarıyla ilgiliyse ve öte yandan, bu noktalar yazılım geliştirmeyle ilgili teknik sorunları (ör. Dosya Bulunamadı) yansıtıyorsa, bu durumda geliştirici bu ikilemi çözmekten yalnızca sorumlu olabilir.

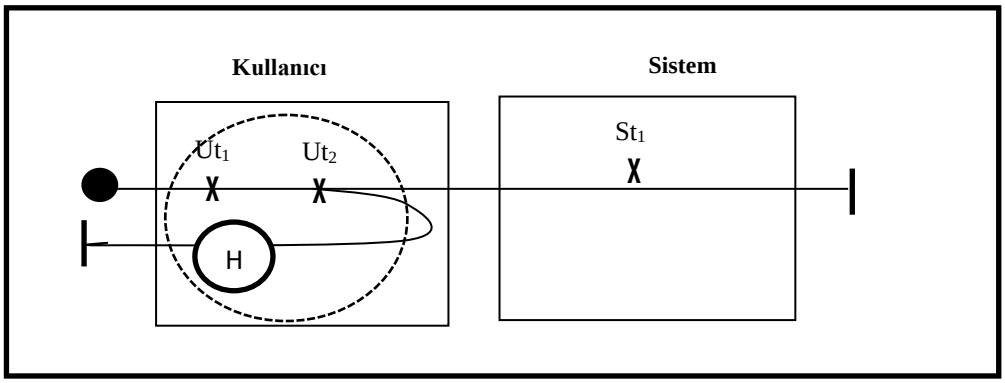
3.5.1. Sonlandırma durumu

Bir işlemi sona erdirmenin yaygın durumlarından biri, kabul edilmeyen bazı işlemler veya kısıtlama ihlalleri nedeniyle sonlandırılmasıdır. Bu kategoride, istisnaların iki durumu belirlenmiştir: İstisnaların bırakılması ve İptal edilmesi.

3.5.1.1. Vazgeçme durumu

Vakanın amacı: bir sürecin sona ermesi tüm süreci tamamlamadan önce gereklidir. Örneğin, tüm süreç "Ut_1", "Ut_2" ve "St_1" görevlerinin tamamlanmasını gerektirir (Şekil 3.11).

Bazı nedenlerden dolayı, sonlandırma, bu görev zincirinin tamamlanmasından önce gerçekleşir. Bu, bir Durdurma durumu oluştuğu anlamına gelir. Bu sonlandırma türü, önceki ilgili görevleri (UCM'deki sorumlulukları) bu sonlandırma hakkında bilgilendirmek için bir yayılım mekanizması gerektirir. Sağdaki daire içindeki "H" harfi bir işleyici gösterir.



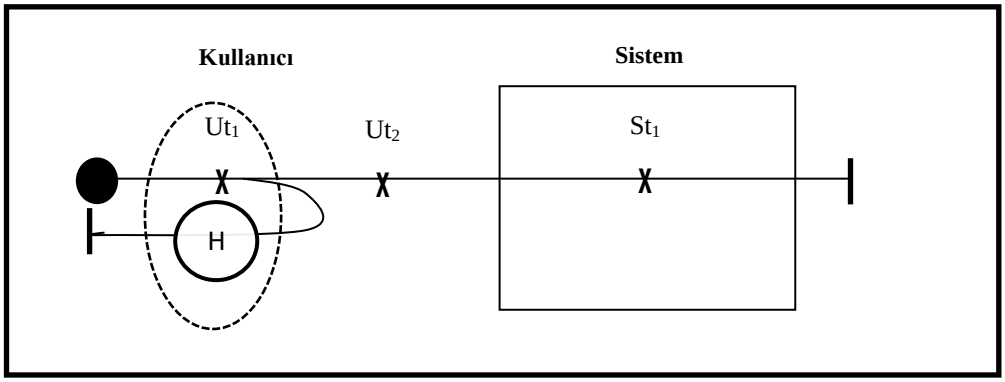
Şekil 3.11 Abort durumunun görsel yapısı

Eliptik noktalı daire yayılma alanını temsil eder; Burada bu örnekte, yayılım sadece "Ut_1" ve "Ut_2" görevlerindedir. Bu, bu görevlerdeki geri alma eylemlerinin gerekli olduğu anlamına gelir. Ayrıca, "Ut_1" görevleri tarafından yapılan bazı değişiklikler olabilir ve bu durumda bu değişiklikler menşei haline getirilmelidir (geri alma işlemi).

3.5.1.2. Durumu iptal etmek

Vakanın amacı: işlem gerçekleşmeden önce bir sürecin sona ermesidir. Örneğin, Ti işlemi başlamadan önce komple işlem iptali gerektirir (Şekil 3.12).

Bazı nedenlerden dolayı, sonlandırma, bu görev zincirinin başlamasından önce gerçekleşir. Bu, bir İptal durumu oluştuğu anlamına gelir. Bu tür bir sonlandırma, bir yayılma mekanizması gerektirmez, çünkü ilk görevlerde (sürecin başlangıcı) oluşur. Bu işlemin neden olduğu herhangi bir değişikliği geçersiz kılmak için bazı işlemleri geri almak gerekebilir (örneğin, Görev Ut_1). Katı çemberin içindeki "H" harfi bir işleyici'yi ve eliptik noktalı çember, işleme alanını temsil eder.



Şekil 3.12 İptal durumunun görsel yapısı

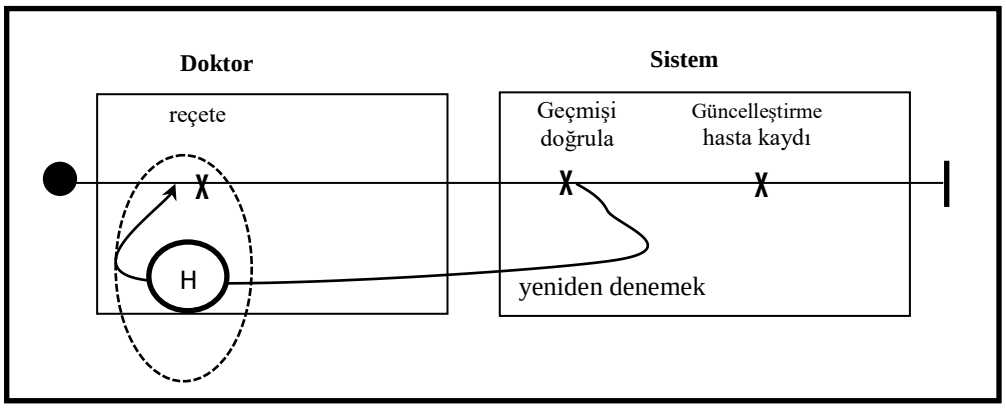
Eliptik noktalı daire yayılma alanını belirtir; Bu örnekte, yayılım gerekli olmayabilir veya gerekirse "Ut_1" görevleri içindeki bazı geri alma işlemleri gerekebilir.

3.5.2. Durumu tekrar başlatma

Bir süreci sona erdirmenin ortak durumlarından bir diğeri, durumun düzeltilmesi için bazı eylemlerin güncellenmesi ya da tekrarlanması gerektiğinden, onun başlatılmasıdır. Bu kategoride, bu durum iki geçiş durumunda tanımlanır: Özel durumların yeniden denenmesi ve yeniden yapılması (bazı alternatif eylemler yapın) durumları.

3.5.2.1. Yeniden deneme durumu

Vakanın amacı: bir sürecin devam etmesi için bir sorunla karşı karşıya kaldığında ve bu durum bazı değişiklikler veya giriş düzeltmeleri ile devam ettirilebiliyorsa. Bu, tekrar deneme işleminin yararlı olabileceği ve sürecin tamamlanmasının devam ettirilebileceği anlamına gelir. Örneğin, bir kullanıcı geçersiz veriler girer; ve bu, doğru girişi tekrar girme şansı vererek düzeltilebilir.

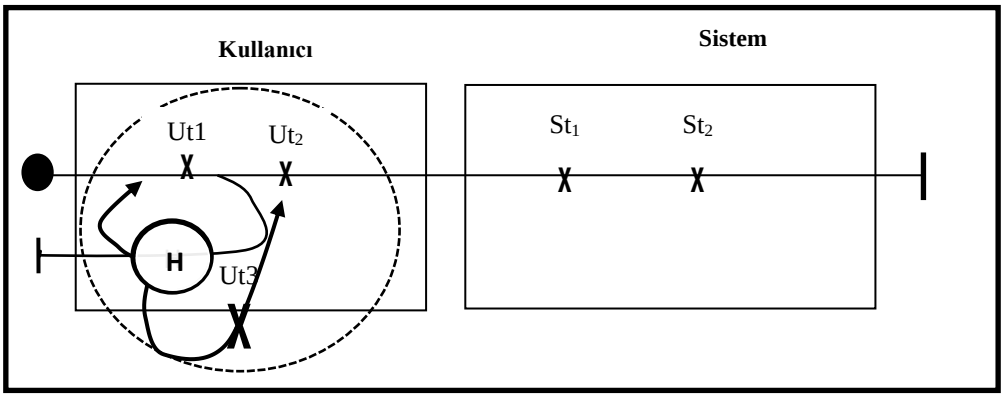


Şekil 3.13. Hasta tedavisi için yeniden deneme durumu örneği

Daha önce de belirtildiği gibi, katı dairenin içindeki "H" harfi bir işleyiciyi temsil eder ve eliptik noktalı daire ise taşıma alanını temsil eder. Bu örnekte, doktor hasta için izin verilen başka bir tıbbi dozu reçete etmek zorundadır.

3.5.2.2. Yeniden çalışma durumu

Amaç: Yeniden Dene durumuna benzer, ancak bazı durumlarda işlemi tekrar başlatmak için bazı yeniden işleme gereklidir. Örneğin; yeniden hesaplamalar veya bazı eylemler eklemek gerekir. Bu, yeniden işleme aksiyonunun yararlı olabileceği ve sürecin tamamlanmasının devam ettirilebileceği anlamına gelir. Örneğin, bir kullanıcı geçersiz verilere girer; ve bu, doğru girişi tekrar girme şansı vererek düzeltilebilir. Şekil 3.14, yeniden çalışma durum yapısının basit bir örneği verilmiştir.



Şekil 3.14. Yeniden durumuna bir örnek

Şekilde görüldüğü gibi, işleyici işlemi durdurmaya mı, yoksa harici bir olay meydana geldiğinden bazı işlemler ekleyerek devam edip etmeyeceğine karar vermelidir. Örneğin, bir yolcu, otel rezervasyonu, bilet rezervasyonu vb. gerektiren bir gezi planladığında ve yolculuğa çıkmadan önce hasta olabilir. Bu, güncelleme yapması, bazı işlemler eklemesi veya seyahati iptal etmesi konusunda karar vermesi gerektiği anlamına gelir. Daha önce gösterildiği gibi, istisna bölgesi eliptik noktalı daire tarafından temsil edilir.

Son olarak, senaryonun aşamaları kısıtlı dilin önerilen adımlarını kullanarak yazıldıktan sonra (bu normal senaryoyu temsil eder) ve önerilen algılama aracı (bu durum istisna durum raporunu temsil eder) kullanılarak İstisnalar tespit edildiğinde senaryo dokümantasyonu süreci bu kurallara uygun olarak faydalı olur (Tablo 3.6). Bu nedenle, istisnaların yayılımlarını kontrol ederek, uygun bir taşıma mekanizması netleşir.

Tablo 3.6. *Senaryo belge şablonu*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	
Ön koşul:	
Giriş ve İstisna algılama (İstisna raporu)	

3.6. Vaka analiz

Vaka analizi: Basit bir kütüphane sistemi
Bu araştırmada önerilen yaklaşımı değerlendirmek için basit bir kütüphane sistemine ilişkin bir vaka çalışması önerilmiştir. Bu vaka çalışmasının seçilmiş olması, günlük işlerden biri olarak kabul edilebilir araştırmacılar için iyi bilinen ve ayrıca faaliyetinin doğasının kolaylığı ve anlayış derecesi nedeniyle seçilmiştir.

Kütüphane, katılımcılarına (Borçlular) aşağıdakiler gibi bir dizi hizmet sunmaktadır:

- Ödünç öğe,
- Ürünü geri vermek,
- Maddenin rezerve edilmesi,
- Rezervasyon iptali

Kütüphaneci rolünün, bu sistemin yönetiminde sorumluluğu olduğu varsayılmıştır ki bu da şöyledir:

- Öğe eklemek
- Öğeyi kaldırmak
- Borçlu eklemek
- Borçlu bilgilerini güncellemek / silmek
- İadeleri almak

Dolayısıyla, bu vaka çalışması bu araştırmada önerilen yaklaşımı kullanarak uygulanmaktadır.

3.6.1 Aday ihtiyaç kümesi

Bu aktivitede, sistemin menfaat sahipleri belirlenir ve çalışmaların niteliğini tanır; Kitapların, araştırma kağıtlarının, dergilerin vb.

- ***Paydaşların kimlikleri***

Bu alt bölümde, sistemin topluluğu, sisteme yakın olan ve onunla ilgilenen kişiler (yani paydaşlar) tarafından tanımlanmaktadır. Burada rollerine göre atanacaklardır.

Bu rollerin tümü paydaşların tanım tablosunda (Tablo 3.7'de gösterildiği gibi)

Tablo 3.7. *Basit bir kütüphane sistemi için paydaşların kimlik sicili*

Paydaş adı	**Pozisyon**	**Sistem ile ilişki**
Borçlu	Katılımcı	• Ödünç alınan ürünler malzeme rezervasyonu yapmak • Rezervasyon iptali.
kütüphaneci	Kütüphane, Sistem yöneticisi	• Öğe eklemek • Öğeyi kaldırmak • Borçlu eklemek • Borçlu bilgilerini güncellemek • Güncelleme dönüşleri japmak

- ***İş belgelerinin incelenmesi***

Yazılım geliştiricileri ve potansiyel paydaşları arasındaki müzakere sürecini iyileştirmek; çalışma belgeleri ve oynadıkları ve sorumlu oldukları roller aracılığıyla yapabilecekleri çalışmalarının doğasını anlamaktır (Tablo 3.8 ve 3.9 sırasıyla Borçlu ve Kütüphanecinin sistemdeki rollerini gösterir).

Tablo 3.8. *Borçlu'nun çalışma sorumlulukları örneği*

Kullanıcı rolü adı	Borçlu
Rol sorumlulukları	İş kuralı (lar) ve kısıtları
Ürün Rezervasyonu	Kredinin gerçekleşmesinden önce ödünç almak için gerekli kalemi ayırtın
Borç almak	Ödünç alma, rezervasyon işleminin tamamlanmasından sonra
Rezervasyon iptali	Rezervasyonu geri al ve iptal et

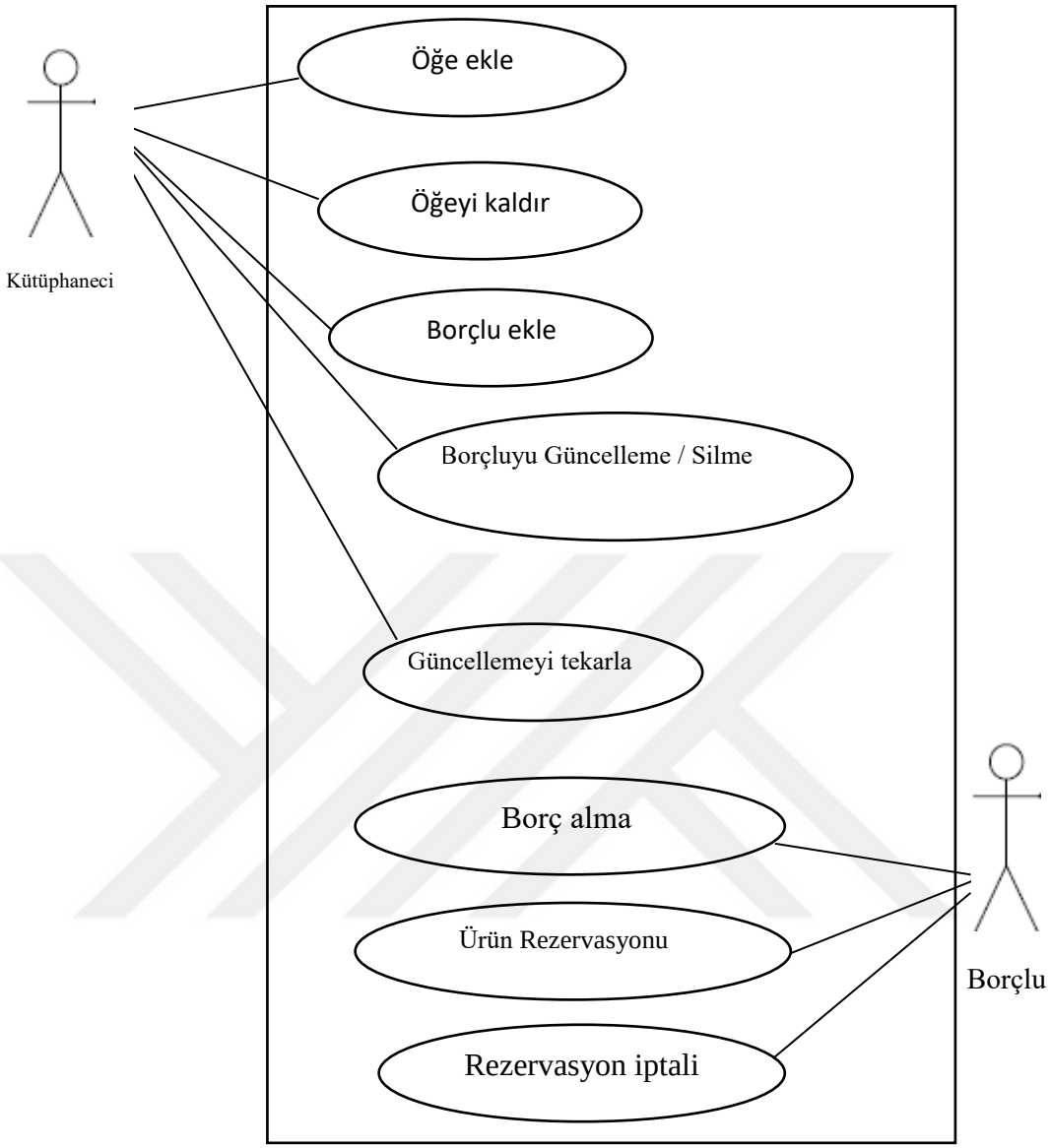
Tablo 3.9. *Kütüphanecinin çalışma sorumlulukları örneği*

