YAZILIM KURUMLARINDA ÇEVİK DÖNÜŞÜMÜN ETKİLERİNİN ÖLÇÜLMESİ VE DEĞERLENDİRİLMESİ: DENEYSEL BİR ÇALIŞMA

MEASURING AND EVALUATING THE EFFECTS OF AGILE TRANSFORMATION IN SOFTWARE ORGANIZATIONS: AN EMPIRICAL STUDY

FEYZA NUR KILIÇASLAN

YRD. DOÇ. DR. AYÇA TARHAN
Tez Danışmanı

DR. HALUK ALTUNEL
Eş Danışman

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

Ocak 2018

FEYZA NUR KILIÇASLAN'ın hazırladığı **"Yazılım Kurumlarında Çevik Dönüşümün Etkilerinin Ölçülmesi ve Değerlendirilmesi: Deneysel Bir Çalışma"** adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**' nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Ali YAZICI
Başkan .....

Yrd. Doç. Dr. Ayça TARHAN
Danışman .....

Dr. Haluk ALTUNEL
Eş Danışman .....

Doç. Dr. Aysu Betin CAN
Üye .....

Doç. Dr. Altan KOÇYİĞİT
Üye .....

Yrd. Doç. Dr. Burcu CAN BUĞLALILAR
Üye .....

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

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Fen Bilimleri Enstitüsü Müdürü

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

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

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

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

☒ **Tezimin/Raporumun** 05/12/2018 **tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç Kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
(Bu sürenin sonunda uzatma için başvuruda bulunmadığım taktirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı ve ya tamamının fotokopisi alınabilir)

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

☐ **Serbest Seçenek/Yazarın Seçimi**

09/01/2018

(İmza)

Öğrencinin Adı Soyadı

Feyza Nur KILIÇASLAN

ETİK

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

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

beyan ederim.

03/01/2018

FEYZA NUR KILIÇASLAN

ÖZET

YAZILIM KURUMLARINDA ÇEVİK DÖNÜŞÜMÜN ETKİLERİNİN ÖLÇÜLMESİ VE DEĞERLENDİRİLMESİ: DENEYSEL BİR ÇALIŞMA

Feyza Nur KILIÇASLAN
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Danışmanı: Yrd. Doç. Dr. Ayça TARHAN
Eş Danışman: Dr. Haluk ALTUNEL
Ocak 2018, 88 sayfa

Birçok yazılım kuruluşu çevik dönüşümün avantajlarından yararlanabilmek için çalışma süreçlerini plan güdümlüden çevik yöntemlere doğru değiştirmeye başlamıştır. Bu sebeple çevik yöntemlerin benimsenmesinin, yazılım kuruluşlarını nasıl etkilediği sıklıkla tartışılmaktadır. Çevik dönüşümün etkilerini ölçmek, çevik yöntemlerin yazılım kuruluşlarına ne ölçüde katkı sağladığını değerlendirmek ve anlamak açısından önemlidir. Bu çalışmada, çevik dönüşümü gerçekleştiren orta ölçekli bir yazılım kurumunda, dönüşümün etkilerini ölçmek amacıyla bilgi ihtiyaçları ve metrikler tanımlanmış ve kurumda uygulanarak ölçüm sonuçları paylaşılmıştır. Ölçmeye temel olacak tanımları belirlerken öncelikle, literatürde mevcut çalışmalar incelenmiş ve bu çalışmalardan çıkarılan metrikler ölçtükleri iş, süreç, ürün ve kaynak varlıklarına göre gruplanmıştır. Ardından, yazılım kurumunda bu varlıklar açısından çevik dönüşümün etkilerini ölçmeyi sağlayacak bilgi ihtiyaçları, ISO 15939 Yazılım Ölçme Standardı rehber alınarak belirlenmiş ve literatürden derlenen metriklerle ilişkilendirilmiştir. Bu bağlamda iş, süreç, ürün ve kaynak açılarından çevik dönüşümün etkilerini ölçmeyi sağlayacak bilgi göstergeleri ve ilişkili ölçme

yapıları (türetilmiş ve temel metrikler, ölçme fonksiyonları vb.) tanımlanmıştır. Son olarak bu bilgi ihtiyaçları üzerinden, yazılım kurumundaki çevik dönüşümün etkileri ölçülmüş ve ölçüm sonuçları değerlendirilmiştir. Değerlendirme sonuçları, çevik dönüşümün yazılım kurumunu genel olarak olumlu yönde etkilediğini göstermektedir.

Anahtar Kelimeler: Çevik dönüşüm, Yazılım metrikleri, Ölçüm tasarımı, ISO 15939

ABSTRACT

MEASURING AND EVALUATING THE EFFECTS OF AGILE TRANSFORMATION IN SOFTWARE ORGANIZATIONS: AN EMPIRICAL STUDY

Feyza Nur KILIÇASLAN
Master of Science, Computer Engineering Department
Supervisor: Asst. Prof. Dr. Ayça TARHAN
Co-Supervisor: Dr. Haluk ALTUNEL
January 2018, 88 pages

Many organizations have started to change their working process from plan-driven to agile in order to leverage the benefits of agile transformation, so it is frequently discussed how adopting agile methodologies affect software organizations. Measuring effects of agile transformation is important in terms of evaluating and understanding to what extent agile methods contribute to software organizations. In this study a set of information needs and metrics, which are designed to measure the effects of agile transformation in a medium-sized software organization that has been going through agile transformation, are described and applied in the organization and the measurement results are shared. While defining the base of measurement, firstly related studies in literature are examined, and metrics derived from these studies are grouped by the measured entities of business, process, product, and resource. The information needs for measuring effects of agile transformation are determined by the guidance of ISO 15939 standard for Software Measurement Process, and then are aligned with the metrics compiled from the

literature. Information indicators and associated measurement constructs (derived and base metrics, measurement functions etc.) that enable measuring impacts of agile transformation are described from business, process, product and resource perspectives. As a result, the effects of agile transformation in the software organization using these information needs are measured and the results of measurement are evaluated. The evaluation results show that the agile transformation affected the software organization positively in general.

Keywords: Agile transformation, Software metrics, Measurement design, ISO 15939

TEŞEKKÜR

Öncelikle tez çalışmamın başından sonuna kadar beni her zaman destekleyen, bilgi ve tecrübeleriyle yol gösteren ve karşılaştığım her sorunda etkili çözümler bulmamı sağlayan tez danışmanım Yrd. Doç. Dr. Ayça TARHAN'a en içten teşekkürlerimi sunarım.

Yoğun iş temposuna rağmen bilgi birikimi ve pratik saha tecrübesiyle özellikle tez çalışmamın analizinde bana destek olan eş danışmanım Dr. Haluk ALTUNEL'e çok teşekkür ederim. Ayrıca değerli tez jüri üyeleri Prof. Dr. Ali YAZICI, Doç. Dr. Altan KOÇYİĞİT, Doç. Dr. Aysu Betin CAN ve Yrd. Doç. Dr. Burcu CAN BUĞLALILAR'a tezimi değerlendirerek yaptıkları katkılardan dolayı teşekkürü bir borç bilirim.