Kullanıcı rolü adı	Kütüphaneci
Rol sorumlulukları	İş kuralı (lar) ve kısıtları
Öğe Ekleme	Kütüphaneye yeni öğeler ekleme süreci
Öğeyi kaldırma	Öğeleri kitaplıktan silme işlemi
Borçlu ekleme	Abone olmaya hak kazandıklarından emin olduktan sonra kütüphaneye yeni aboneler eklemek
Borçlunun bilgilerini güncelleme	Kütüphane sistemindeki katılımcı bilgilerini silin veya güncelleyin
Rezervasyonu iade etme	İade tarihi ve öğenin durumu gibi iade bilgilerini kontrol edin

3.6.2. Aday kullanıcı gereksinimleri belgeleri

- ***Kullanıcı rolü doğrulaması***

Bu aktivitede, aday görevleri ilgili kullanıcı rolleri ile birleştirilir. Rollerin sorumlulukları, kullanım durumu diyagramında kullanım örneği olarak kabul edilir. Kullanıcı rolleri sistemde aktörler olarak görülürken, önerilen kütüphane sistemindeki aktörler kütüphaneci ve borçludur (Şekil 3.15). Vaka incelemesini basitleştirmek için, yalnızca önemli kullanım durumları yaklaşımımızla gösterilmek üzere seçilmiştir.



Şekil 3.15 Basit bir Kütüphane sisteminin kullanım durumu diyagramı

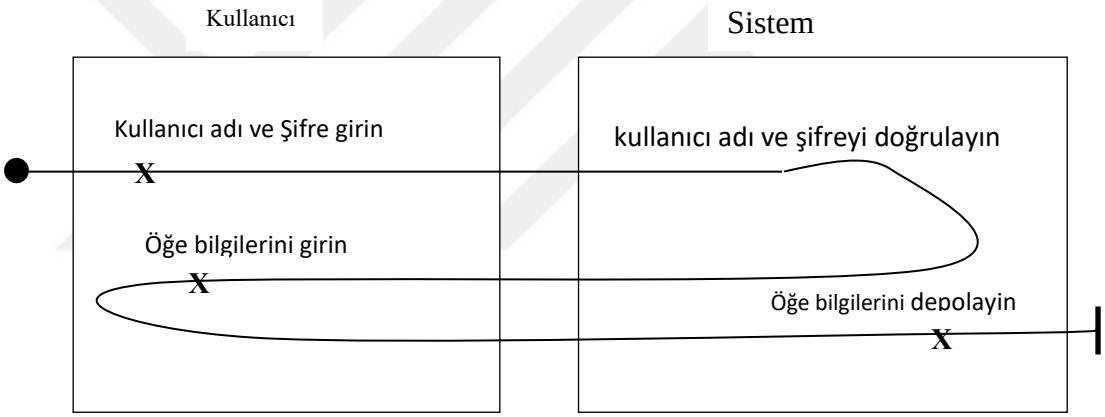
- ***Kullanıcı gereksinimlerini görselleştirmek***

Kullanım örneği: Öğe ekle

Senaryo, alınan maddelerin kütüphaneye eklenmesinden sorumlu kütüphaneci ile sistemin operasyonlarının gerçekleştirilmesinden sorumlu kütüphane sistemi arasında yer almaktadır. Tablo 3.10, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.16 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.10. *Kullanım örneği "Kısıtlı dil kullanarak öğe ekle"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Öğe ekle
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci kütüphane sistemine ürün eklemek istiyor.
Olayların normal akışı	
U: Enter user name and Password S: validate user name and password U: Enter Item's information S: Store Item's information	U :Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi doğrulayın U: Öğe bilgilerini girin S: Öğe bilgilerini depolayın



Şekil 3.16. "Öğe ekle" kullanım durumu UCM'

Kullanıcı bileşeni, kitaplığa öğelerin eklenmesinden sorumlu kütüphane sistemindeki Kütüphaneciyi temsil eder ve ikinci bileşen kütüphane sistemidir (diğer bir deyişle uygulama). Kütüphaneci giriş durumundayken kütüphane sistemine öğe eklemek isterse, sistem bu işlemin yetkisini yerine getiren kullanıcı adı ve parolasını denetler. Kullanıcı yetkili bir kişiyse, öğenin bilgilerine girer ve sistem bu bilgiyi kütüphane sisteminde saklar.

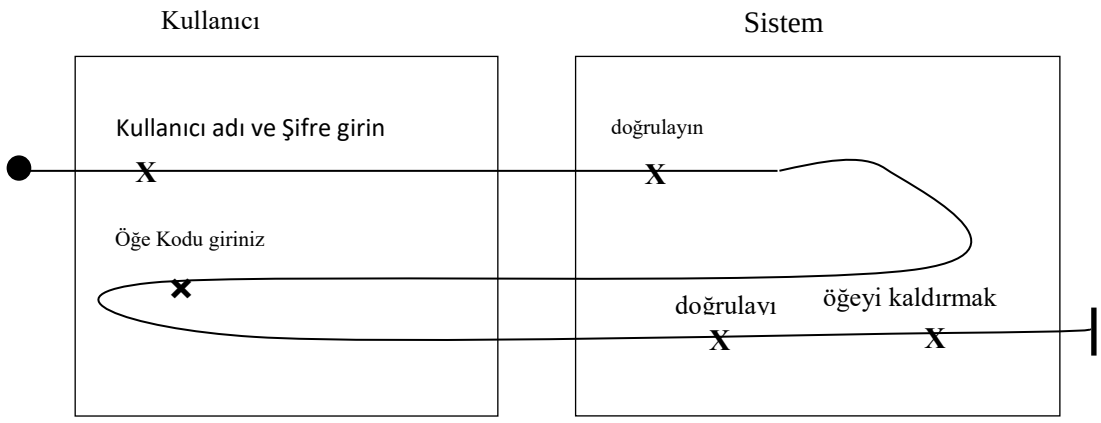
Kullanım örneği: öğeyi kaldır

Senaryo, kütüphaneden öğelerin silinmesinden sorumlu olan kütüphaneci ile sistemin operasyonlarının gerçekleştirilmesinden sorumlu olan kütüphane sistemi arasında

bulunur. Tablo 3.11, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.17 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.11. *Kullanım örneği "öğeyi kaldır"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	öğeyi kaldır
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor
Olayların normal akışı	
U: Enter user name and Password S: validate user name and password U: Enter Item's code S: Validate Item's information S: Remove Item's information	U :Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi doğrulayın U: Öğe Kodu girin S: Öğe bilgilerini doğrulayın S: Öğe bilgilerini kaldırın



Şekil 3.17 "Öğeyi kaldır" kullanım durumu UCM'

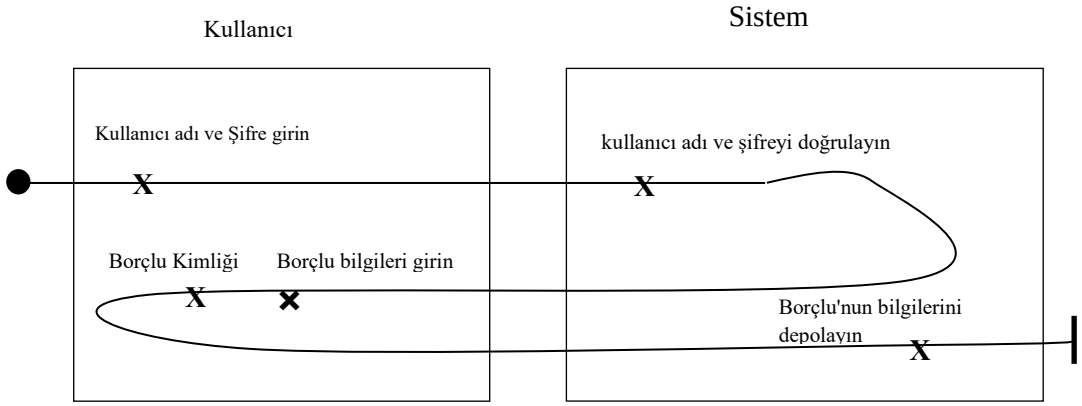
Kullanıcı bileşeni, kitaplıktan öğelerin kütüphaneden kaldırılmasından sorumlu kütüphane sistemindeki Kütüphaneciyi temsil eder ve ikinci bileşen kütüphane sistemidir (diğer bir deyişle uygulama). Kütüphaneci giriş durumundadır, s (o) bir öğeyi kütüphane sisteminden kaldırmak ister ve sistem, kaldırma işleminin yetkisini sağlayacak kullanıcı adı ve parolasını denetler. Kullanıcı yetkili bir kişiyse, öğenin kodunu girer ve sistem öğe bilgilerini kütüphane sisteminden kaldırır.

Kullanım örneği: Borçlu ekle

Senaryo, kütüphane sistemine borçlu eklemeden sorumlu olan kütüphaneci ile sistemin operasyonlarının yürütülmesinden sorumlu kütüphane sistemi arasında yer almaktadır. Tablo 3.12, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.18 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.12. *Kullanım örneği "Sınırlı Dili Kullanarak Borçlu Ekle"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Borçlu ekle
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor
Olayların normal akışı	
U: Enter user name and Password S: validate user name and password U: Enter Borrower ID U: Enter Borrower information S: Store Borrower's information	U :Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi doğrulayın U: Borçlu Kimliği Girin U: Borçlu bilgileri girin S: Borçlu'nun bilgilerini depolayın



Şekil 3.18 "Borçlunun Eklenmesi" kullanım durumu UCM'

Kullanıcı bileşeni, yeni Borçluların kütüphane sistemine eklenmesinden sorumlu olan kütüphane sistemindeki Kütüphaneciyi temsil eder ve ikinci bileşen kütüphane sistemidir (diğer bir deyişle uygulama). Kütüphaneci giriş durumundayken kütüphane sistemine öğe eklemek ister, sistem bu işlemin yetkisini yerine getiren

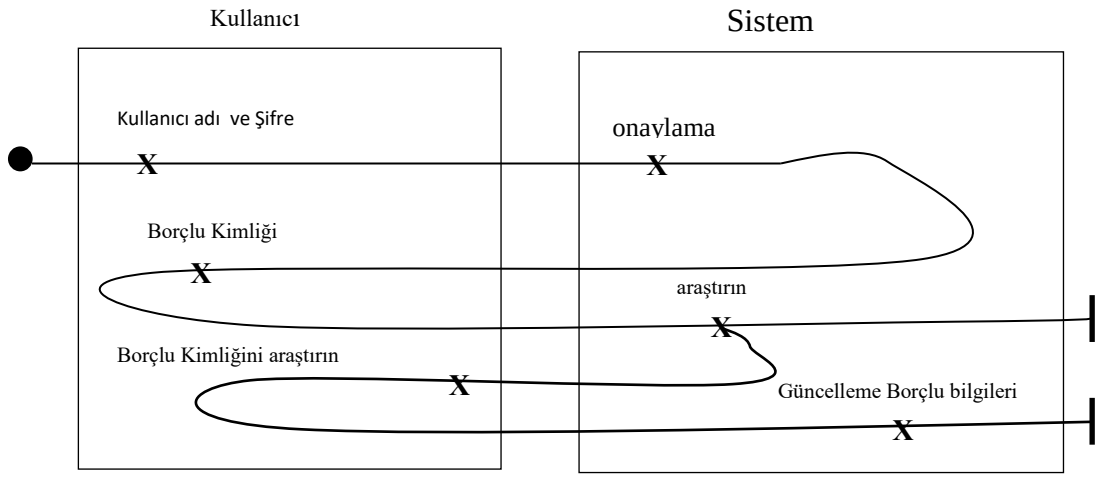
kullanıcı adı ve parolasını denetler. Kullanıcı yetkili bir kişiyse, Borçlunun bilgilerine girer ve sistem bu bilgiyi kütüphane sisteminde saklar.

Kullanım örneği: Borçluyu Güncelleme / Silme

Senaryo, kütüphane sistemindeki borçluların güncellenmesi / silinmesinden sorumlu olan kütüphaneci ile sistemin operasyonlarının yürütülmesinden sorumlu olan kütüphane sistemi arasında yer almaktadır. Tablo 3.13, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.19 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.13. *Kullanım örneği "Sınırlı dili kullanarak Borçluyu güncelle / sil"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Borçluyu Güncelleme / Silme
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor
Olayların normal akışı	
U: Enter user name and Password S: validate user name and password U: Enter Borrower ID U: Enter Borrower information to update S: Search Borrower ID S: Update Borrower's information	U: Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi doğrulayın U: Borçlu Kimliği Girin U: Güncellemek için Borçlu bilgilerini girin S: Borçlu Kimliğini araştırın S: Borçlu bilgilerini güncelleyin



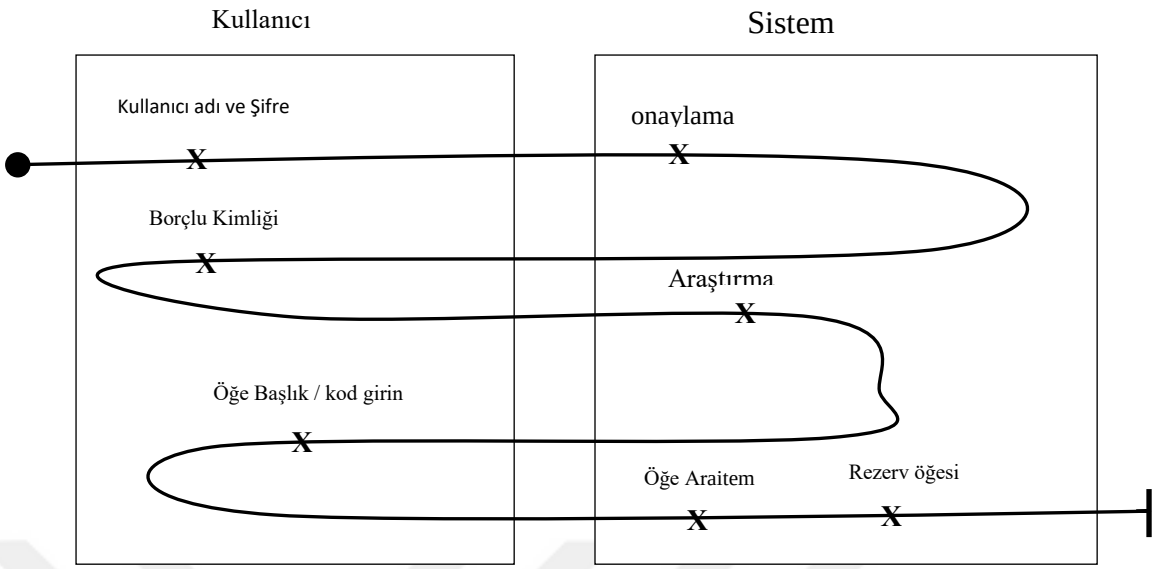
Şekil 3.19. "Borçlunun Güncelleştirilmesi / Silinmesi" kullanım durumu UCM'

Kullanım örneği: Rezerv madde

Senaryo, borç alanların talebi üzerine kayıt unsurları çekincelerinden sorumlu olan kütüphaneci ile sistemin operasyonlarının yürütülmesinden sorumlu kütüphane sistemi arasında yer almaktadır. Tablo 3.14, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.20 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.14. *Kullanım örneği "Rezerv madde"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Rezerv madde
Ön koşul:	Madde rezervasyon talebi
Olayların normal akışı	
U: Enter user name and Password S: validate user name and password U: Enter Borrower ID S: Search Borrower ID U: Enter Item title/Code S: Search item S: Reserve item	U: Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi U: Borçlu Kimliği Girin S: Borçlu Kimliğini araştırız U: Öğe Başlık / kod girin doğrulayın S: Araştırma öğesi S: Rezerv öğesi



Şekil 3.20 "Rezerv madde" kullanım durumu UCM'

Kullanıcı bileşeni, rezervasyon bilgisinin depolanmasından sorumlu olan bir Borçluyu temsil eder ve ikinci bileşen kütüphane sistemidir (yani, uygulama). Borçlu giriş durumundayken, kullanıcı adı ve şifresini kütüphane sistemini kullanarak girdiğinde, sistem bu sürecin yetkisini sağlayan kullanıcı adını ve şifresini kontrol eder. Borçlu yetkili ise, kimlik belgesi girer. Ardından sistem borçlu ve yetkili Borçlu'yu arar, sistem Borçlu'ya gerekli öğenin başlığını veya kodunu girmesini ister ve nihayet sistem öğe mevcutsa rezervasyonu depolar.

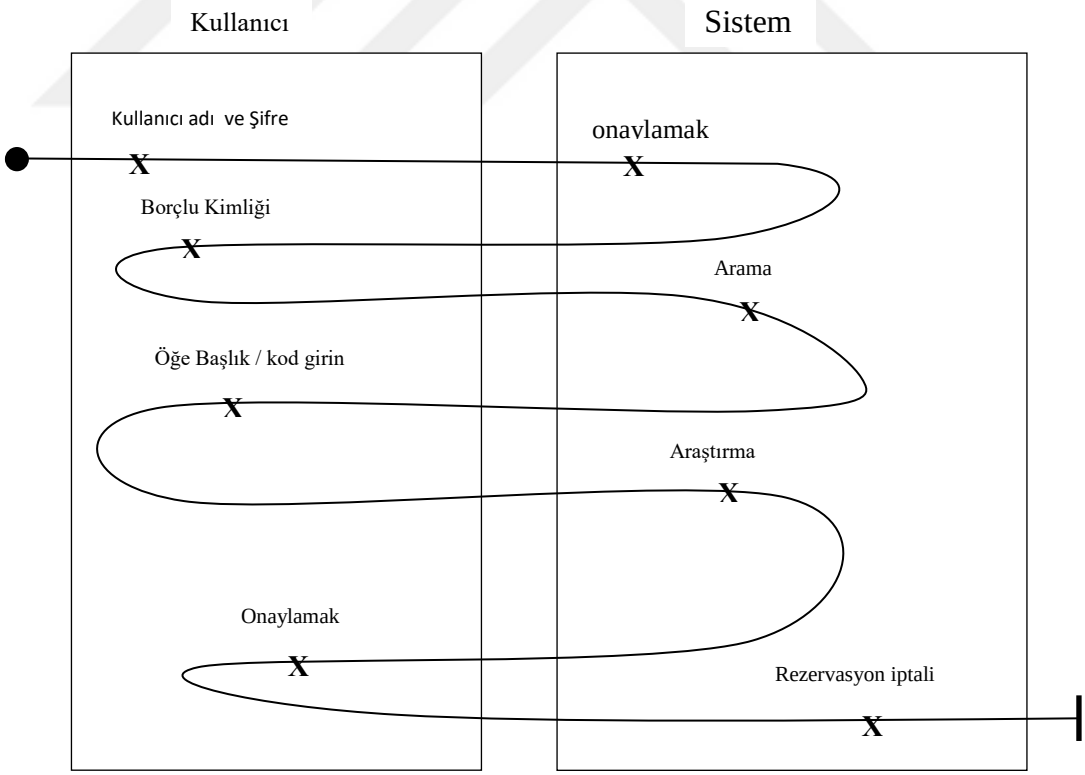
Kullanım örneği: Rezervasyon iptal

Senaryo, borçluların talebi üzerine öğelerin rezervasyonlarını iptal etme sorumluluğunu üstlenen kütüphaneci ile sistemin operasyonlarının yürütülmesinden sorumlu kütüphane sistemi arasında yer almaktadır.

Tablo 3.15, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.21 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.15. *Kullanım örneği "Kısıtlı dil kullanarak rezervasyon iptal"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Rezervasyon iptal
Ön koşul:	Borçlu rezervasyon iptal etmek istiyor
Olayların normal akışı	
U: Enter user name and Password S: Validate user name and password U: Enter Borrower ID S: Search Borrower ID U: Enter Item title/Code S: Search item U: Confirm cancelation S: Cancel reservation	U: Kullanıcı adı ve Şifre gir S: kullanıcı adı ve şifreyi doğrula U: Borçlu Kimliği Gir S: Borçlu Kimliğini araştır U: Öğe Başlık / kod gir S: Arama öğesi U: İptal işlemini onayla S: Rezervasyonu iptal et



Şekil 3.21. "Rezervasyon İptal Et" kullanım durumu UCM'

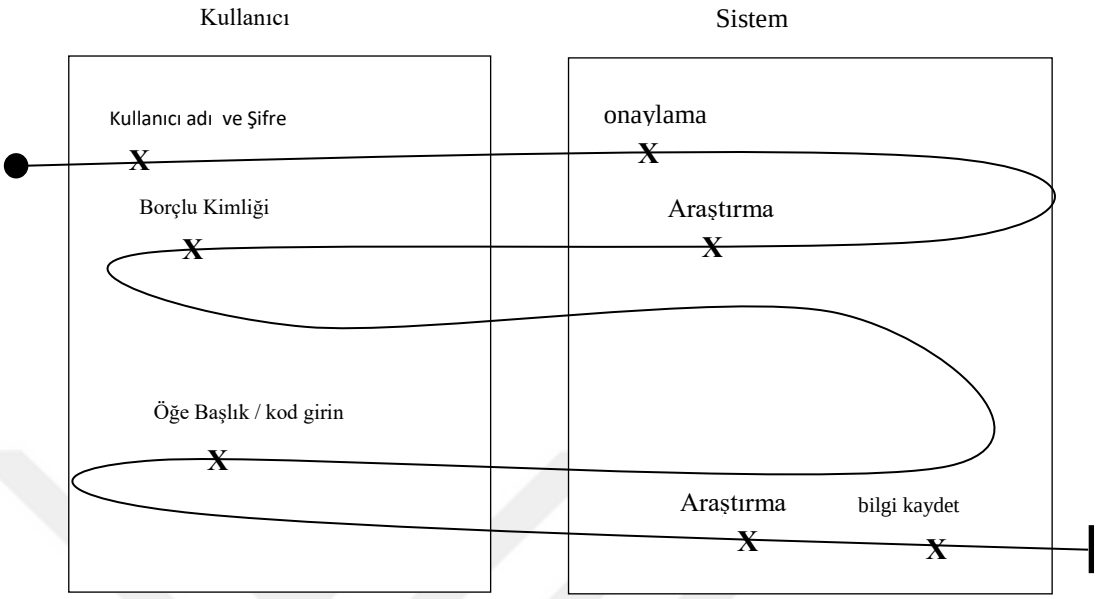
Kullanıcı bileşeni, rezervasyon bilgisinin iptal edilmesinden sorumlu kitaplık sistemindeki Borçlu'yu ve ikinci bileşen kütüphane sistemini (yani başvuru) belirtir. Borçlu giriş durumundayken, kullanıcı adı ve şifresini kütüphane sistemini kullanarak girdiğinde, sistem bu işlemin yetkisini sağlayan kullanıcı adını ve şifresini kontrol eder. Borçlu yetkili ise, kimlik belgesi girer. Daha sonra sistem borçluyu arar ve eğer yetkili bir Borçlu ise, sistem Borçlu'dan gerekli kalemin adını veya kodunu girmesini isterse sistem gerekli öğeyi arar. Nihayet, Borçlu, iptali onaylar; sistem, ayrılmış kalemin bilgilerini iptal eder.

Kullanım örneği: Ödünç öğe

Senaryo, kütüphane sistemindeki borç verenlere borç verme sorumluluğunu yükleyen kütüphaneciden ve sistemin operasyonlarının gerçekleştirilmesinden sorumlu olan kütüphane sisteminden oluşur. Tablo 3.16, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu ve Şekil 3.22 UCM'leri kullanarak oluşturulan görsel senaryosunu göstermektedir.

Tablo 3.16. *Kullanım örneği "Ödünç öğe"*

<table>
<tr><th colspan="3">Kısıtlı dil kullanan senaryo dökümantasyonu</th></tr>
<tr><td>Kullanım durumunun adı</td><td colspan="2">Ödünç öğe</td></tr>
<tr><td>Ön koşul:</td><td colspan="2">Ödünççe isteği</td></tr>
<tr><th colspan="3">Olayların normal akışı</th></tr>
<tr><td colspan="2">U: Enter user name and Password
S: validate user name and password
U: Enter Borrower ID
S: Search Borrower ID
U: Enter Item title/Code
S: Search item
S: Store lend information</td><td>U: Kullanıcı adı ve Şifre girin
S: kullanıcı adı ve şifreyi doğrulayın
U: Borçlu Kimliği Girin
S: Arama Borçlu Kimliği
U: Öğe Başlık / kod girin
S: Arama öğesi
S: Borç bilgilerini kaydet</td></tr>
</table>



Şekil 3.22 "Ödünç öğe" kullanım durumu UCM'

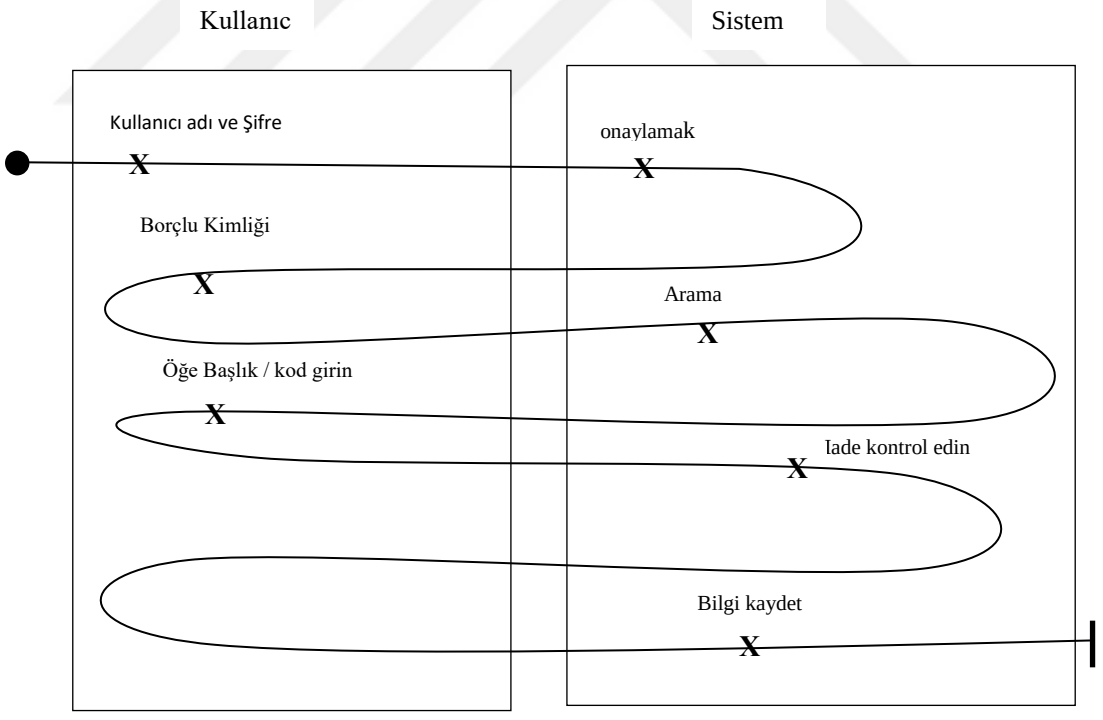
Kullanıcı bileşeni, borç verme bilgilerinin depolanmasından sorumlu olan kütüphane sisteminde bir Kütüphaneciyi temsil eder.ve ikinci bileşen kütüphane sistemidir (yani uygulama). Kütüphaneci giriş durumundayken, kullanıcı adı ve şifresini kütüphane sistemini kullanarak girdiğinde, sistem bu işlemin yetkisini yerine getiren kullanıcı adını ve şifreyi denetler. Kütüphaneci yetkili kişi ise, Borçlu'nun kimlik kartını girer. Ardından sistem borçluyu araştırır, sistem kütüphaneciden istenen öğenin başlığını veya kodunu girmesini ister ve nihayet sistem öğe mevcutsa ödünç alınan öğenin bilgilerini arar ve saklar.

Kullanım örneği: Güncelleme döner

Bu senaryo kütüphane sisteminin borçlularından iade edilen kalemleri almaktan sorumlu kütüphaneci ile sistem faaliyetlerini yürütmekten sorumlu kütüphane sistemi arasındadır. Tablo 3.17, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu ve Şekil 3.23 UCM'leri kullanarak elde edilen görsel senaryosunu göstermektedir.

Tablo 3.17. *Kullanım örneği "Güncelleme döner"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Güncelleme döner
Ön koşul:	Güncelleme iade talebi
Güncelleme iade talebi	
U: Enter user name and Password S: validate user name and password U: Enter Borrower ID S: Search Borrower ID U: Enter Item title/Code S: Check returns S: Store returns information	U: Kullanıcı adı ve Şifre girin S: kullanıcı adı ve şifreyi doğrulayın U: Borçlu Kimliği Girin S: Borçlu Kimliğini araştırın U: Öğe Başlık / kod girin S: İadeyi kontrol edin S: Bilgi kaydet



Şekil 3.23. "Güncellemeleri döndür" kullanım durumu UCM'

Kullanıcı bileşeni, geri gönderilen öğelerin saklanmasından sorumlu kitaplık sistemindeki bir Kütüphaneci'yi ve ikinci bileşen kütüphane sistemini (yani, başvuruyu) temsil eder.

Kütüphaneci yetkili kişi ise, ödünç alan kişinin kimlik kartını girer. Daha sonra sistem borçlanıcıyı arar, sistem kütüphanecinin soyadını veya iade edilen öğenin kodunu girmesini ister, ardından sistem iade edilen ürün bilgisini kontrol eder ve son olarak öğenin iade bilgisini depolar.

- ***Süreç Hedefi modeli***

Önceden açıklandığı gibi, bazı kullanıcı gereksinimleri diğer gereksinimlerle entegre edilmelidir çünkü bir kez monte edildiğinde elde edilmesi gereken tek bir tamamlanmış hedefi temsil eder. Örneğin, bir öğe satın almak iki temel işlemden oluşabilir: Satın alınacak öğeyi seçmek (seçim işlemi başarılı olacak şekilde seçilmesi gereken birkaç adım gerekebilir) ve daha sonra satın alma işleminin tamamlanması. Bu adımlar, "bir ürün satın al" diyebileceğimiz bir hedefi temsil eder.

Dolayısıyla, vaka çalışması ile ilgili olarak, hedef model tablo 3.18 de gösterilebilir.

Tablo 3.18. *Vaka analizinin Süreç Hedef modeli*

Hedef	**Kullanıcı gereksinimleri**	**Kullanıcı rolü**
Öğe ekle	Öğe ekle	Kütüphaneci
Öğeyi kaldır	Öğeyi kaldır	Kütüphaneci
Borçlu ekle	Borçlu ekle	Kütüphaneci
Borçluyu Güncelleme / Silme	Borçluyu Güncelleme / Silme	
Borç alma	Rezerv madde	Borçlu
	Borç alma	
Rezervasyon iptali	Rezervasyon iptali	Borrower
Güncelleme döner	Güncelleme döner	Kütüphaneci

Bu tablo, rezervasyon ve ödünç vermenin iki tamamlayıcı süreç olduğu, yani "borç verme öğesi" adı verilen hedefin tamamlandığığını açıkça gösterir.