Tez çalışmam boyunca yardım ve desteklerinden dolayı oda arkadaşlarıma teşekkür ederim.

Hiçbir zaman desteklerini esirgemeyen değerli aileme ve çalışmalarım boyunca beni cesaretlendiren ve motive eden eşim Mehmet'e çok teşekkür ederim.

İÇİNDEKİLER

ÇİZELGELER

ŞEKİLLER

KISALTMALAR

DSD Dynamic Systems Development

FDD Feature Driven Development

GQM Goal-Question-Metric Framework

GUI Graphical User Interface

PMBOK Project Management Body of Knowledge

PMI Project Management Institute

RCI Rules Compliance Index

SLR Systematic Literature Review

TDD Test Driven Development

UAT User Acceptance Testing

XP Extreme Programming

1. GİRİŞ

1.1. Genel Bakış

Çevik dönüşüm, bir kuruluşun biçimini veya niteliğini, kademeli olarak esnek, işbirlikçi, kendi kendini organize eden ve hızlı değişen ortamı benimseyen ve geliştiren bir niteliğe dönüştürme eylemidir [1]. Kuruluşların gerçek, sağlıklı bir çevikliğe kavuşmanın faydalarından yararlanabilmesi için, çevik dönüşümün ne anlama geldiğini ve bu dönüşümün kattığı değeri iyi anlamaları önemlidir.

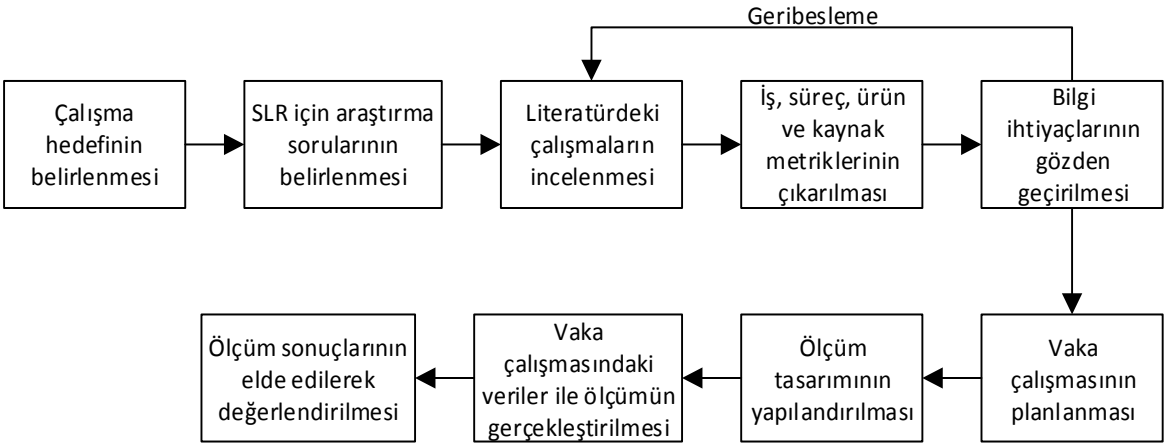
Çevik yazılım geliştirme yöntemleri, ürün teslimatını hızlandırması, değişen öncelikleri yönetme becerisini geliştirmesi, üretkenliği arttırması ve yazılım kalitesini iyileştirmesi [2] gibi sebeplerle farklı büyüklükteki firmalar arasında oldukça yaygınlaşmakta ve her yıl sektörde artan oranda benimsenmektedir. Bununla birlikte çevik dönüşümün yazılım firmalarını nasıl etkilediği birçok kez tartışma konusu olmuştur. Bu bağlamda çevik dönüşümün etkilerinin ölçülmesi, çevik yöntemlerin yazılım firmalarına sağladığı katkıların daha iyi anlaşılabilmesi ve değerlendirilebilmesi bakımından önemli bir yere sahiptir.

Bir deneyim raporunda [3], Yahoo! Music'deki çevik dönüşümün üretkenliği ve takım moralini önemli ölçüde artırdığı öne sürülmüştür. Başka bir şirket olan Primavera Systems tarafından, çevik yöntemlerin bir türü olan Scrum'ı uyguladıktan sonra müşteri tarafından bildirilen kusur sayısıyla ölçülen kalitede %30 artış olduğu bildirilmiştir [4]. Prochazka ve ark.'nın yaptığı bir çalışmada [5] ise "yeni bir çalışma yolu" olarak belirtilen çevik dönüşümün, müşteri ve takım memnuniyetini arttırmaya olanak sağladığı yönünde kanıtlar ortaya koyulmuştur.

Çevik yazılım geliştirmenin birçok açıdan iyileştirmeler sağladığı düşünülse de [6, 7] bazı çalışmalar çevik dönüşüm çabalarından yeterince kazanım sağlanamadığını ya da çabaların başarısızlıkla sonuçlandığını bildirmiştir. Zamana yayılan bir vaka çalışmasında [8], çevik yazılım geliştirme yöntemi benimsendikten sonra takip edilen projedeki hata yoğunluğunda önemli bir azalma ya da hata profillerinde değişiklik gözlemlenmediği raporlanmıştır. Ericsson'da bir pilot çevik dönüşüm projesi kapsamında yapılan başka bir çalışmada [9] ise çevik çalışma yöntemlerinin projelerin başarı oranını arttırmadığı bildirilmiştir.

Yukarıda bahsedilen hususlar çerçevesinde, birçok yazılım kurumunun çevik yöntemleri uygulamaya başlamasıyla çevik dönüşümün etkilerini ölçmek, mutlak bir ihtiyaç haline gelmiştir. Çevik dönüşümün firmaya sağladığı katkıların objektif bir şekilde ölçülebilmesi ise bu amaca hizmet edecek bilgi ihtiyaçlarının ve ilişkili metriklerin, firmada mevcut verilerin gözetilerek tanımlanmasıyla sağlanabilir.

Bahsedilen ihtiyaçtan hareketle bu tez çalışmasında, yazılım kurumlarında çevik dönüşümün etkilerinin ölçülmesi ve değerlendirilmesi amacıyla, literatüre dayalı bir ölçüm tasarımı oluşturulmuş ve orta-ölçekli bir yazılım firmasında uygulanarak sonuçları raporlanmıştır. Bu bağlamda (1) çevik dönüşümü ölçen çalışmalar üzerine bir sistematik literatür taraması (İng. Systematic Literature Review-SLR) yürütülmüş ve dönüşümü ölçmekte kullanılan metrikler çıkarılmış, (2) bu metrikler, orta-ölçekli bir yazılım firmasının bilgi ihtiyaçları ve topladığı veriler gözetilerek uyarlanmış ve firmanın ihtiyacına göre yeniden tasarlanmış, (3) oluşturulan ölçüm tasarımı, firmada çevik dönüşümün etkilerini ölçmek için bir vaka çalışması kapsamında uygulanarak sonuçları değerlendirilmiştir. Bu tez kapsamındaki çalışmaların gerçekleştirilmesi için izlenen adımlar Şekil 1.1'de sunulmuştur.