3.6.3. İstisnalar algılama mekanizması

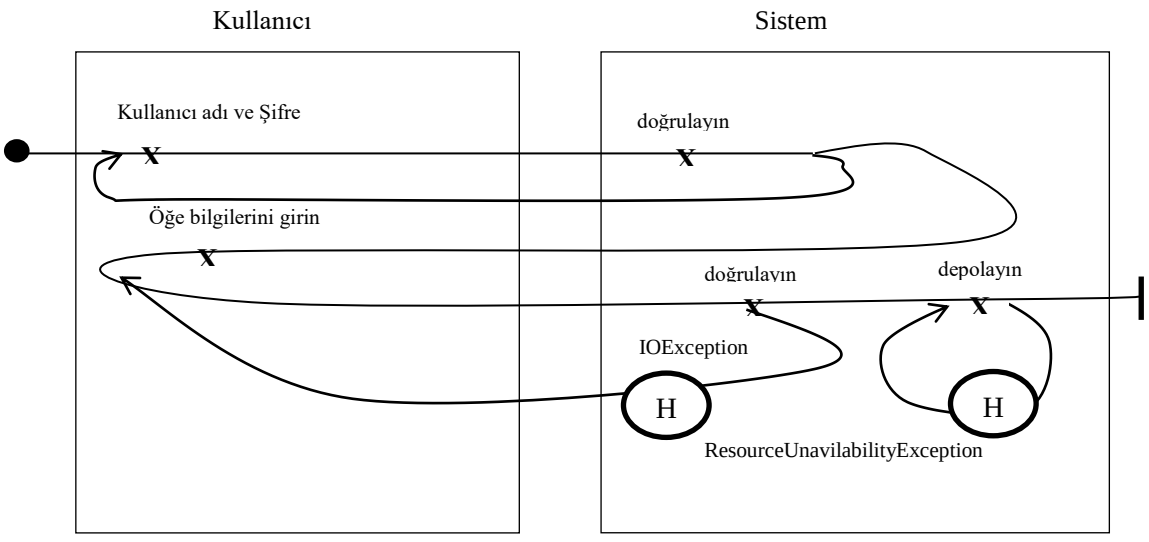
Tablo 3.19, istisnai durumları da dahil olmak üzere kullanım durumunun dokümantasyonunu göstermektedir ve Şekil 3.24 UCM'leri kullanarak görsel senaryosunu göstermektedir.

Tablo 3.19. *Durum kılıfı " Öğe ekleme " ile istisnalar*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Giriş penceresinde kütüphaneci. Kütüphaneci kütüphane sistemine ürün eklemek istiyor.
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci kütüphane sistemine ürün eklemek istiyor.

```
                                Type Exceptions Report
                                ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Item's information
      S: store Item's information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Item's information
      User Exception: IOException

      S: store Item's information
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
{
    U :Kullanıcı adı ve Şifre girin
    S: kullanıcı adı ve şifreyi doğrulayın
    U: Öğe bilgilerini girin
    S: Öğe bilgilerinizi depolayın
}

Yakalamak (Istisna E)
{
    U :Kullanıcı adı ve Şifre girin
    Kullanıcı İstisnası: IOException

    U: Öğe bilgilerini girin
    Kullanıcı İstisnası: IOException

    S: Öğe bilgilerinizi depolayın
    Sistem İstisnası: ResourceUnavilabilityException

}
```



Şekil 3.24. "Öğe ekleme" UCM, istisnalar içeren kullanım örneği

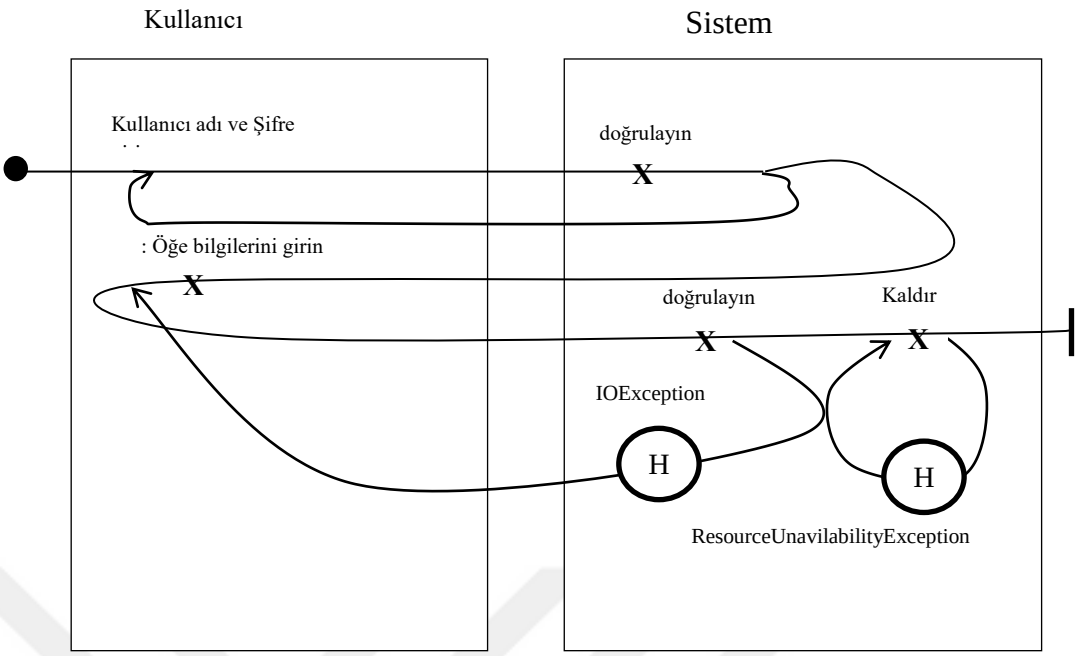
Tablo 3.20, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonu verilmiştir ve Şekil 3.25 UCM'leri kullanarak oluşturulan görsel senaryosu verilmiştir.

Tablo 3.20. *Kullanım örneği "Öğe kaldır" ile istisnalar*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Öğe kaldır
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor

```
                                    Type Exceptions Report
                                    ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Item's information
      S: Validate Item's information
      S: Remove Item's information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Item's information
      User Exception: IOException

      S: Validate Item's information
      System Exception: IOException

      S: Remove Item's information
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
{
    U :Kullanıcı adı ve Şifre girin
    S: kullanıcı adı ve şifreyi doğrulayın
    U: Öğe bilgilerini girin
    S: Öğenin bilgilerini doğrulayın
    S: Öğe bilgilerini kaldır

}
Yakalamak (Istisna E)
{
    U :Kullanıcı adı ve Şifre girin
    Kullanıcı İstisnası: IOException

    U: Öğe bilgilerini girin
    Kullanıcı İstisnası: IOException

    S: Öğenin bilgilerini doğrulayın
    Sistem İstisnası: IOException

    S: Öğe bilgilerini kaldır
    Sistem İstisnası: ResourceUnavilabilityException
}
```



Şekil 3.25 "Kaldır öğe" UCM, istisnalar içeren kullanım örneği

Kullanım örneği: Borçlu Ekle

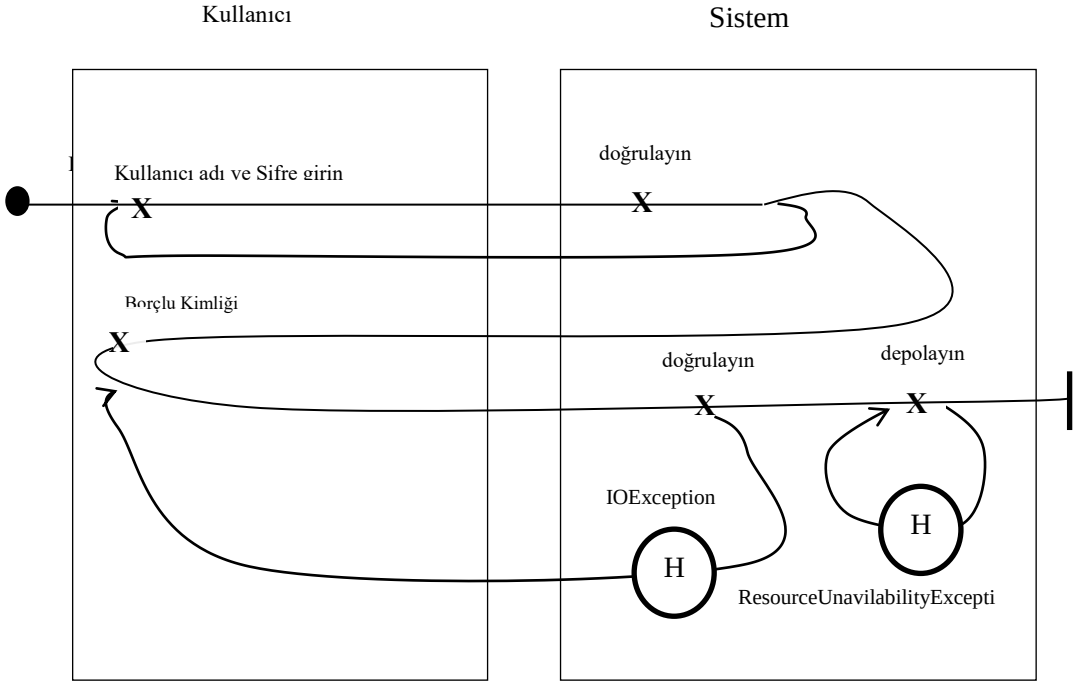
Tablo 3.21, istisnai durumları da dahil olmak üzere kullanım durumunun dokümantasyonunu ve Şekil 3.26 UCM'leri kullanarak elde edilen görsel senaryoyu gösterir.

Tablo 3.21. *Kullanım örneği "İsteğe bağlı Borçlayıcı Ekle"*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Borçlu ekle
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor

```
                                    Type Exceptions Report
                                    ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Borrower ID
      U: Enter Borrower information
      S: store Borrower's information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      U: Enter Borrower information
      User Exception: IOException

      S: store Borrower's information
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
{
    U :Kullanıcı adı ve Şifre girin
    S: kullanıcı adı ve şifreyi doğrulayın
    U: Borçlu Kimliği Girin
    U: Borçlu bilgileri girin
    S: Borçlu'nun bilgilerini depolayın

}

Yakalamak (Istisna E)
{
    U :Kullanıcı adı ve Şifre girin
    Kullanıcı İstisnası: IOException

    U: Borçlu Kimliği Girin
    Kullanıcı İstisnası: IOException

    U: Öğenin bilgilerini doğrulayın
    Kullanıcı İstisnası: IOException

    S: Borçlu'nun bilgilerini depolayın
    Sistem İstisnası: ResourceUnavilabilityException
}
```



Şekil 3.26. "Borçlu Ekle" UCM, istisnalar içeren kullanım örneği

- Kullanım örneği: Borçluyu Güncelleme / Silme

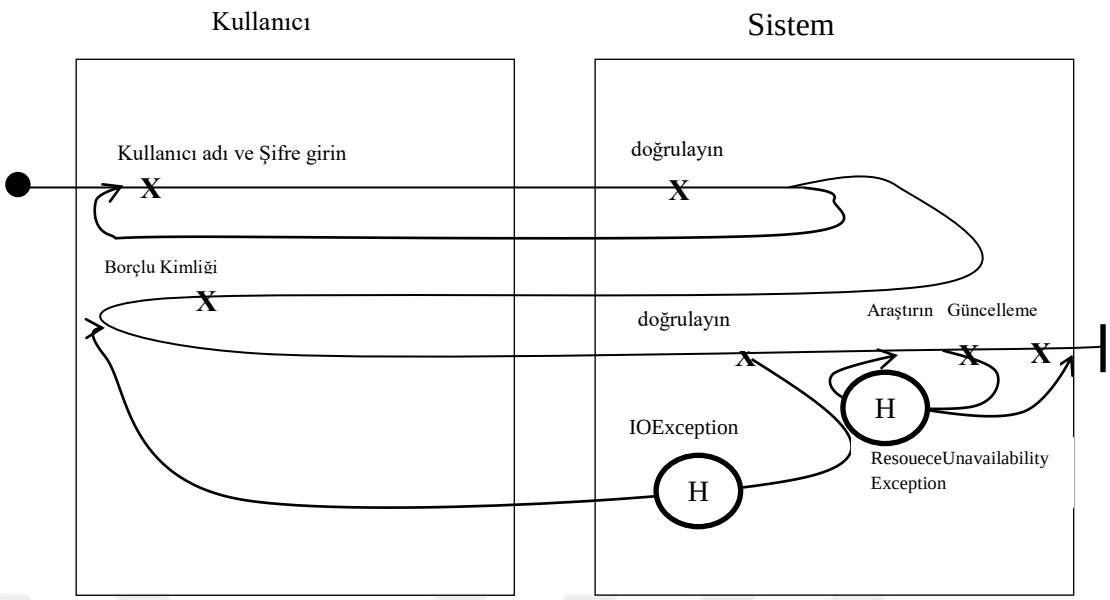
Tablo 3.22, istisnai durumları da dahil olmak üzere kullanım durumunun dokümantasyonunu ve Şekil 3.27, UCM'lerin kullanılmasıyla elde edilen görsel senaryosunu göstermektedir.

Tablo 3.22. *Kullanım örneği "İsteğe bağlı Borçlunun Güncellenmesi / Silinmesi" istisnalarla*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Borçluyu Güncelleme / Silme
Ön koşul:	Giriş penceresinde kütüphaneci. Kütüphaneci bir öğeyi kütüphane sisteminden kaldırmak istiyor

```
                    Type Exceptions Report
                    ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Borrower ID
      U: Enter Borrower information
      S: Search Borrower ID
      S: Update/Delete Borrower's information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      U: Enter Borrower information
      User Exception: IOException

      S: Search Borrower ID
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
 {
    U: Kullanıcı adı ve Şifre girin
    S:  kullanıcı adı ve şifreyi doğrulayın
    U: Borçlu Kimliği Girin
    U: Güncellemek için Borçlu bilgilerini girin
    S: Borçlu Kimliğini araştırın
    S: Güncelleme Borçlu bilgileri

 }

Yakalamak (Istisna E)
 {
      U :Kullanıcı adı ve Şifre girin
      Kullanıcı İstisnası: IOException

      U: Borçlu Kimliği Girin
      Kullanıcı İstisnası: IOException

      U: Öğenin bilgilerini doğrulayın
      Kullanıcı İstisnası: IOException


      S: Borçlu Kimliğini araştırın
      Sistem İstisnası: ResourceUnavilabilityException
}
```



Şekil 3.27 " Borçlunun Güncellenmesi / Silinmesi " UCM, istisnalar içeren kullanım örneği

- Kullanım örneği: Rezerv madde

Tablo 3.23, istisnai durumları da dahil olmak üzere bu kullanım durumunun belgelerini ve Şekil 3.28 UCM'leri kullanarak oluşturulan görsel senaryoyu gösterir.