Şekil 1.1: Tez çalışmasının gerçekleştirilmesinde izlenen adımlar

1.2. Tez Yapısı

Bu tez raporunun izleyen bölümleri şu şekilde yapılandırılmıştır: İkinci bölümde sıklıkla kullanılan yazılım geliştirme yöntemleri, yazılım ölçüm modelleri ve yazılım metriklerine ilişkin esaslar açıklanmıştır. Üçüncü bölümde çevik dönüşümü ölçen çalışmalar üzerine bir sistematik literatür taraması sunulmuştur. Dördüncü bölümde, çevik dönüşüm gerçekleştiren bir firmada yapılan vaka çalışmasında kullanılmak

üzere, literatürdeki metriklerden uyarlanarak oluşturulan ölçme tasarımı anlatılmıştır. Beşinci bölümde vaka çalışmasından elde edilen sonuçlara, tartışmaya ve vaka çalışmasına yönelik geçerlilik tehditlerine yer verilmiştir. Son olarak altıncı bölümde ise bu tez çalışmasının sonuçları ve gelecek çalışmalardan bahsedilmiştir.

2. ÖN BİLGİ

Yazılım geliştirme, bir yazılım ürününün ortaya çıkmasıyla sonuçlanan faaliyetler grubundan oluşur. Bu faaliyetler yeni geliştirme, düzeltme/değişim, yeniden kullanım, yeniden mühendislik (İng. Re-engineering), bakım ya da yazılım ürünüyle sonuçlanan diğer faaliyetleri içerebilir [10]. Yazılım geliştirme yöntemleri geliştirme ortamına, kişilere ve geliştirilecek ürüne bağlı olarak değişir. Yazılım ürünü geliştiren kuruluşlar süreç, ürün, kaynak ya da işin değerlendirilmesi, öngörülmesi veya geliştirilmesi gibi birçok farklı amaçlarla ölçüm ve metriklere ihtiyaç duymakta ve bunları kullanmaktadırlar. Aşağıdaki alt bölümlerde, yaygın olarak kullanılan yazılım geliştirme yöntemleri, yazılım ölçümü ve metriklerine ilişkin temeller hakkında açıklamalara yer verilmiştir.

2.1. Geleneksel Yazılım Geliştirme Yöntemleri

Geleneksel yazılım geliştirme yöntemleri (İng. Traditional Software Development Methodologies) yaygın olarak “Plan güdümlü” (İng. Plan-driven) yöntemler olarak bilinir. Geleneksel yazılım geliştirmedeki yaklaşım, projeyi yazılım geliştirme yaşam döngüsünün her adımında titizlikle ve ayrıntılı olarak planlamaktır.

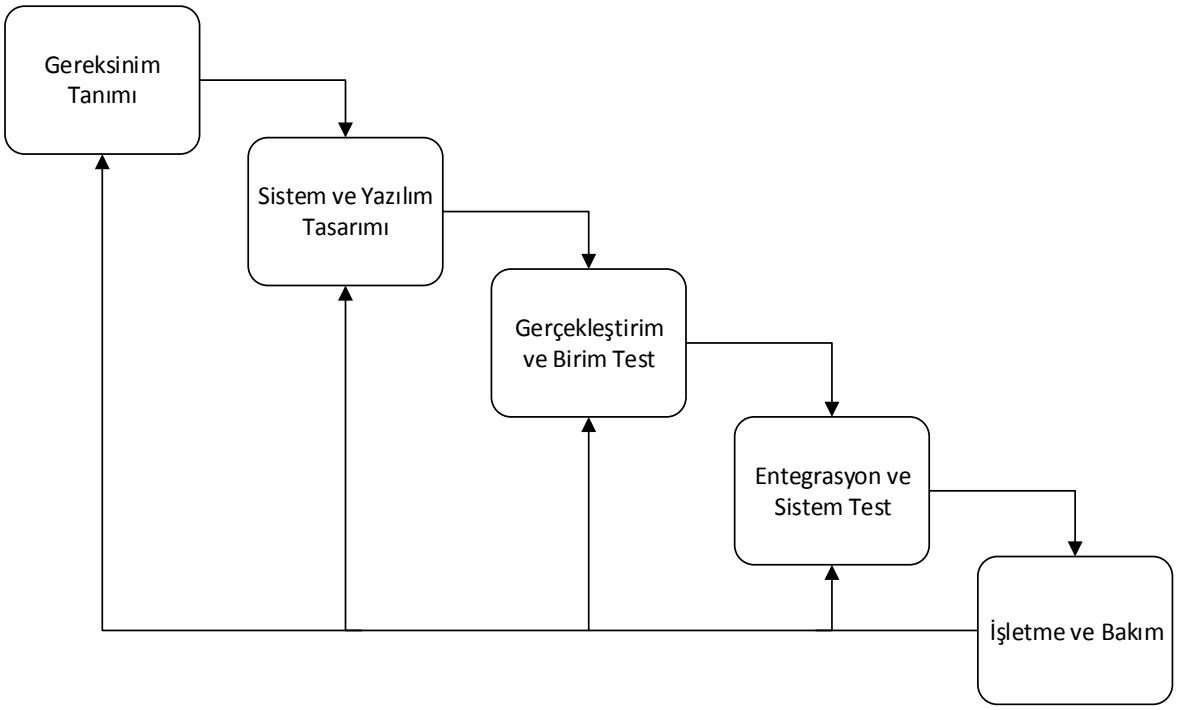
2.1.1. Çağlayan Modeli

Çağlayan ya da diğer bir deyişle şelale modeli (İng. Waterfall Model), önemli ve en çok kullanılan geleneksel yazılım geliştirme modellerinden biridir. Sommerville’e [11] göre çağlayan modeli, şartname (İng. Specification), geliştirme (İng. Development), doğrulama (İng. Verification) ve evrimin (İng. Evolution) temel süreç aktivitelerini ele alır ve bunları gereksinim şartnamesi, yazılım tasarımı, uygulama, test ve benzeri gibi ayrı işlem aşamaları olarak temsil eder. Çağlayan modelindeki aşamalar Şekil 2.1’de gösterilmiş ve aşağıda maddeler halinde açıklanmıştır.

- Gereksinim Tanımı (İng. Requirements Definition): İşlevsel (fonksiyonel), işlevsel olmayan, geliştirme ve bakım gereksinimleri [12], sistemin kısıtlamaları ve amaçları göz önüne alınarak tanımlanır.

- Sistem ve Yazılım Tasarımı (İng. System and Software Design): Tanımlanan gereksinimlere göre, yazılım ürününün nasıl geliştirilmesi gerektiği ve sistem mimarisi belgelendirilir.
- Gerçekleştirim ve Birim Test (İng. Implementation and Unit Testing): Önceki aşamalardaki tasarım ve gereksinimler, bir programa dönüştürülür. Programın her birimi, birimin şartnamesine uygun olup olmadığını doğrulamak için test edilmelidir.
- Entegrasyon ve Sistem Test (İng. Integration and System Testing): Programların birimleri, geliştirilmiş yazılım sisteminin gerektiği gibi çalışıp çalışmadığını teyit etmek için bütün bir sistem olarak bir araya getirilir ve test edilir.
- İşletme ve Bakım (İng. Operation and Maintenance): Son aşamada, sistem müşteri ortamında çalıştırılır. Önceki aşamalarda ortaya çıkmayan hataları düzeltmek ve sistemin daha iyi sürümler halinde piyasaya sürülmesini (İng. Release) sağlamak, bakımın bir parçasıdır.

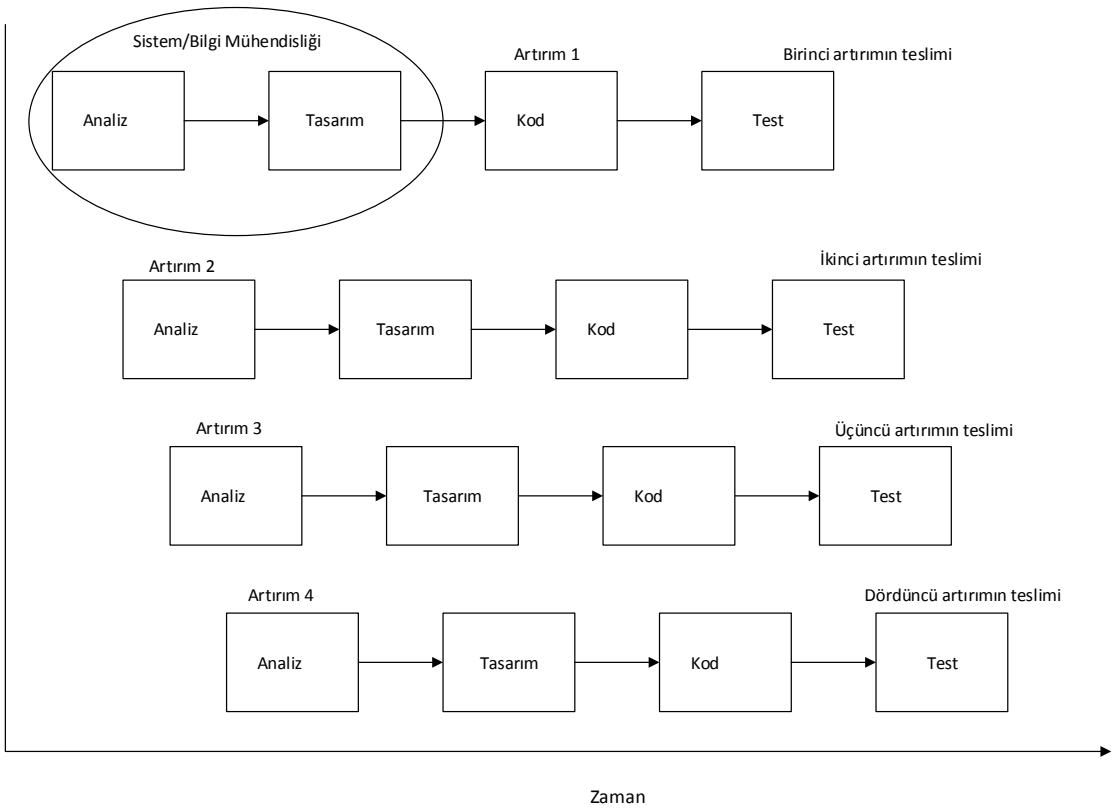
Bu modelde, bir sonraki aşamaya geçmeden önce her fazın tamamlanması gerekmektedir. Doğrusal yapı karakteristiği (İng. Linear Structure Characteristic) olmasına rağmen, pratikte fazlar arasında geri bildirim olabilir. Çağlayan modeli, genel olarak ürün tanımının istikrarlı olduğu ve gereksinimlerin iyi belgelendiği projeler için uygundur.



Şekil 2.1: Çağlayan Modeli [11]

2.1.2. Artırımsal Model

Artırımsal model (İng. Incremental Model), diğer bir geleneksel yazılım geliştirme yöntemidir. Artırımsal geliştirme modelinde yaşam döngüsü etkinlikleri, her bir artırımın sonunda bir teslimat üretmek için dönüşümlü olarak çalıştırılır. Bu süreç, Şekil 2.2'de gösterildiği gibi ürün son sürümüne ulaşıncaya kadar her artırım için tekrarlanmaktadır. Artırımsal geliştirmenin kullanılması, müşterinin ürünü geliştirme sürecinin ilk safhalarında değerlendirmesini sağlar ve geliştiricilerin harcadıkları çabalar sonucunda elde ettikleri başarılı sonuçlar sayesinde motivasyonlarını artırır [13]. Bu model, genellikle web uygulaması ve ürün bazlı projeler için tercih edilir.

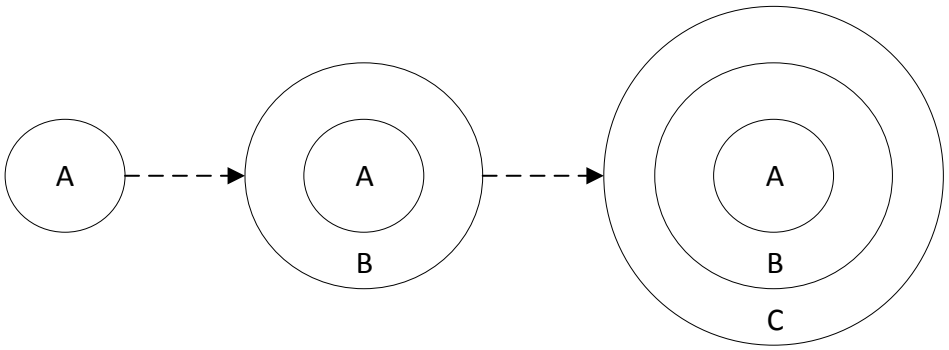


Şekil 2.2: Artırımsal Model [13]

2.1.3. Evrimsel Model

Evrimsel modelde (İng. Evolutionary Model) yazılım geliştirme takımı, işe gereksinimleri küçük modüllere bölerek başlar ve gereksinimlerle ilgili müşteri görüşmelerinden ortaya çıkan yazılım sistemini aşamalı olarak geliştirir. Şekil 2.3'de evrimsel modelin geliştirme süreci gösterilmektedir.

Evrimsel modeller tekrarlamalıdır. Bu modelleri kullanarak geliştirme takımları, gittikçe ilerleme kaydeder ve proje zamanlamasında daha eksiksiz olan ürün sürümlerini ortaya koyarlar. Yaygın olarak kullanılan evrimsel modellerden ikisi, prototipleme (İng. Prototyping) ve spiral modeller olup aşağıdaki paragraflarda açıklanmıştır.

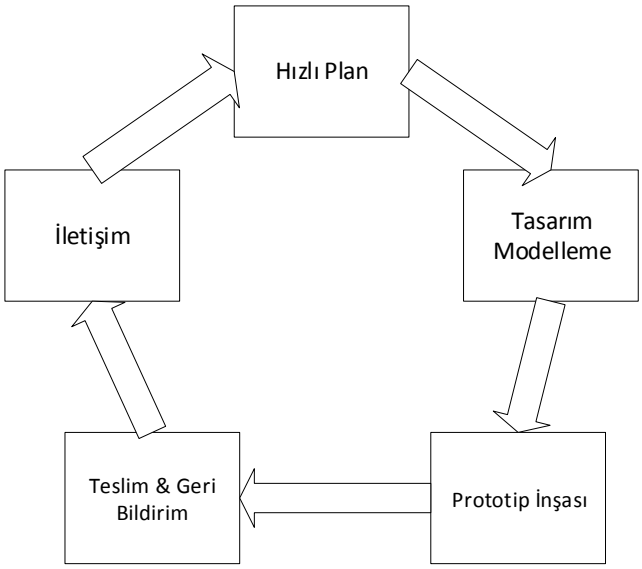


A, B, C aşamalı olarak geliştirilen ve teslim edilen bir yazılım ürününün modülleridir.