Tablo 3.23. *Kullanım örneği "Rezerv madde" istisnalarla*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Rezerv madde
Ön koşul:	Madde rezervasyon talebi

```
                                Type Exceptions Report
                                ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Borrower ID
      S: Search Borrower
      U: Enter Item title/Code
      S: Search item
      S: Reserve item
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      S: Search Borrower
      System Exception: ResourceUnavailbilityException

      U: Enter Item title/Code
      User Exception: IOException

      S: Search item
      System Exception: ResourceUnavailbilityException

      S: Reserve item
      System Exception: ResourceUnavailbilityException

  }
```

Deneyin
{
U: Kullanıcı adı ve Şifre girin
S: kullanıcı adı ve şifreyi doğrulayın
U: Borçlu Kimliği Girin
S: Borçlu Kimliğini araştırın
U: Öğe Başlık / kod girin doğrulayın
S: Araştırma öğesi
S: Rezerv öğesi

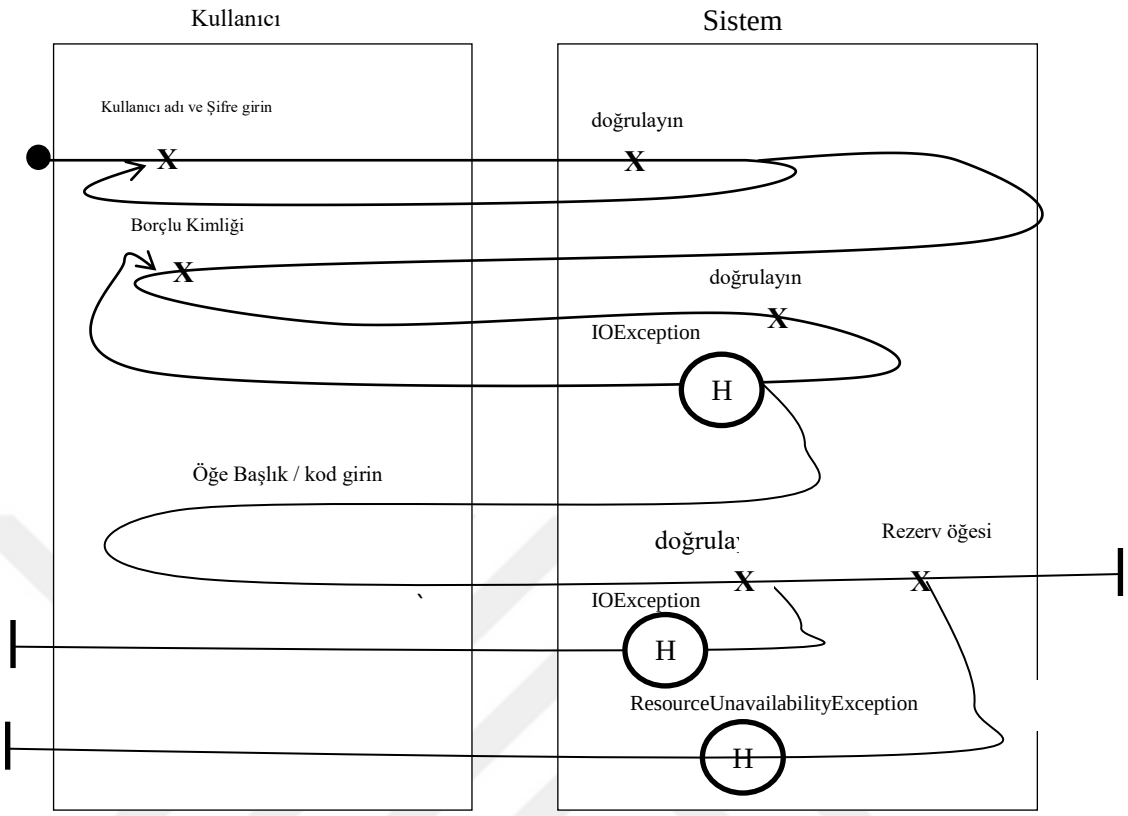
}

Yakalamak (Istisna E)
{
U :Kullanıcı adı ve Şifre girin
Kullanıcı İstisnası: IOException

U: Borçlu Kimliği Girin
Kullanıcı İstisnası: IOException

S: Borçlu Kimliğini araştırın
Kullanıcı İstisnası: ResourceUnvailabilityException

S: Rezerv öğesi
Sistem İstisnası: ResourceUnavilabilityException
}



Şekil 3.28 "Rezerv madde" UCM, istisnalar içeren kullanım örneği

- Kullanım örneği: Rezervasyon iptal

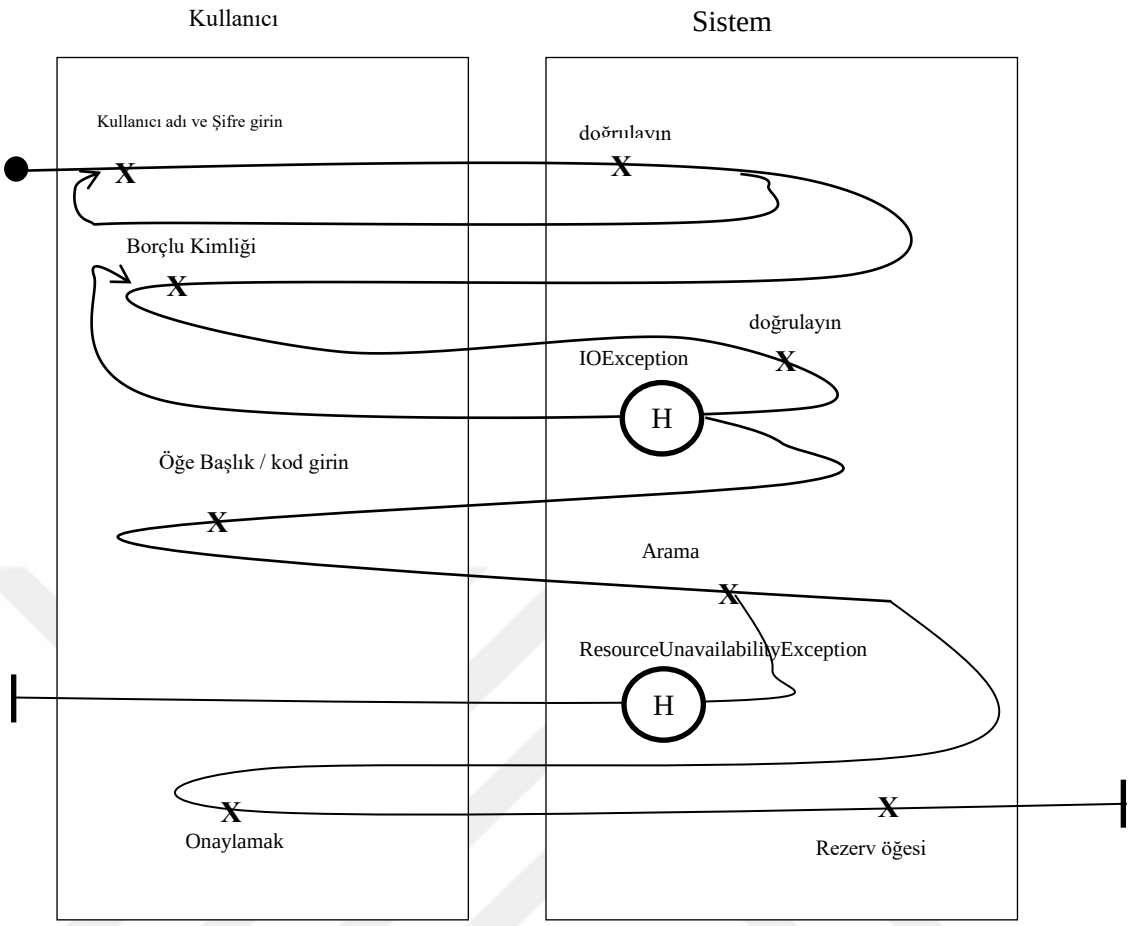
Tablo 3.24, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu ve Şekil 3.29, UCM'leri kullanarak elde edilen görsel senaryosunu göstermektedir.

Tablo 3.24. *Kullanım örneği "Rezervasyon iptal" istisnalarla*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Rezervasyon iptal
Ön koşul:	Borçlu rezervasyon iptal etmek istiyor

```
                                Type Exceptions Report
                                ======================
Try
  {
      U: Enter user name and Password
      S: Validate user name and password
      U: Enter Borrower ID
      S: Search Borrower
      U: Enter Item title/Code
      S: Search item
      U: Confirm cancelation
      S: Cancel reservation
  }

Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      S: Search Borrower
      System Exception: ResourceUnavailbilityException

      U: Enter Item title/Code
      User Exception: IOException

      S: Search item
      System Exception: ResourceUnavailbilityException

      U: Confirm cancelation
      User Exception: The user cancels out of the use case

      S: Cancel reservation
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
 {
     U: Kullanıcı adı ve Şifre girin
     S:  kullanıcı adı ve şifreyi doğrulayın
     U: Borçlu Kimliği Girin
     S: Borçlu Kimliğini araştırız
     U: Öğe Başlık / kod girin
     S: Arama öğesi
     U: İptal işlemini onayla
     S: Rezervasyonu iptal et
 }

Yakalamak (Istisna E)
 {
     U :Kullanıcı adı ve Şifre girin
     Kullanıcı İstisnası: IOException

     U: Borçlu Kimliği Girin
     Kullanıcı İstisnası: IOException

     S: Borçlu Kimliğini araştırın
     Kullanıcı İstisnası: ResourceUnvailabilityException

     U: Öğe Başlık / kod girin
     Kullanıcı İstisnası: IOException

     S: Arama öğesi
     Sistem İstisnası: ResourceUnavilabilityException

     U: İptal işlemini onayla
     Kullanıcı İstisnası: IptalException

    S: Rezerv öğesi
     Sistem İstisnası: ResourceUnavilabilityException
 }
```



Şekil 3.29. "Rezervasyon İptal et" UCM, istisnalar içeren kullanım örneği

- Kullanım örneği: Ödünç öğe

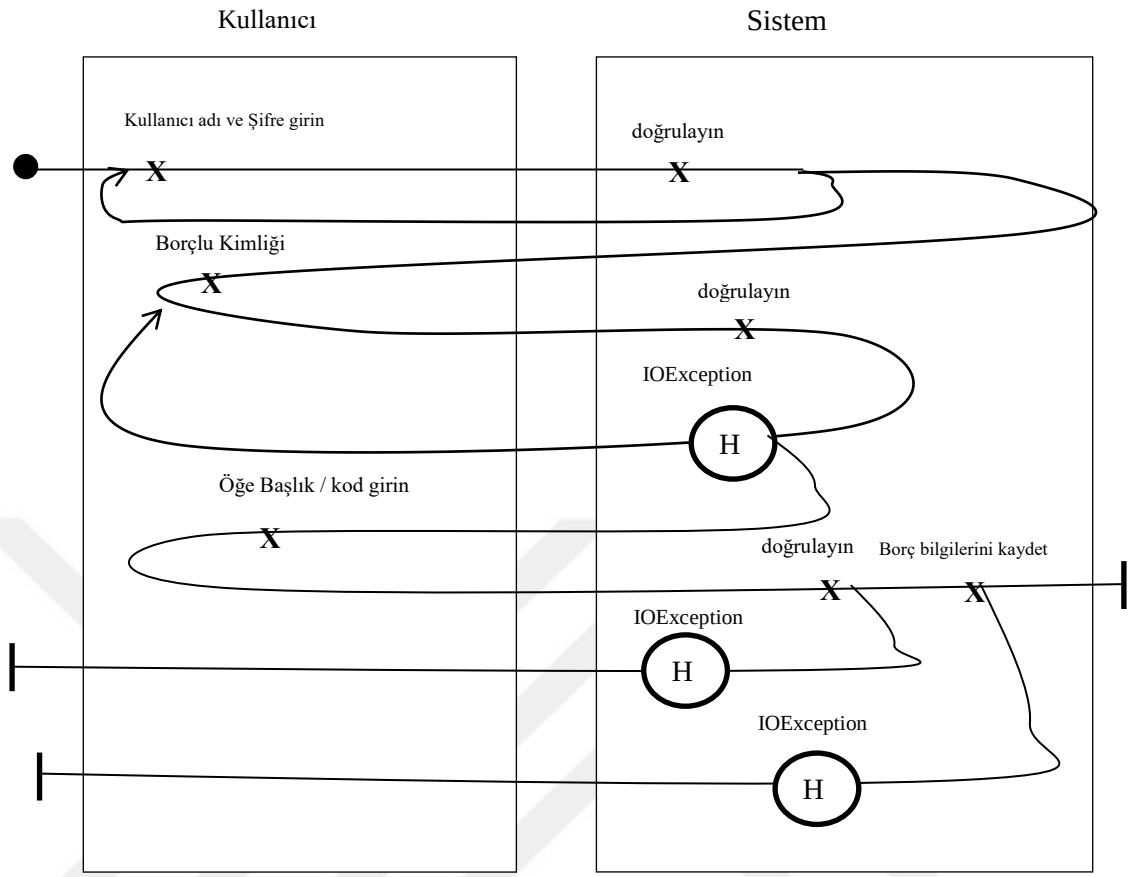
Tablo 3.25, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu ve Şekil 3.30 UCM'leri kullanarak oluşturulan görsel senaryoyu göstermektedir.

Tablo 3.25. *Kullanım örneği " Ödünç öğe "*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	Ödünç öğe
Ön koşul:	Ödünç isteği

```
                                Type Exceptions Report
                                ======================
Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Borrower ID
      S: Search Borrower
      U: Enter Item title/Code
      S: Search item
      S: Store lend information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      S: Search Borrower
      System Exception: ResourceUnavailbilityException

      U: Enter Item title/Code
      User Exception: IOException

      S: Search item
      System Exception: ResourceUnavailbilityException

      S: Store lend information
      System Exception: ResourceUnavailbilityException

  }
```

```
Deneyin
 {
    U: Kullanıcı adı ve Şifre girin
    S:  kullanıcı adı ve şifreyi doğrulayın
    U: Borçlu Kimliği Girin
    S: Borçlu Kimliğini araştırıb
   U: Öğe Başlık / kod girin
   S: Arama öğesi
   S: Borç bilgilerini kaydet
 }

Yakalamak (Istisna E)
 {
     U :Kullanıcı adı ve Şifre girin
     Kullanıcı İstisnası: IOException

     U: Borçlu Kimliği Girin
     Kullanıcı İstisnası: IOException

     S: Borçlu Kimliğini araştırın
     Kullanıcı İstisnası: ResourceUnvailabilityException

     U: Öğe Başlık / kod girin
     Kullanıcı İstisnası: IOException

     S: Arama öğesi
     Sistem İstisnası: ResourceUnavilabilityException

     U: İptal işlemini onayla
     Kullanıcı İstisnası: IptalException

    S: Borç bilgilerini kaydet
     Sistem İstisnası: ResourceUnavilabilityException
}
```



Şekil 3.30 "Ödünç öğe" UCM, istisnalar içeren kullanım örneği

Kullanım örneği: Ödünç alınan dökümanın iadesi

Tablo 3.26, istisnai durumları da dahil olmak üzere bu kullanım durumunun dokümantasyonunu ve Şekil 3.31 UCM'leri kullanarak elde edilen görsel senaryosunu göstermektedir.

Tablo 3.26. *Kullanım örneği "İade maddesi" (istisnalarla)*

Kısıtlı dil kullanan senaryo dökümantasyonu	
Kullanım durumunun adı	İade maddesi
Precondition:	Güncelleme iade talebi

```
                              Type Exceptions Report
                              ======================

Try
  {
      U: Enter user name and Password
      S: validate user name and password
      U: Enter Borrower ID
      S: Search Borrower
      U: Enter Item title/Code
      S: Check returns
      S: Store returns information
  }


Catch (Exception E)
  {
      U: Enter user name and Password
      User Exception: IOException

      U: Enter Borrower ID
      User Exception: IOException

      S: Search Borrower
      System Exception: ResourceUnavailbilityException

      U: Enter Item title/Code
      User Exception: IOException

      S: Check returns
      System Exception: IOException

      S: Store returns information
      System Exception: ResourceUnavailbilityException

  }
```

Deneyin
{
U: Kullanıcı adı ve Şifre girin
S: kullanıcı adı ve şifreyi doğrulayın
U: Borçlu Kimliği Girin
S: Borçlu Kimliğini araştırın
U: Öğe Başlık / kod girin
S: Iadeyi kontrol edin
S: Bilgi kaydet
}

Yakalamak (Istisna E)
{
U :Kullanıcı adı ve Şifre girin
Kullanıcı İstisnası: IOException

U: Borçlu Kimliği Girin
Kullanıcı İstisnası: IOException

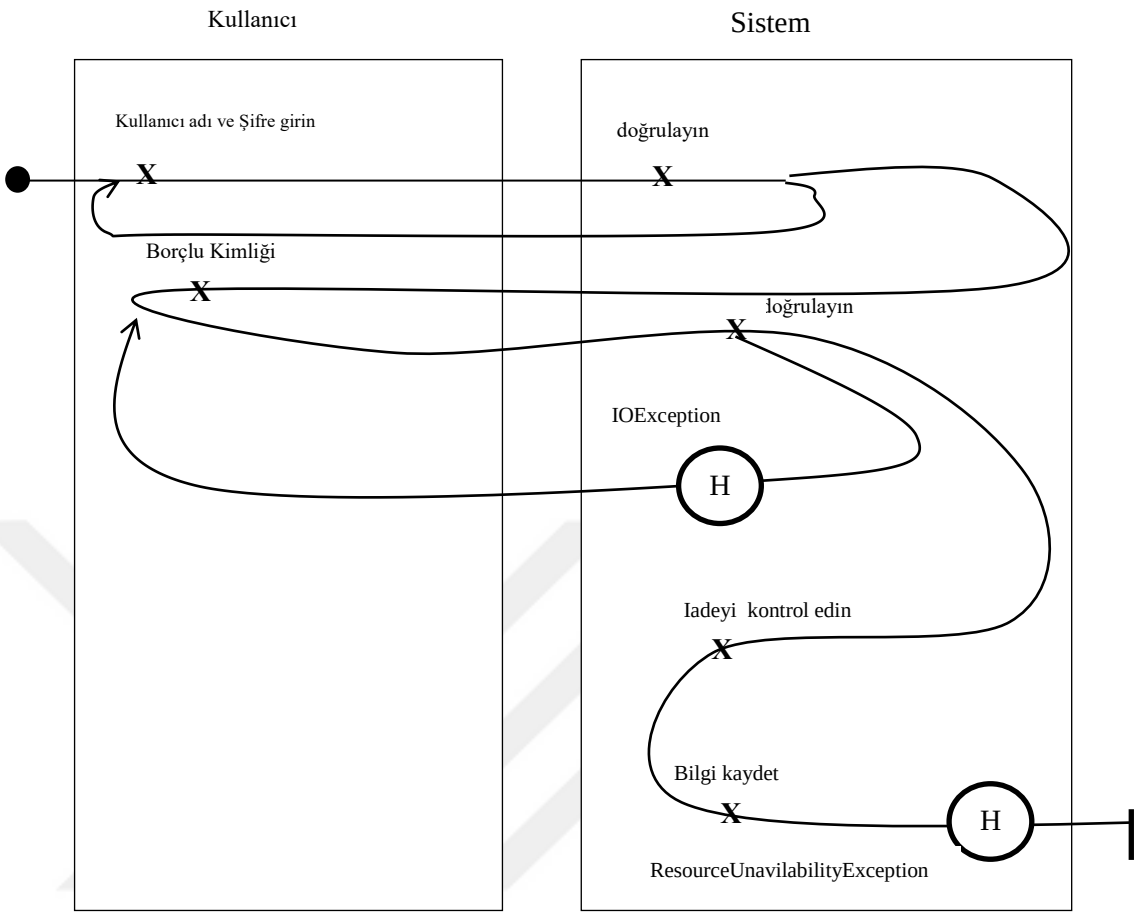
S: Borçlu Kimliğini araştırım
Sistem İstisnası: ResourceUnvailabilityException

U: Öğe Başlık / kod girin
Kullanıcı İstisnası: IOException

S: Arama öğesi
Sistem İstisnası: ResourceUnavilabilityException

S: Iadeyi kontrol edin
Kullanıcı İstisnası: IOException

S: Bilgi kaydet
Sistem İstisnası: ResourceUnavilabilityException
}



Şekil 3.31 "İade maddesi" UCM, istisnalar içeren kullanım örneği

4. BULGULAR VE TARTIŞMA

Bu bölümde, şartların ortaya çıkarılması ve istisnaların ortaya çıkarılması için yapılan araştırmaların kısa bir özeti verilmiştir. Yukarıda belirtildiği gibi, istisnaların ortaya çıkmasının ana nedenlerinden birisi, gereksinimlerin eksikliği ve belirsizliğin olmasıdır. Bu nedenle, bu bölümde tezde yapılan çalışmaların tartışılması üzerine odaklanılmıştır.

4.1. Gerekliliklerin ortaya çıkartılması

Kaliteli bir ürün üretmek için daha fazla yazılım geliştirme çalışması gerekmektedir. İhtiyaç arama, SDLC'nin en kritik faaliyetidir; çünkü bir sistemin beklenen gereksinimlerine odaklanır.

Birçok çalışma mevcut gereksinimlerin ekstraksiyon tekniklerinin, geliştirilecek sistemlerin gereksinimlerini karşılayan açık gereksinimleri karşılamadığını ve genellikle bu sistemlerin sapmalarına neden olan yazılım hatalarının % 40'ının kendilerine ait olduğunu ortaya koymaktadır hedefler bunlardır: yetersiz gereksinimlerin ortaya çıkması veya yorumlanmasında hata olması nedeniyle ortaya çıkmasıdır. Bu, sonuçta bekledikleri sistemlerin zayıflığı açısından paydaşlara tatmin edici bir son vermeyecektir, bu sistemin kabul seviyesi, sistemin gerçekleştirilmesi gereken ihtiyaçları karşılaması gerçeğine bağlıdır. (Achimugu, Selamat, İbrahim, & Mahrin, 2014; Doran; Mulla & Girase, 2012)

Yapılan literatür çalışmalaından anlaşıldığı gibi, mükemmel şartların elde edilmesi için, üretim gereksinimlerin üretilmesi ile anormal durumlar (istisnalar) arasındaki boşluk SDLC'nin erken evresinde dikkate alınması gerekir. Güçlü bilgisayar sistemlerini oluşturma konusundaki önemi ve etkisi hakkındaki bilgi temelinde iyi tanımlanmış gereksinimleri kolaylaştırmak için pek çok çaba gösterildi (de Lemos & Romanovsky, 2001; Rajagopal ve diğerleri, 2005; Recker, 2010).

Güçlü bilgisayar sistemlerinin gelişmesinden dolayı, iyi türetilmiş gereksinimleri kolaylaştırmak için birçok çalışma yapılmıştır.

Gereksinim zorluklarının incelenmesi birkaç yıldan fazla bir araştırma alanı olmasına rağmen, halen zorluklarla karşı karşıya olduğumuz bulguları, SDLC'nin erken evrelerinde istisnalarla başa çıkmak gibi küçük iyileştirmelere ihtiyaç vardır

Buna ek olarak, istisnaların erken tespit edilmesi ve ele alınmasıyla ilgili çalışmaların birçoğunun hala soyut biçimde olduğuna veya hiçbir zaman bunların asla ele alınmayacağına inanıyoruz. Yazılım ürün iyileştirme formlarının, gelişme faaliyetlerinin ilerleyen şeklini açık bir şekilde desteklemesi ve bu tür ürünlerin (sistemlerin) spesifikasyonunda ortaya çıkabilecek sorunlar ve zorluklarla mücadele için donanımlı olması gerekmektedir.

Yukarıdaki sonuçların bir sonucu olarak, açık gereksinimlerin oluşturulması arasındaki boşluğun, bir sistemin geliştiricileri ve kullanıcıları arasındaki zayıf iletişimden geldiği söylenebilir. Bu, eksik, belirsiz veya gereksiz üretilen gereksinimlerin olduğu anlamına gelir. Bu zorluklar kritik önem taşımakta ve sistem başarısı üzerinde etkili olmaktadır ve bu sorunlar, kullanıcı gereksinimlerinin yanlış anlaşılmasından veya bu gereksinimlerde sık yapılan değişikliklerden kaynaklanmaktadır. Yazılım geliştiricisinin, bir sistemden ihtiyaçlarının ne olduğunu bilmeyen belirsiz müşterilerle yüz yüze gelmesi bir örnek olarak verilebilir. Bu, tutarsızlık, kayıp, belirsiz veya eksik gereksinimlere neden olabilir; bu da hedefindeki sistemin değişiklik derecesini arttırır ve daha sonra arızalanmasına neden olur.

Bu tür “eksikliklerin üstesinden gelmek için, iki ana istisna türü aracılığıyla bu sorunları azaltmak ve ele almak için bir mekanizma sunulmuştur. (1) eksik gereksinim istisna türleri ve (2) Zayıf etkileşim istisna türleri (kullanıcı davranış ve sistem yanıtları istisna türleri olarak sınıflandırılabilir). Sistematik bir yaklaşım önerisi, İlgili paydaşlarından (kullanıcılardan) basit bir şekilde sistemin kesin kullanıcı gereksinimlerinin nasıl oluşturulacağına odaklanmasıdır. Bu tezde önerilen istisna tespit mekanizmasının temeli oluşturulmuştur.

Önerilen gereksinim şartları yaklaşımının temel amacı, yazılım geliştiricilerinin, eksik ve belirsizlikler içermeyen kullanıcı gereksinimlerini oluşturmak için paydaşlarla aktif ve işbirliği içinde çalışmalarına yardımcı olmaktır. Önerilen yaklaşım, yazılım geliştiricilerinin ve kullanıcıların kesin gereksinimler oluşturmaya daha fazla dikkat etmelerini ve teşvik etmelerini sağlayacak ve kritik taleplerin erken odaklanması gereken istisnalar üzerine yoğunlaşmayı göz önüne alan bir kılavuz adımları sunmaktadır.

4.2. İstisnalar

Bazı çalışmalar, çoğu geliştiricinin erken aşamada istisnaları tanımlamaya ve değerlendirmeye odaklanmadığını veya önemsemediğini ve istisna sürecinin, geliştirme ekibindeki rollere eklenen başka bir "istisna mühendisi" tarafından gerçekleştirildiği sonucuna varmıştır (Ebert ve ark., 2015; H. Shah ve diğerleri, 2008). Tekrar tekrar belirttiğimiz gibi, bu, daha sonraki bir tarihte geliştirme sürecinin başarısını olumsuz yönde etkileyebilir.

Öte yandan, birçok çalışma, test ve hata ayıklama aşamasındaki istisnaların ele alınmasına (Schroter ve diğerleri, 2010) odaklanırken, diğerleri bu kodları izlemek için özel test durumlarının tanımlanmasını önermişlerdir. (Mao ve Lu, 2005).

Bu çalışmalar hala soyut ve bazı vakalara odaklanma yetersiz olabilir ve bu prosedür test kapsamına girmeyen diğer olayları hesaba katmayabilir. Dolayısıyla temel safhanın (şartların çıkarılması aşamasının) kuvvetli olması olması, anormal durumların azalmasına neden olır ve bu tezde üzerinde durmaya çalıştığımız konu daba sonraki anormalliklerin azaltılmış olmasıdır., İstisnai durumların nedenlerinin

iki ana kategoride ortaya çıkabileceğini düşünüyoruz: kullanıcının ihtiyaçları hakkında zayıf bilgi ve bunlardan kaynaklanabilecek beklenmedik durumlar, ve sistem kullanıcılarından kaynaklanabilecek kötü etkileşimdir. Burada fikirlerimizi temel alarak, herhangi bir eylemin güçlü temelinin durduğu sağlam zemin olduğunu düşünüyoruz: Herhangi bir güçlü sistemin temeli, sistemin ne yaptığını ve en üst düzeyde ne olduğunu bilmeğe e dayanır. Bu nedenle, en üst derecede güvenilirliği elde etmeyi amaçlayan bir rehber oluşturmak, mesajın amaçlarından biridir.

Araştırmacılar, istisnalar konusunu hafifletmeye çalışmışlar. Yazılım geliştiricilerini destekleyen sınıflandırmalar ve yaklaşımlar önermişlerdirler. Örneğin, Russel, (Nick Russell ve diğerleri, 2006a) işyeri başarısızlığının bu tür sınıflandırmalardan biri olduğunu varsaymıştır. Buna katılıyoruz ve herhangi bir işlemin amacına ulaşamaması argümanını güçlendiriyor, istisna ortaya çıkıyor demektir.

Kullanıcı gereksinimlerini etkili bir şekilde oluşturmaya odaklandığımızda, bu sınıflamayı dolaylı olarak kabul edilmiştir ancak bu sınıflamalardan hiçbiri, yanlış gereksinimlerin, istisnaların nedenlerinden biri olduğu gerçeğine odaklanmamışkardır. Öte yandan, Bu istisnaların ortaya çıkmasını önlemek için kullanıcının sistemle zayif etkileşmesin'n ele alınması gereken önemli bir nokta olduğu anlaşılmıştır.

İstisnaların sayısını ve zorluklarını azalrmak için bazı modeller önerilmiştir: kullanıcı rolü modeli, kullanıcıların çalışmalarında yaptıkları işleri yakalamak için kullanılır ve görselleştirme, gereksinimlerin eksik veya yanlış anlaşılmasını engellemek için önemlidir.

Kullanıcı görevlerini simüle etmek için UML kullanım durum şeması önerilmiştir. O zaman sistematik yaklaşım, kullanıcı gereksinimlerinin yapımını ve yakalanmasını destekler ve bu aşamada istisnaların tanımını da içeren ve daha sonra olumsuz etkilerini azaltan diğer modellerin oluşmasını sağlar. Böylece istisnaları tespit etme ve işleme süreci bu senaryoların analizine dayanacaktır. Bu analiz, istisna tespiti için bir araç geliştirmemizi sağlamıştır. İstisnalar olduğu anlamına gelen NLP'nin

durdurma noktalarının anlaşılmasını arttırmak için analizlerimizin uygulanabileceği gösterilmiştir.

Kullanıcı gereksinimleri senaryolarının yazılmasında izlenecek bir dizi kural tanımlanmıştır, bu nedenle istisnaların tespiti ve ele alınması süreci bu senaryoların analizine dayanacaktır. Bu analiz, istisnalar algılama için bir araç geliştirmemizi sağladı. Analizlerimiz, NLP'nin durdurma noktaları anlayışını arttırmak için uygulanabileceğini ve bunun da istisnaların olduğu anlamına geldiğini göstermiştir.

Çalışma bize, SDLC'nin erken aşamalarında dikkate alınması gereken başka zorlukların olduğunu hatırlatmıştır. Bir kullanım durumu iyi analiz edildiğinde, hatasız yazılım geliştirmeye katkıda bulunacaktır. Önceki bölümlerde gösterildiği gibi, bu yaklaşım, gereksinim analizi seviyesini artırabilir.

Çok sayıda kullanım senaryoları üzerinde çalışılınca, geliştiricilerin yakalayamayacağı birçok istisna olduğunu keşfedilmiştir. SDLC'nin eğilimli aşaması olan gereksinim analizi aşamasını geliştirmek için bu tür bir analizin gerekli olduğuna inanıyoruz. Bu nedenle, maliyet ve efor tahmini derin bir analiz olduğunda geliştirilecek ve hangi yazılımın yapması gerektiğini iyi anlaşılacaktır. yazılımın başarısız olmasına yol açabilecek durumların çoğunu dahil ederek geliştirilecektir.

Çalışma, SDÖK'nin erken safhalarında düşünülmesi gereken başka güçlüklerin olduğunu bize hatırlattı. Bir kullanım durumu iyi analiz edildiğinde, bu, hatasız yazılım geliştirmek için katkıda bulunacaktır. Önceki bölümlerde gösterildiği gibi, bu yaklaşım gereksinim analizi düzeyini artırabilir

Bir çok kullanım durumu senaryosunu incelediğimizde, geliştiricilerin yakalayamayacakları çok sayıda istisna bulunduğunu keşfettik. SDLC'nin eğilimli aşaması olan ihtiyaç analizi aşamasını iyileştirmek için böyle bir analizin yapılması gerektiğine inanıyoruz. Bu nedenle, derin bir analiz yapıldığında ve hangi yazılımın yapması gerektiğini iyi anladığında, maliyet ve emek tahminleri geliştirilecektir. Bu, yazılımın başarısız olmasına neden olabilecek durumların çoğunu ekleyerek geliştirilecektir.

Rehberlik yaklaşımının olmaması nedeniyle, geliştiriciler genellikle istisnalarla başa çıkmayı ihmal ederler ve sadece normal durumlara odaklanırlar. İstisnaların göz ardı edilmesi yazılım kalitesini düşürecek ve başarısızlığa yol açacaktır Bu nedenle, SDLC'nin erken aşamasında istisnalarla uğraşmak gereklidir..

İstisnaları anlamak ve bunlarla başetmek için, aşağıdaki tavsiyeler dikkate alınmalıdır:

- Senaryodaki her ifade tarafından atılan istisnaları belgeleyin. Bu, istisnaların izlenebilirliğini erkenden kolaylaştıracak ve doğru işlem mekanizması daha kolay hale gelecektir.

- Özel durumun ayrıntılı bir açıklaması ve istisna neden olan eylem gereklidir. Bu, önceki noktada açıklanan izlenebilirlik ve istisna işleme işlemini geliştirecekti.

5. SONUÇLAR

Çalışmamızda, birçok geliştiricinin SDLC'nin erken aşamasında istisnalarla uğraşmayı ihmal etmesinin nedenleri araştırılmıştır. Bu durum, bu sorunların ele alınmasında zaman kaybına yol açabilir veya özellikle sistem gereksinimlerinin eksik veya belirsiz olması durumunda, bu tür programların zamanında teslim edilmesinde başarısızlığa veya gecikmeye neden olabilir.

Bu tez, ihtiyaç aşamasını etkileyebilecek istisnaların sınıflandırılmasını sunmaktadır ve erken tespit edilip ele alınmadığı takdirde daha sonraki aşamaları etkileyecektir. Bundan dolayı, çalışmamızın temel amaçlarından biri, geliştiricilerin bu konuya odaklanmaları ve önemli bir rol olarak düşünerek, SDLC'nin ilk aşamasına dâhil edilmelidir. Önerilen yaklaşım üç temel aşamayı içermektedir; istisnaların sınıflandırılması, gereksinimlerin ortaya çıkması, istisnaların tespiti ve modellenmesi. Özellikle istisnaların sınıflandırılmasında, (1) Eksik gereksinimler, (2) yazılım kullanıcıları ve sistem yanıtı arasındaki zayıf etkileşim, sınıflandırılmasından kaynaklanabilecek istisnai durumlar daha önceden belirlenmiş ve açıklanmıştır. İlk sınıflandırmayı ele almak için, kullanıcı gereksinimlerinin ortaya çıkartılması sürecini kolaylaştırmak ve daha sonra bu gereksinimlerin eksikliği veya belirsizliği nedeniyle ortaya çıkabilecek istisnai durumlarda şansını azaltmak için sistematik bir yaklaşım önerilmiştir. Bu yaklaşımın ilk aşaması gereksinimlerin üretilmesi ile ilgilidir. Bu aşama, görüşlerin yakınlaşmasına ve geliştiriciler ve paydaşlar arasında ortak bir anlayış dilinin oluşturulmasına katkıda bulunan sistematik bir yaklaşımı benimsemiştir. İkinci aşamada, tanımlanmış sınırlı kurallara göre gereksinim senaryolarının belgelendirilmesi süreci ele alınırken, daha sonra bu senaryolarda istisnaların yakalanmasına yardımcı olunmuştur.