Şekil 2.3: Evrimsel Model [14]

Prototipleme modeli, Şekil 2.4'de gösterildiği gibi, müşteriler/kullanıcılar ve geliştiriciler arasındaki iletişimle başlar. Talep edilen ürünle ilgili genel hedefler belirlenir ve ardından tasarım ortaya çıkar. Geliştiriciler, müşterilerin kabaca özetlediği bir dizi genel hedefleri ve temel işlevleri gösteren yazılım ürününün bölümlerini oluşturarak bir prototip oluşturmaya başlar.

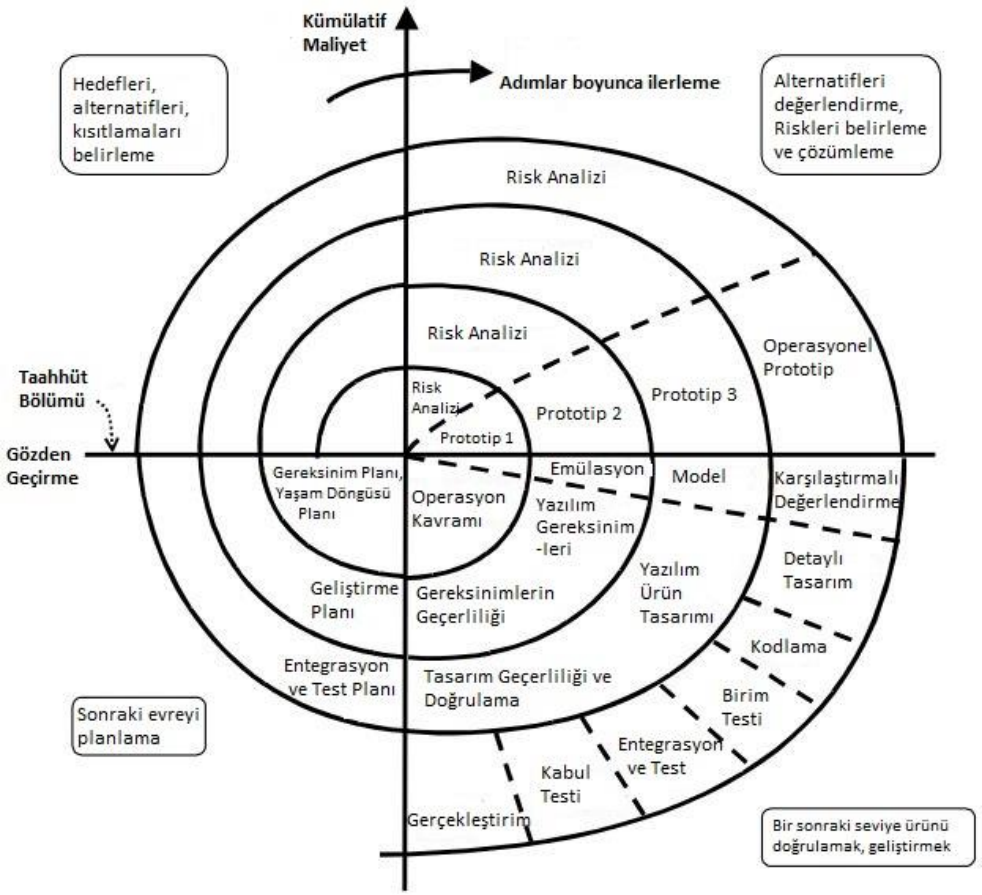
Prototipler müşteri için sunulan ürün ve talep edilen ürün arasındaki uygunluğun değerlendirilmesine olanak sağlar. Böylece müşteriler, prototipin güçlü ve zayıf yönlerini görebilir ve gereksinimler üzerinde rötuşlar yapabilir ya da yeni gereksinimler önerebilir. Prototipler, müşterilerin tüm gereksinimlerini karşılayan uygun prototipe ulaşana kadar, onların verdiği geri bildirimler temel alınarak değiştirilir. Prototipleme modeli, Grafik Kullanıcı Arayüzü (İng. Graphical User Interface-GUI) alt sistemlerinin geliştirilmesi için daha uygundur.



Şekil 2.4: Prototipleme Modeli [13]

Boehm [15] tarafından önerilen spiral model; artırımsal model, çağlayan model ve prototipleme modellerinin bir sentezidir. Genellikle küçük projeler için kullanım maliyeti nedeniyle tercih edilmez ve hantal bulunur. Şekil 2.5'de gösterildiği gibi spiral model, aşağıda açıklanan dört temel evreden oluşur.

- Planlama evresi: Yazılım takımı projeye, iş gereksinimlerini toplamakla başlar ve daha sonra sistem gereksinimleri, alt sistem gereksinimleri ve birim gereksinimleri ile devam eder. Bu evrenin sonunda gereksinim listesinin kesinleşmiş hali ortaya çıkar.
- Risk analizi evresi: Takım, olası tüm riskleri belirlemek için, önceki aşamada toplanan gereksinimler üzerine kuramsal olarak düşünür. Son olarak, alternatif çözümler bu aşamada risk azaltma stratejisi olarak üretilir.
- Mühendislik evresi: Yazılım ürünü geliştirmesi boyunca ileri testler bu evrede uygulanır. Test durumları (İng. Test Case), test sonuçları gibi kod ve test iş ürünleri (İng. Artifact), bu aşamanın sonunda üretilir.
- Değerlendirme evresi: Nihai yazılım ürünü, müşteriler tarafından değerlendirilir ve test edilir. Bir sonraki spiralle devam etmeden önce, geri bildirim ve onaylar alınır.



Şekil 2.5: Spiral Model [15]

2.2. Çevik Yazılım Geliştirme Yöntemleri

Günümüzde modern iş ortamı, yeni fırsatlar yakalamak ve müşterilerin taleplerini karşılamak için bilgisayar tabanlı sistemlerin ve yazılım ürünlerinin giderek yaygınlaşmasıyla, hızlı bir tempoyla çalışmaktadır. Geleneksel geliştirme yöntemleri çoğunlukla ağır dokümantasyona bağlı olduğundan, yazılım geliştirme süreci uzun sürmektedir; müşterilerin taleplerini karşılamak veya geri bildirimlerine dönüş sağlayabilmek, bu yöntemlerde kolaylıkla yapılamamaktadır. Oysa hızlı bir geliştirme ve teslimat, büyük miktarda pazar payı sağlamak için çok önemlidir.