İstisnaların algılama aşamasında, yazılımın kullanıcıları ile yazılımın arasındaki etkileşimi izlemek ve bu etkileşimler sırasında ortaya çıkabilecek istisnaları yakalamaya odaklanmak üzerinde duruldu. Bu aşamada, Python dili kullanılarak bir yazılım aracı geliştirildi. Algılama süreci, önceki senaryolarda bulunabilecek istisnaların keşfedilmesini kolaylaştırmak için bir dizi doğal dil kavramına dayanmaktadır. Bu istisnalar, gereksinim senaryolarını daha iyi belgelemek için

yazılım geliştiricileri ve sistem paydaşları arasında müzakere noktaları olarak işlev gören kesme noktalarını temsil edecektir.

İstisnaların tespit mekanizması, ilk olarak, kullanıcıların gerçekleştirebileceği bazı olaylarda temsil ettiği ve bu nedenle mevcut senaryonun eksik olmasına ve dolayısıyla tamamlanamamasına neden olan zayıf etkileşim yoluyla iki temel etkileşime dayanmaktadır. İkinci olarak, genellikle gözlemlenmesi ve önceden beklenmesi gereken teknik olay olan istisnalar, örneğin saklama operasyonları, sıfır bölme gibi bazı matematiksel işlemler gibi istisnalardır. İhtiyaç testi süreci, hedeflerine ulaşılamamasına veya muhtemelen başarısızlığa neden olabilecek kusursuz bir yazılım sisteminin oluşturulmasını sağlamak için önemli bir unsur ve omuriliktir. Yazılım sistemlerinin senaryolarını modellemek için bir grafik yazılımı (UCM) benimsedik ve istisnaların ortaya çıkışını temsil eden kesme noktalarını açıkladık ve böylece onların işleme sürecini kolaylaştıracağız. Grafik yazılımı, senaryoların gereksinimleri üretme ve belgeleme aşamasında yanlışlıkla ihmal edilmiş olabilecek diğer koşulların keşfedilmesine veya türetilmesine katkıda bulunabilecek daha kapsamlı bir şekilde gösterilmesine yardımcı olmuştur.

Son olarak, önerilen yaklaşımın etkinliğini test etmek için, önerilen aşamaların uygulandığı bir vaka çalışması sunduk ve yaklaşım, kusur ve eksikliklerin olmadığı gereksinimleri ortaya çıkarma sürecinde bir iyileştirme mekanizması gösterdi ve böylece bu süreç cesaretlendirici oldu. Yazılım geliştiricilerinin istisnaları erken odaklaması ve ele alması, sistemin paydaşlarını tatmin ederek daha sonra önerilen yaklaşımın etkililiğini artırması için bir fırsat sunmaktadır.

6. ÖNERİLER

Bu bölümde, yazılım geliştirme sürecinin iyileştirilmesinde, özellikle de istisna sorunlarının ele alınmasını teşvik etmek ve onları ihmal etmemek için gelecekteki çalışmalar ele alınmıştır. Bunlar arasında, diğer araştırma alanlarını açan, özellikle istisnalar üzerinde durulması gereken önemli bir konu olarak odaklanan diğer noktalar bulunmaktadır.

- Etkili yazılımlara olan talebin artması nedeniyle, geliştiricilerin istisna durumlarını ele almaları ve uygun düzeltici önlemleri almaları gerekmektedir, çünkü bu istisna durumlarının üstesinden gelememesi, yazılım kalitesinin düşük kalmasına neden olabilir. Böylece, buradaki soru, yazılım yaşam döngüsünün iyileştirilmesine nasıl odaklanılacağıdır.

- Bu tezde, paydaşlardan gelen taleplerin çıkarılması sürecinde ihmal edilmiş olabilen istisnaların çoğunu yakalamak için kullanıcı gereksinimlerini izleme konusunda doğal işleme dilinin birçok kavramı tanıtılmıştır. Ancak, diğer kavramlar ve teknikler örneğin bu süreçte birçok yapay zeka fikrinin uygulanması gibi prosesleri geliştirmek için, daha sonra araştırma noktaları olarak düşünülebilir.

- Regnell ve diğerleri (2009), tarafından ifade edildiği gibi, özellikle kalite gereksinimleri (QR) talebi nedeniyle yazılım gereksinimleri daha karmaşık hale gelecektir. Bu nedenle, mevcut yaklaşımda QR'nin (Fonksiyonel olmayan gereksinimler) entegrasyonu, daha sonra ele alınacak eğilimlerden biri olabilir.

- Bilindiği gibi, gelişim sürecinde kaydedilen ilerlemeyle, sistem artefaktlarının daha fazla anlaşılması gittikçe daha fazla gerekmektedir. İstisnalarla uğraşmanın temeli; sadece test aşamasında ele almak yerine yazılım geliştirmenin ilk aşamasının temeli olduğunu tahmin ettiğimiz için sorun, diğer yazılım sorunlarını nasıl tahmin edebileceğimizdir. Diğer yazılım

sorunlarını çözmek için daha fazla araştırmanın gerekli olduğunu gösterir. İleride bu konu ile ilgili çalışmalara yağunlaşılmalıdır.

- Önerilen yaklaşımı bir dizi karmaşık örnek olay üzerinde pratik olarak uygulayarak daha fazla iyileştirme yapılabilir ve geri bildirimin alınması bu yaklaşımı geliştirecektir.

- Shah H. ve diğerleri (2008), istisnaların işlenmesinin zor bir görev olduğunu bu nedenle "İstisna Mühendis" adlı yeni bir rolün eklenmesi gerektiğini belirtmiştir. Sorun şu ki, bu rol gerçekten gerekli ve gereksinim mühendisinin rolü ile bu rol arasındaki bir çelişki olup olmadığıdır. Ek olarak, gelecekte başka sorunlar tespit edilirse ne olacağıdır.

- Yazılım geliştiricileri, geliştirme sürecindeki ayrı bir süreç olarak resmi matematiksel veya mantıksal terimlerde istisna davranış davranışını tanımlamalı / tanımlamalıdır. Bu, program kodunun durumları etkili bir şekilde otomatik olarak analiz etmesine ve tespit etmesine yardımcı olur.

- Gelişimde ayrı bir süreç olarak yazılım geliştiricileri, resmi matematiksel veya mantıksal olarak istisnaların davranışını tanımlayan bir açıklama / şartname yazmalıdır. Durumları otomatik olarak analiz etmek ve tespit etmede. Bu programı kodlamada yardımcı olacaktır.

.

KAYNAKLAR

Achimugu, P., Selamat, A., Ibrahim, R., Mahrin, M. N. (2014). A systematic literature review of software requirements prioritization research. *Information and Software Technology, 5*6(6), 568-585.

Alrajeh, D., Kramer, J., Van Lamsweerde, A., Russo, A., & Uchitel, S. (2012). Generating obstacle conditions for requirements completeness. Paper presented at the Software Engineering (ICSE), 2012 *34th International Conference on.*

Amyot, D., Charfi, L., Gorse, N., Gray, T., Logrippo, L., Sincennes, J., Ware, T. (2000). Feature Description and Feature Interaction Analysis with Use Case Maps and LOTOS. Paper presented at the FIW.

Amyot, D., Mussbacher, G. (2000). On the extension of UML with use case maps concepts. Paper presented at the International Conference on the Unified Modeling Language.

Baker, P. (2006). Glossary of corpus linguistics: *Edinburgh University Press.*

Balaji, S., Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1), 26-30.

Bano, M., Zowghi, D. (2015). A systematic review on the relationship between user involvement and system success. *Information and Software Technology*, 58, 148-169.

Barbosa, E. A., Garcia, A.,Mezini, M. (2012). Heuristic strategies for recommendation of exception handling code. Paper presented at the Software Engineering (SBES), 2012 26th Brazilian Symposium on.

Bendraou, R., Osterweil, L., Kannengiesser, U., Christov, S., Lerner, B., Wise, A. (2010). Exception Handling Patterns for Process Modeling.

Berenbach, B. (2006). Impact of organizational structure on distributed requirements engineering processes: lessons learned. Paper presented at the Proceedings of the 2006 international workshop on Global software development for the practitioner.

Bruntink, M., Van Deursen, A., Tourwé, T. (2006). Discovering faults in idiom-based exception handling. Paper presented at the Proceedings of the 28th international conference on Software engineering.

Buhr, P. A., Mok, W. R. (2000). Advanced exception handling mechanisms. I*EEE Transactions on Software Engineering*, 26(9), 820-836.

Buhr, R. J., Casselman, R. S. (1995). Use case maps for object-oriented systems: Prentice-Hall, Inc.

Cabral, B., Marques, P. (2008). A case for automatic exception handling. Paper presented at the Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering.

Cacho, N., Barbosa, E. A., Araujo, J., Pranto, F., Garcia, A., Cesar, T., .Garcia, I. (2014). How does exception handling behavior evolve? an exploratory study in java and c# applications. Paper presented at the Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.

Cailliau, A., van Lamsweerde, A. (2013). Assessing requirements-related risks through probabilistic goals and obstacles. Requirements Engineering, 18(2), 129-146.

Cailliau, A., & van Lamsweerde, A. (2014a). Integrating exception handling in goal models. Paper presented at the Requirements Engineering Conference (RE), 2014 IEEE 22nd International.

Cailliau, A., van Lamsweerde, A. (2014b). Integrating exception handling in goal models. Paper presented at the 2014 IEEE 22nd International Requirements Engineering Conference (RE).

Cambria, E.,White, B. (2014). Jumping NLP curves: A review of natural language processing research. IEEE Computational intelligence magazine, 9(2), 48-57.

Carrizo, D., Dieste, O., Juristo, N. (2014). Systematizing requirements elicitation technique selection. *Information and Software Technology*, 56(6), 644-669.

Casati, F., Fugini, M., & Mirbel, I. (1999). An environment for designing exceptions in workflows. Information systems, 24(3), 255-273.

Castor Filho, F., Guerra, P. A. d. C., Pagano, V. A., & Rubira, C. M. F. (2005). A systematic approach for structuring exception handling in robust component-based software. *Journal of the Brazilian Computer Societ*y, 10(3), 5-19.

Chen, J., Xu, X., Osterweil, L. J., Zhu, L., Brun, Y., Bass, L., . . . Wang, Q. (2015). Using simulation to evaluate error detection strategies: A case study of cloud-based deployment processes. Journal of Systems and Software, 110, 205-221.

Cornu, B., Seinturier, L., Monperrus, M. (2015). Exception handling analysis and transformation using fault injection: Study of resilience against unanticipated exceptions. I*nformation and Software Technolog*y, 57, 66-76.

Coughlan, J., Lycett, M.,Macredie, R. D. (2003). Communication issues in requirements elicitation: a content analysis of stakeholder experiences. *Information and Software Technolog*y, 45(8), 525-537.

Damian, D., Chisan, J. (2006). An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Transactions on Software Engineering, 32(7), 433-453.

Damian, D., Chisan, J., Vaidyanathasamy, L.,Pal, Y. (2005). Requirements engineering and downstream software development: Findings from a case study. *Empirical Software Engineerin*g, 10(3), 255-283.

Davey, B., & Parker, K. R. (2015). Requirements elicitation problems: a literature analysis. Issues in Informing Science and Information Technology, 12, 71-82.

Davis, C. J., Fuller, R. M., Tremblay, M. C., Berndt, D. J. (2006). Communication challenges in requirements elicitation and the use of the repertory grid technique. Journal of Computer Information Systems, 46(5), 78-86.

De Lemos, R., Romanovsky, A.(2001). Exception handling in the software lifecycle. International Journal of Computer Systems Science and Engineering, 16(2), 167-181.

Dijkman, R., van IJzendoorn, G., Türetken, O., de Vries, M. (2015). The effect of exceptions in business processes.

Doran, M. A comparison of Problem Frames, a problem-based method, and KAOS, a goal-based method, for Requirements Analysis within a financial environment.

Ebert, F., & Castor, F. (2013). A study on developers' perceptions about exception handling bugs. Paper presented at the Software Maintenance (ICSM), 2013 29th IEEE International Conference on.

Ebert, F., Castor, F., & Serebrenik, A. (2015). An exploratory study on exception handling bugs in Java programs. J*ournal of Systems and Softwar*e, 106, 82-101.

Ferreira, J. E., Takai, O. K., Malkowski, S., & Pu, C. (2010). Reducing exception handling complexity in business process modeling and implementation: The wed-flow approach. Paper presented at the OTM Confederated International Conferences" On the Move to Meaningful Internet Systems".

Fricker, S. A., Grau, R., Zwingli, A. (2015). Requirements engineering: best practice Requirements Engineering for Digital Health (pp. 25-46): Springer.

Fu, J., Bastani, F. B., Yen, I.-L. (2007). Model-driven prototyping based requirements elicitation. Paper presented at the Monterey Workshop.

Gillam, L., Tariq, M., Ahmad, K. (2005). Terminology and the construction of ontology. Terminology. International Journal of Theoretical and Applied Issues in Specialized Communication, 11(1), 55-81.

Glinz, M., Fricker, S. A. (2015). On shared understanding in software engineering: an essay. Computer Science-Research and Development, 30(3-4), 363-376.

Guimarães, L. R., Souza Vilela, P. R. (2005). Comparing software development models using CDM. Paper presented at the Proceedings of the 6th conference on Information technology education.

Hadar, I.,Zamansky, A. (2015). Cognitive factors in inconsistency management. Paper presented at the Requirements Engineering Conference (RE), 2015 IEEE 23rd International.

Hagal, M., Kandemirli, F, Amsaad. (2017), "Developers' Views Regarding Exception Handling in Software Development", presented at International Conference On Recent Advances in Computer Science and Information Technology(ICRACSIT), pp 62-65, Istanbul, Turkey.

Harrold, M. (2010). Understanding Exception Handling: Viewpoints of Novices and Experts.

Hastie, S., Wojewoda, S. (2015). Standish Group 2015 Chaos Report-Q&A with Jennifer Lynch. Retrieved, 1(15), 2016.

Hecht, H. (2008). Requirements for software exception handling. Paper presented at the Aerospace Conference, 2008 IEEE.

Henneman, E. A., Avrunin, G. S., Clarke, L. A., Osterweil, L. J., Andrzejewski, C., Merrigan, K., Henneman, P. L. (2007). Increasing patient safety and efficiency in transfusion therapy using formal process definitions. Transfusion Medicine Reviews, 21(1), 49-57.

Hijazi, H., Khdour, T., Alarabeyyat, A. (2012). A review of risk management in different software development methodologies. International Journal of Computer Applications, 45(7), 8-12.

Huzooree, G.,Ramdoo, V. D. (2015). A Systematic Study on Requirement Engineering Processes and Practices in Mauritius. *International Journal, of advanced research in computer science and software enginnering* 5(2).

Hwang, S.-Y.,Tang, J. (2004). Consulting past exceptions to facilitate workflow exception handling. Decision support systems, 37(1), 49-69.

Iqbal, T., Suaib, M. (2014). Requirement Elicitation Technique:-A Review Paper. Int. J. Comput. Math. Sci, 3(9).

Joseph, M. Collaborative Requirements Elicitation Using Elicitation Tool for Small Projects.

Kelly, D. F. (2007).A software chasm: Software engineering and scientific computing. IEEE software, 24(6).

Kienzle, J. (2008). On exceptions and the software development life cycle. Paper presented at the Proceedings of the 4th international workshop on Exception handling.

Konaté, J.,Sahraoui, A. E. K., Kolfschoten, G. L. (2014). Collaborative requirements elicitation: A process-centred approach. Group Decision and Negotiation, 23(4), 847-877.

Kösters, G., Six, H.-W., Winter, M. (2001). Validation and verification of use cases and class models. Paper presented at the 7th International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'2001, Proc.).

Kukkanen, J.,Väkeväinen, K., Kauppinen, M.,Uusitalo, E. (2009). Applying a systematic approach to link requirements and testing: A case study. Paper presented at the Software Engineering Conference, 2009. APSEC'09. Asia-Pacific.

Leopold, H.,Niepert,M., Weidlich, M., Mendling, J.,Dijkman, R., & Stuckenschmidt, H. (2012). Probabilistic optimization of semantic process model matching. Paper presented at the International Conference on Business Process Management.

Letsholo, K., Chioasca, E.-V., Zhao, L. (2012). An integration framework for multi-perspective business process modeling. Paper presented at the Services Computing (SCC), 2012 IEEE Ninth International Conference on.

Liu, A., Li, Q., Xiao, M. (2007). A declarative approach to enhancing the reliability of bpel processes. Paper presented at the Web Services, 2007. ICWS 2007. IEEE International Conference on.

Lloyd, W. J., Rosson, M. B., Arthur, J. D. (2002). Effectiveness of elicitation techniques in distributed requirements engineering. Paper presented at the Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on.

Mao, C.-Y.,Lu, Y.-S. (2005). Improving the robustness and reliability of object-oriented programs through exception analysis and testing. Paper presented at the Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on.

Mourão, H., Antunes, P.(2005). A collaborative framework for unexpected exception handling. Paper presented at the International Conference on Collaboration and Technology.

Mulla, N., Girase, S. (2012). A new approach to requirement elicitation based on stakeholder recommendation and collaborative filtering. I*nternational Journal of Software Engineering and Applications,* 3(3), 51.

Munassar, N. M. A., Govardhan, A. (2010). A comparison between five models of software engineering. IJCSI, 5, 95-101.

Mussbacher, G., Amyot, D., & Weiss, M. (2007). Visualizing early aspects with use case maps Transactions on aspect-oriented software development III 105-143, Springer.

Natella, R., Cotroneo, D., Duraes, J. A., & Madeira, H. S. (2013). On fault representativeness of software fault injection. I*EEE Transactions on Software Engineering*, 39(1), 80-96.

Pa, N. C., Zin, A. M. (2011). Requirement elicitation: identifying the communication challenges between developer and customer. *International Journal of New Computer Architectures and their Applications (IJNCA*), 1(2), 371-383.

Pacheco, C., Garcia, I. (2008). Stakeholder identification methods in software requirements: empirical findings derived from a systematic review. Paper presented at the Software Engineering Advances, 2008. ICSEA'08. The third International Conference on.

Pandey, D.,Suman, U., Ramani, A. (2010). An effective requirement engineering process model for software development and requirements management. Paper presented at the Advances in Recent Technologies in Communication and Computing (Artcom), 2010 International Conference on.

Pohl, K. (2010).Requirements engineering: fundamentals, principles, and techniques: Springer Publishing Company.

Pressman, R. S. (2005). Software engineering: a practitioner's approach: Palgrave Macmillan.

Rajagopal, P., Lee, R., Ahlswede, T., Chiang, C.-C., & Karolak, D. (2005). A new approach for software requirements elicitation. Paper presented at the Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on.

Ramesh, B., Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1), 58-93.

Recker, J. (2010). Opportunities and constraints: the current struggle with BPMN. Business Process Management Journal, 16(1), 181-201.

Robertson, S., Robertson, J. (2012). Mastering the requirements process: Getting requirements right: Addison-wesley.

Robillard, M. P., Murphy, G. C. (1999). Analyzing exception flow in Java programs (Vol. 24): Springer-Verlag.

Romanovsky, A. (2007). On exceptions, exception handling, requirements and software lifecycle.

Russell, N., van der Aalst, W. (2006). Arthur,"Exception Handling Patterns in Process-Aware Information Systems," BPMcenter: org, Tech. Rep.

Russell, N., van der Aalst, W., & ter Hofstede, A. (2006a). Workflow exception patterns. Paper presented at the International Conference on Advanced Information Systems Engineering.

Russell, N., van der Aalst, W. M., Ter Hofstede, A. H. (2006b). Exception handling patterns in process-aware information systems. BPM Center Report BPM-06-04, BPMcenter. org, 208.

Sampaio, P. (2012). Can Business Process Modeling Bridge The Gap Between Business and Information Systems?

Schroter, A., Schröter, A., Bettenburg, N., Premraj, R. (2010). Do stack traces help developers fix bugs? Paper presented at the Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on.

Shah, H., Görg, C., & Harrold, M. J. (2008). Why do developers neglect exception handling? Paper presented at the Proceedings of the 4th international workshop on Exception handling.

Shah, U. S., Jinwala, D. C. (2015). Resolving ambiguities in natural language software requirements: a comprehensive survey. ACM SIGSOFT Software Engineering Notes, 40(5), 1-7.

Shahrokni, A., Feldt, R. (2013). A systematic review of software robustness. I*nformation and Software Technology*, 55(1), 1-17.

Sharma, S., Pandey, S. (2013). Revisiting requirements elicitation techniques. I*nternational Journal of Computer Applications*, 75(12).

Shui, A., Mustafiz, S., Kienzle, J., & Dony, C. (2005). Exceptional use cases. Paper presented at the International Conference on Model Driven Engineering Languages and Systems.

Sinha, S., Orso, A., & Harrold, M. J. (2004). Automated support for development, maintenance, and testing in the presence of implicit control flow. Paper presented at the Proceedings of the 26th International Conference on Software Engineering.

Sommerville, I. (2010). Software engineering: Pearson.

Sousa, K., Schilling, A., & Furtado, E. (2007). Integrating usability, semiotic, and software engineering into a method for evaluating user interfaces Verification, Validation and Testing in Software Engineering (pp. 55-81): IGI Global.

Stone, A., Sawyer, P. (2006). Identifying tacit knowledge-based requirements. IEE Proceedings-Software, 153(6), 211-218.

Sutcliffe, A., Sawyer, P. (2013). Requirements elicitation: Towards the unknown unknowns. Paper presented at the Requirements Engineering Conference (RE), 2013 21st IEEE International.

Van Dooren, M., Steegmans, E. (2005). Combining the robustness of checked exceptions with the flexibility of unchecked exceptions using anchored exception declarations. Paper presented at the ACM SIGPLAN Notices.

Van Lamsweerde, A., Letier, E. (2000). Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineerin*g, 26(10), 978-1005.

Vijayan, J., Raju, G. (2011). A new approach to requirements elicitation using paper prototype. I*nternational Journal of Advanced Science and Technology*, 28, 9-16.

Weber, B., Reijers, H. A., Zugal, S., Wild, W. (2009). The declarative approach to business process execution: An empirical test. Paper presented at the International Conference on Advanced Information Systems Engineering.

Xie, T., Thummalapenta, S. (2012). Making exceptions on exception handling. Paper presented at the Exception Handling (WEH), 2012 5th International Workshop on.

Yang, H.-L., Tang, J.-H. (2003). A three-stage model of requirements elicitation for Web-based information systems. *Industrial Management & Data System*s, 103(6), 398-409.

Yue, T., Briand, L. C., Labiche, Y. (2009). A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. Paper presented at the International conference on model driven engineering languages and systems.

Zowghi, D., Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools Engineering and managing software requirements 19-46, Springer.

EKLER

Ek1. İSTISNA ALGILAMA PROGRAM KODU

```
import sqlite3
import nltk
from nltk.corpus import wordnet
from tkinter import *
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.tokenize import TweetTokenizer
from nltk.tokenize import MWETokenizer
tknzr = TweetTokenizer()
import os
ps=PorterStemmer()
conn = sqlite3.connect("exceptiondb.db")
cursor=conn.execute("SELECT * FROM userexceptionslist")
cursor2=conn.execute("SELECT * FROM systemexceptionslist")
T1=[]
T2=[]
for row in cursor:
    T1.append(ps.stem(row[0]))
    T2.append(row[1])
T3=[]
T4=[]
for row in cursor2:
    T3.append(ps.stem(row[0]))
    T4.append(row[1])
stop_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',
'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',
'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',
'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']
root = Tk()
root.title("Use case senario. Put u: for user statement, and s: for system responses, result in
        ExceptionResult.txt")
text = Text(root)
text.pack()
print('\n' *30)
text.focus_set()
def callback():
    tfile = open("ExceptionResult.txt","w")
    tfile.write("                    Type Exceptions Report \n")
```

```
tfile.write("                          ====================== \n")
print('\n' *30)
text.focus_set()
array = []
v = text.get("1.0",END)
z = len(v)
newlinedenoter="\n"
newlinelength=len(newlinedenoter)
newstring="
tfile.write("Try"+"\n")
tfile.write("  {"+"\n")
tfile.write("      ")
for i in range(z):
    if (v[i:i+newlinelength]==newlinedenoter):
        array +=[newstring]
        newstring="
        tfile.write("\n")
        if i+1 in range(z):
         tfile.write("      ")
    else:
        newstring +=v[i]
        tfile.write(v[i])
#tfile.write("\n")
tfile.write("  }"+"\n")
tfile.write("\n")
tfile.write("\n")
tfile.write("Catch (Exception E)"+"\n")
tfile.write("  {"+"\n")
tokenizer = MWETokenizer([('a', 'little'), ('a', 'little', 'bit'), ('a', 'lot')])
#get the exceptions list
sentence = nltk.word_tokenize("the customer inputs the card into the ATM")
tp2="
if len(array)>0:
 for p in range(len(array)):
    sentence=[]
    sentence=tokenizer.tokenize(array[p].split())
    sent=nltk.pos_tag(sentence)
    #print("--------------------------- VERBS ARE ----------------")
    tokenizer = MWETokenizer([('a', 'little'), ('a', 'little', 'bit'), ('a', 'lot')])
    synonymstable=[]
    stablex=[]
    stemtable=[]
    # POS: VB- Verb base form,  VBZ- verb presnt tense third personnel , VBD- verb past
      tense, JJ- adjective or numeral, NNP is a proper noun and sometimes will be used as
      verb such as enter
    for i in range(60):
        print
    checkvar=0
    vr1=0
    for word, pos in nltk.pos_tag(sentence):
       if pos == 'NNp' or 'VB' or pos=='VBZ' or pos=='VBD' or pos=='JJ' or pos=='VBG'
      or pos=='VBN' or pos=='VBP' or pos=='NNS' or pos=='NN':
            ####### Search will be on this word if it exists in the repository,
            ####### if not found we we will search its sysnonyms in the dictionary
```

```
        if (sentence[0] !=" "):
            word=word.lower()
            if (sentence[0]=="U:" or sentence[0]=="u:"):
                    x="
                    iii=0
                    for p in range(len(T1)):
                     checkvar=0
                     if (T1[p].lower()==ps.stem(word)):
                        for st in sentence:
                            x=x+st+' '
                        tfile.write("      "+x+"\n")
                        tfile.write("      "+"User Exception: "+T2[p]+"\n")
                        tfile.write("              "+"\n")
                        tp2=T2[p]
                        checkvar=1
                        break
                        ### An exception occured
            if (sentence[0]=="S:" or sentence[0]=="s:"):
                iii=1
                x="
                i=0
                for p in range(len(T3)):
                  if (T3[p]==ps.stem(word)):
                        x="
                        for st in sentence:
                            x=x+st+' '
                        if (T4[p] !=tp2):
                         tfile.write("      "+x+"\n")
                         tfile.write("      "+"System Exception: "+T4[p]+"\n")
                         tfile.write("              "+"\n")
                        checkvar=1
                        ### This means it is an exception

                        break
    if pos == 'VB' or pos=='VBZ' or pos=='VBD' or pos=='JJ' or pos=='VBG' or
      pos=='VBN' or pos=='VBP' or pos=='NNS' or pos=='NN':
     if checkvar==0:
        ### There is no exception equaivalance to the core key word,
        ### so we will search on its synonyms
        #("------------------- Search in synonyms------------------")
        stablex=[]
        stemtable=[]
        synonymstable=[]
        synonyms = wordnet.synsets(word)
        for word in synonyms:
          synonymstable.append(word.lemma_names())
        for word in synonymstable:
           for t in word:
              if ps.stem(t) not in stablex:
                 stablex.append(ps.stem(t))
        ### search on its synonyms
        if (sentence[0]=="U:" or sentence[0]=="u:"):
                x="
                tp2="
```

```
                    iii=0
                    for q in range(len(stablex)):
                      for p in range(len(T1)):
                       if (T1[p]==ps.stem(stablex[q])):
                          for st in sentence:
                             x=x+st+' '
                          tfile.write("      "+x+"\n")
                          tfile.write("      "+"User Exception: "+T2[p]+"\n")
                          tp2=T2[p]
                          tfile.write("               "+"\n")
                          checkvar=1
                          break
                          ### This means it is an exception
                      if checkvar==1:
                             break
            if (sentence[0]=="S:" or sentence[0]=="s:"):
                  x="
                  iii=1
                  for q in range(len(stablex)):
                     for p in range(len(T3)):
                       if (T3[p]==ps.stem(stablex[q])):
                          for st in sentence:
                             x=x+st+' '
                          if T4[p] != tp2:
                           tfile.write("      "+x+"\n")
                           tfile.write("      "+"System Exception: "+T4[p]+"\n")
                           tfile.write("               "+"\n")
                          checkvar=1
                          break
                    if checkvar==1:
                          break
            #("--------------------End of Synonyms------------------")
    tfile.write("  }"+"\n")
    tfile.close()
b = Button(root, text="Check exceptions", width=30, command=callback)
b.pack()
mainloop()
conn.close()
```

ÖZGEÇMİŞ

Adı Soyadı : Mohamed Ali S. HAGAL
Doğum Yeri ve Yılı : 14-10-1965, Benghazi-Libya
Medeni Hali : Evli + 4 çocuk
Yabancı Dili : İngilizce
E-posta : mohdhg@yahoo.com

Eğitim Durumu

Lise : Shouhada yanier, Benghazi-Libya

Lisans : Lisans, Bilgisayar Bilimi, Fen Fakültesi, Benghazi Üniversitesi, Libya.

Yüksek Lisans : Bilgisayar bilimi yüksek lisansi, Bilgi Teknolojileri Fakültesi Benghazi Üniversitesi, Libya.

Mesleki Deneyim

İş doğası	İş Yeri	İtibaren	İçin
Doktora öğrencisi	Kastamonu universitesi , Yazılım Mühendisliği	2014	2018
Okutman	Bilgi Teknolojileri Fakültesi, Benghazi Üniversitesi-Libya	2012	2014
Kalite koordinatörü	Bilgi Teknolojileri Fakültesi, Benghazi Üniversitesi-Libya	2010	2014
Öğretim görevlisi asistanı	Bilgi Teknolojileri Fakültesi, Benghazi Üniversitesi-Libya	2008	2012
ICDL" sertifikası " eğiticisi	Araştırma ve danışmanlık merkezi, Benghazi Üniversitesi-Libya	2008	2008
Öğretim asistanı	Fen Fakültesi, Benghazi Üniversitesi-Libya	1992	2007
Geliştirici	Bazı şirketlerde geliştirici, Libya	1992	2005
Yazılım Mühendisi	Genel elektrik şirketi, Benghazi-Libya	1989	1992

Yayınlar

Hagal, M. A. & et-al (2012), Algorithms and data structures (Academic Book-Arabic edition), ISBN:978-1-0940-8, Benghazi, Libya.

Hagal, M. A., & Sallabi, O. M. (2011). A Structured Approach for Extracting Functional Requirements from Unclear Customers. In Arab Conference on Information Technology (ACIT International), International conference on (pp. 67-73). Riyadh, Saudi.

El-Shaari, M. A., Hagal, M. A., & El-Badry, Z. S. (2011). A General Framework to Bridge the Gap between Conceptual System and Abstract System in Software Development. In Arab Conference on Information Technology (ACIT International), International conference on (pp. 61-65). Riyadh, Saudi.

Hagal, M. A., & Fazzani, F. H. (2012, December). A use case map as a visual approach to reduce the degree of inconsistency. In Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on (pp. 1-4). IEEE.

Hagal, M. A., & Alshareef, S. F. (2013, December). A systematic approach to generate and clarify consistent requirements. Presented at IT Convergence and Security conference (ICITCS), (pp. 1-4). Macau, China, IEEE.

Abdelaziz, T. M., Zada, Y. N., & Hagal, M. A. (2014). AStructural APPROACH TO IMPROVE SOFTWARE DESIGN REUSABILITY., presented at international conference on Software Engineering and Applications (JSES2014), Zurich, Switzerland.

Hagal, M. & Kandemili F. (2017), "Reducing missing requirements issues: Complete, Unambiguous and Necessary Requirements Elicitation", International journal of Advanced research in computer science and software engineering 7(1), pp 10-14.

Sallabi, O., Benhaloum, A. & Hagal, M. (2017), "A cyclomatic complexity Visual tool for simple source code coverage", International journal of Advanced research in computer science and software engineering, 7(5), pp 136-141.

Hagal, M. A. (2017), " Do Exception handling a hard task and should be delayed to later stages?", International Conference on Multidisciplinary, Engineering, Science, Education and Technology (IMESET'17 Baku), (pp. 47), Azerbaijan Technical University, Baku, Azerbaijan.

Hagal, M., Kandemirli, F. & Amsaad F. (2017), "Developers' Views Regarding Exception Handling in Software Development", presented at International Conference On Recent Advances in Computer Science and Information Technology(ICRACSIT), pp 62-65, Istanbul, Turkey.