Değişime ve hızlı geliştirmeye ayak uydurabilen bir geliştirme sistemine duyulan ihtiyaç, çevik yazılım geliştirmenin ortaya çıkmasına zemin hazırlamıştır. 2001 yılında çevik geliştirme felsefesini sunan çevik manifesto [16], Kent Beck ve diğer birçok yazılım geliştiricisi tarafından imzalanmıştır. Bu manifesto şu şekildedir:

"Bizler daha iyi yazılım geliştirme yollarını, uygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkartıyoruz.

Bu çalışmaların sonucunda:

Süreçler ve araçlardan ziyade *bireyler ve etkileşimlere,*

Kapsamlı dökümantasyondan ziyade *çalışan yazılıma,*

Sözleşme pazarlıklarından ziyade *müşteri ile işbirliğine,*

Bir plana bağlı kalmaktan ziyade *değişime karşılık vermeye,*

değer vermeye kanaat getirdik.

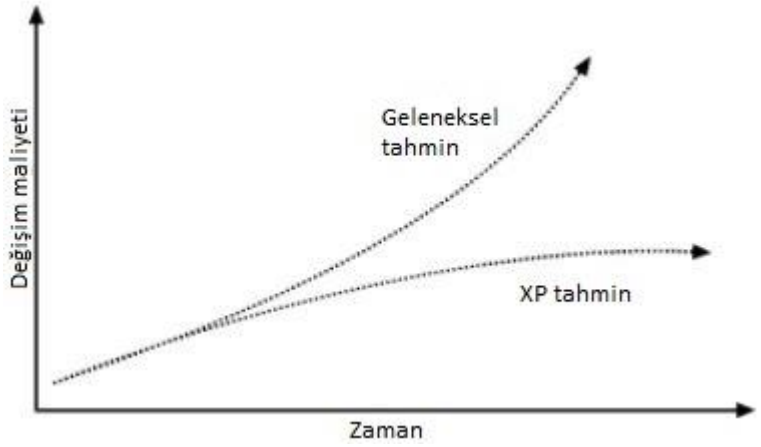
Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte,

sağ taraftaki maddeleri daha değerli bulmaktayız."

Çevik yöntemler genelde küçük veya orta ölçekli ürünler geliştirmek için kullanılır. Uç Programlama (İng. Extreme Programming-XP), Scrum, Yalın Geliştirme (İng. Lean Development) ve diğer popüler çevik yöntemler, aşağıdaki bölümlerde açıklanmıştır.

2.2.1. Uç Programlama

Uç programlama basitlik, iletişim, geri besleme, cesaret ve saygı olmak üzere beş değerden oluşur. Uç programlamanın ana değeri, yapılan her şeyde basitlik uygulamaktır. İleri düzey tasarımlara odaklanmak, müşterilerin istemeyecekleri uzun zaman harcama riskini arttırır [17]. Karmaşık olmayan kod geliştirmek, geliştiricinin kodları kolayca değiştirebilmesini ve müşterilerin farklı istekleri olması durumunda yeni tasarıma ayak uydurabilmesini sağlar. Buna ek olarak, karşılaşılan her soruna basit çözümler uygulamak, zaman içindeki değişim maliyetini düşürür. Şekil 2.6'da değişim maliyetinin, geleneksel ve uç programlama varsayımlarında nasıl farklılaştığı görülmektedir.



Şekil 2.6: Bir XP projesinin değişiklik maliyeti [17]

İletişim, bir projede başarılı olmak için önemli bir role sahiptir. Proje takımı üyeleri, yönetici ve müşteri arasında, iletişim eksikliği nedeniyle sorunlar ortaya çıkabilir. Bir projede sağlıklı bir iletişim aşağıdaki önlemlerle sağlanabilir.

- Mevcut sistemde değişiklik olması durumunda, projedeki sorumlu kişiler bilgilendirilmelidir.
- Müşteriler, bazen arzuladıkları ürün özelliklerini anlatırken yeterince açık ifadeler kullanmayabilir; bu sebeple geliştiriciler ürünün ayrıntılarını, özellikle de kritik noktaları sormaya ve öğrenmeye dikkat etmelidir.
- Yöneticiler, geliştiriciler tarafından yanlış anlaşılmamak için, proje ilerleme durumu hakkında bilgi alışverişinde bulunurken dikkatli olmalıdır.

Geliştirme boyunca somut geri bildirimler, doğru ürünün teslim edilebilmesi ve teslimden sonra değişiklik talepleri durumlarına uyum sağlanabilmesi açısından çok değerlidir. Uç programlamada kısa yinelemeler, müşteriden zamanında geri bildirim alınabilmesi için teşvik edilir. Müşteri, her yinelemede ürünün durumunu değerlendirebilir ve gerekirse yeni özellikler isteyebilir. Geliştiriciler görevlerin düzgün çalışıp çalışmadığını kontrol etmek için, birim testleri yazarlar. Kabul testleri, hikâyelerin (İng. Stories) beklendiği gibi çalışıp çalışmadığını kontrol etmeyi sağlar.

Kent Beck kitabında [18] cesareti, korku karşısında etkili bir eylem olarak tanımlamıştır. Uç programlama takımları, geri bildirimi iletmek ve kabul etmek için cesarete sahiptirler. Cesaret, mevcut yazılım sisteminin, geri bildirime göre yeniden

yapılandırılmasına yardımcı olur ve geliştirme sürecinde meydana gelen değişiklikler karşısında, korkunun motivasyonu azaltmasına izin vermez.

Tüm takım üyeleri, insanlık ve yaptıkları iş açısından değerlidir. Bu nedenle, takım üyeleri birbirlerine saygı göstermeli ve uç programlamanın düzgün işleyebilmesi için çalışmalarına saygı duymalıdır. Ayrıca, takım üyeleri sadece kendi aralarında değil, müşterilerle de karşılıklı olarak birbirlerinin uzmanlığına saygı duyarlar.

Uç programlama için Kent Beck kitabında [18], bu geliştirme modelinde kullanılan 13 temel uygulamayı şu şekilde listelemiştir: Birlikte oturma, bütün takım, bilgilendirici çalışma alanı, enerji veren iş, eşli programlama (İng. Pair Programming), hikâyeler, haftalık çevrim, üç aylık çevrim, gevşeklik (İng. Slack), on dakikada inşa (İng. Ten-minute Build), sürekli entegrasyon (İng. Continuous Integration), önce test programlama (İng. Test-First Programming), artırımlı tasarım.

1. Birlikte oturma: Birlikte oturmak, insanların etkileşimde bulunmasına ve birlikte çalışmasına yardımcı olur. Takımlar, hem konuşabilecekleri ve etkileşimde bulanabilecekleri işbirliğine dayalı bir alana hem de bireylerin gerektiğinde odaklanabilmeleri için özel alana ihtiyaç duyar. Bu basit uygulama, çoğu kuruluş için en zor olanlardan biridir.
2. Bütün takım: Projenin başarılı olması için gerekli tüm beceri ve bakış açıları takım üyelerinde vardır.
3. Bilgilendirici çalışma alanı: Çalışma alanı sadece insanları bir arada tutmak için kullanılan bir yer değil, aynı zamanda paydaşlara (İng. Stakeholders) ve takım üyelerine yapılan işin hikâyesinin anlatıldığı bir yer olmalıdır. Duvarlara asılan hikâye kartları ve takım norm listeleri, takımın hedeflerini ve ilerlemelerini gösterir.
4. Enerji veren iş: Takım üyeleri, üretken olabildiği ve tutarlı bir şekilde sürdürülebildiği kadar çok çalışmalıdır.
5. Eşli programlama: İki kişinin bir klavye ile aynı proje üzerinde işbirliği yaparak yazılım geliştirmesidir. Genel olarak bir kişi konuşur ve izlerken diğeri kodlama yapar ve iki kişi arasında roller sık sık değiştirilir. Bu uygulama analiz, tasarım ve test de dâhil olmak üzere, birçok işlev üzerinde faydalı olabilir.

6. Hikâyeler: Müşteriye görünür işlevsellik birimleri olan kullanıcı hikâyeleri ile iş planı yapılır. Proje takımı, maliyet tahmini ve proje yönetimi için kullanıcı hikâyelerini temel alır.
7. Haftalık çevrim: XP'de her seferinde bir haftanın planlaması yapılır. Scrum'da yaklaşık iki haftalık döngü varken XP her hafta yeniden planlamayı önerir.
8. Üç aylık çevrim: Her seferinde çeyrek çalışma planı oluşturulur. Üç aylık çevrim, takımın büyük resmi yansıtmasını ve dikkate almasını sağlar; böylece takım proje hedefleri ile uyum içinde ilerler.
9. Gevşeklik: Genel taahhütleri yerine getirmek için gerektiğinde, ihmal edilebilecek bazı küçük görevler eklenebilir. Fazladan taahhüt eklenerek yetersiz teslim yerine, gerçekçi ve ulaşılabilir hedefler belirlenmelidir.
10. On dakikada inşa: On dakika içinde bütün sistem kurulur ve tüm testler koşturulur. On dakikadan uzun süren bir yapı, daha az sıklıkla kullanılacak ve geri bildirim fırsatının kaçırılmasına sebep olacaktır.
11. Sürekli entegrasyon: Değişiklikler en fazla birkaç saat sonra entegre edilmeli ve test edilmelidir. Sürekli entegrasyon, sistemin çalıştığını kanıtlamak için geri bildirim sağlar.
12. Önce test programlama: Geliştiricinin iş problemlerini çözdüklerini kanıtlamak için kullanacağı testleri yazmasıyla başlayan güçlü bir uygulamadır.
13. Artırımlı tasarım: Önceki gün elde edilen bilgiler kullanılarak her gün tasarıma yatırım yapılır, böylece tasarım sürekli ihtiyaca dayalı olarak gelişir. Küçük kararlar vermek, değişim maliyetini düşürmeye yardımcı olur ve kararların ihtiyaç noktasına daha yakın olmasını sağlar.

2.2.2. Scrum

Scrum, 1990'ların başında Jeff Sutherland ve geliştirme ekibi tarafından tasarlanan çevik bir yazılım geliştirme yöntemidir. Scrum ilkeleri, çevik manifesto ile uyumludur ve süreç içindeki geliştirme faaliyetlerine rehberlik etmesi için kullanılır. Bu geliştirme faaliyetleri gereksinimler, analiz, tasarım, evrim ve teslimattan oluşur. Her bir Scrum çerçevesi etkinliği içinde, iş görevleri 'sprint' adı verilen bir süreç modeli kapsamında gerçekleşir [19]. Bu süreç modellerinin her biri, bir dizi geliştirme eylemini tanımlar.

İş listesi (İng. Backlog), müşteri için iş değeri (İng. Business Value) sağlayan proje gereksinimlerinin ya da özelliklerinin önceliklendirilmiş bir listesidir. Öğeler, herhangi

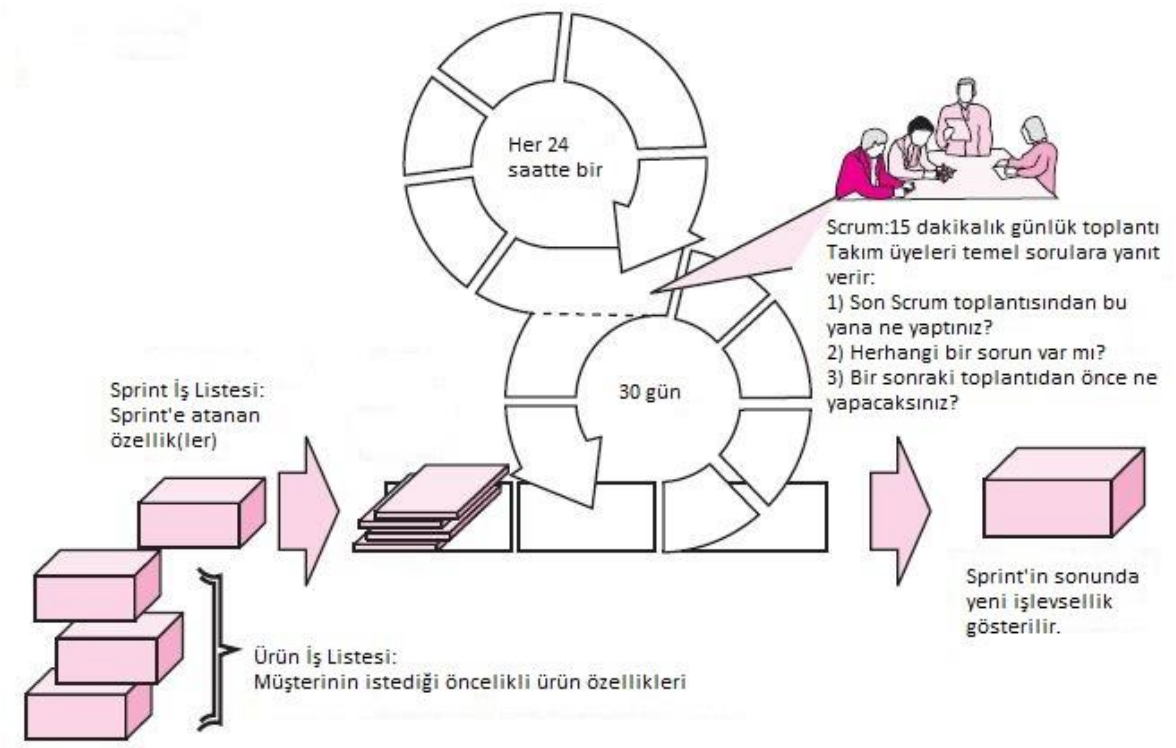
bir zamanda iş listesine eklenebilir. Ürün sahibi (İng. Product Owner), iş listesini değerlendirir ve öncelikleri gerektiği gibi günceller. Scrum Master, takımın bir antrenör gibi Scrum değerlerine ve uygulamalarına devam etmesini sağlayan kişidir. Karşılaşılan engelleri kaldırır, toplantıları kolaylaştırır ve ürün sahipleriyle birlikte çalışır. Scrum Master, takım üzerinde yetki sahibi olmayan, ancak süreç üzerinde yetki sahibi olan, hizmetkâr bir liderdir. İnsanları işten çıkartamazlar, fakat sprint'lerin ne kadar süreceğini değiştirebilirler.

Sprint'ler, genellikle bir ay veya daha az zaman sınırı olan önceden tanımlanmış gereksinimleri yerine getirebilmek için yapılması gereken iş ünitelerinden oluşur. Sprint sırasında iş listesindeki öğeler üzerinde değişiklik yapılmaz. Dolayısıyla her sprint, takım üyelerinin kısa süreli ancak istikralı bir ortamda çalışmasına olanak sağlar.

Scrum toplantıları, günlük olarak Scrum takımı tarafından kısa süreli (genellikle 15 dakika) yapılan toplantılardır. Toplantıda aşağıda belirtilen üç kilit soru üzerinde yoğunlaşılır ve tüm takım üyeleri bu soruları yanıtlar.

- Son takım toplantısından beri ne yaptınız?
- Karşılaştığınız sorunlar nelerdir?
- Bir sonraki takım toplantısına kadar ne yapmayı planlıyorsunuz?

Scrum etkinliği için gerekli sprint sayısı, ürünün karmaşıklığına ve boyutuna bağlı olarak değişir. Scrum sürecinin genel akışı Şekil 2.7'de gösterilmektedir.



Şekil 2.7: Scrum süreç akışı [19]

2.2.3. Yalın Yazılım Geliştirme Modeli

Yalın yazılım geliştirme (İng. Lean Software Development), yalın üretim ilkelerini yazılım mühendisliği dünyasına uyarlamıştır. Yalın yazılım geliştirme yönteminde temel fikir, israfları en aza indirirken müşteri değerini en üst düzeye çıkarmaktır [20]. Daha az kaynak kullanarak daha fazla müşteri için bir değer oluşturulması amaçlanmaktadır. Bu yazılım geliştirme yöntemi, Stober'in kitabında [21] bahsettiği, aşağıda yer alan şu yönergeleri içermektedir.

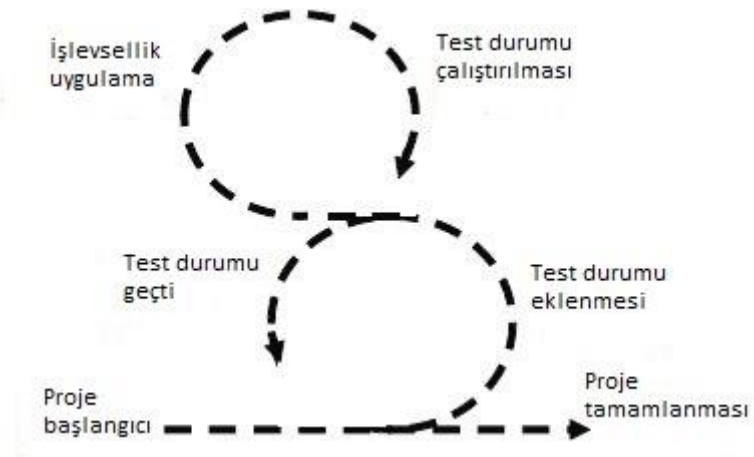
1. İsrafları bertaraf etmek: Yazılım geliştirmede israf, birçok açıdan ortaya çıkabilir. Yazılım mühendisleri parlak teknolojilere çok daha fazla odaklanmaya eğilim gösterirler ve gerçek işletme değerini ihmal ederler. Bu nedenle, müşteri değeriyle ilgisi olmayan herhangi bir geliştirme çalışmasından kaçınılmalıdır. Örneğin, mevcut projenin zamanı çerçevesinde gerçekte uygulanabileceğinden daha fazla gereksinimin detaylandırılması ve tasarlanması, israfa sebep olur.
2. Öğrenmeye odaklanmak: Planlama bir dereceye kadar yararlı olsa da öğrenme şarttır. Çevik projeler prototiplemeyi, geri bildirim ve iyileştirmeler için bir kaynak olarak teşvik eder. Prototip; tasarımları, tahminleri, proje

ilerlemesini ve müşteri gereksinimlerini doğrulamak için güvenilir bir vasıtadır. Bir prototip mevcut standartlara meydan okumak ve yeni çözümler geliştirmek için iyi bir yoldur. Ayrıca, başarısızlıkları analiz etmek, temel nedenlerini araştırmak ve aynı başarısızlığın tekrar ortaya çıkmayacağından emin olmak da öğrenme kapsamındadır.

3. Dâhili kalite: Test takımlarının çok fazla hata bulması, geliştirme sürecinin gerektiği gibi çalışmadığını gösterir. Test odaklı bir geliştirme, başlangıçtan itibaren uygun geliştirme kodunu oluşturur. Ancak kalite, sadece hatasız koddan ibaret değildir. Kalite, müşterilerin beklentilerinin karşılandığı anlamını taşır. Bu nedenle, müşterilerin gereksinimlerini ve müşteri değerini oluşturan unsurları anlamak büyük bir öneme sahiptir.
4. Erteleme taahhüdü: Projenin bütününü kapsayan mükemmel bir plan oluşturulmasının zorluğundan dolayı geliştirmeye, tam bir şartname üzerinden değil de birçok küçük artırımsal adım üzerinden yaklaşılması gerekir. Genel sistem mimarisi, herhangi bir zamanda herhangi bir özellik eklenmesini desteklemelidir. Tasarım, belli bir sorun için çözüm seçenekleri içermelidir. Geri alınamayacak kararlar, prototipleme gibi iyi bir hazırlıkla alınmalıdır.
5. Hızlı teslimat: Aşırı iş yükü, takımların ilerlemesini yavaşlatacaktır. Takımlar üzerine büyük bir yığın iş yüklemek yerine süreç içinde daha az ve sık iş verilerek takımların iş bitirme zamanı azaltılabilir. İş yükünü gerçekçi bir kapasiteye sınırlayan, güvenilir, tekrarlanabilir ve sürdürülebilir her adım, yüksek kalitede ve düşük maliyette hızlı teslimatla tamamen uyumludur.
6. İnsana saygı: Ortak bir hedef doğrultusunda ortaklaşa çalışan motive bir takım, sürdürülebilir rekabet avantajı sağlar. Bu yönerge gurur, bağlılık, saygı ve güveni teşvik eden bir liderlik anlamına gelir. Böyle bir çalışma ortamında, mevcut yeteneklerdeki yaratıcılık, zekâ ve kararlılık ortaya çıkar.
7. Bütünü optimize etmek: Bütünü optimize etmekle, tüm değer akışına bakmak kastedilmektedir. Bütünün optimize edilmesi için hem ürünün kendisi hem de ürünü geliştirme süreci sürekli olarak gözden geçirilmelidir. Sonuçların, geliştirme zamanı ya da müşteri memnuniyeti ölçümü gibi yöntemlerle sayısallaştırılması gerekir.

2.2.4. Test Güdümlü Geliştirme Modeli

Test güdümlü yazılım geliştirmede (İng. Test Driven Development-TDD) yazılım bileşen gereksinimleri, arayüzü kullanan ve bileşen tarafından gönderilen veri yapılarında ve işlevlerde hatalar bulmaya çalışan bir dizi test durumlarının (İng. Test Case) yaratılmasına temel oluşturur. Test güdümlü yazılım geliştirmede kaynak kodunun oluşturulmasından önce test durumları tasarımının önemi vurgulanır [19]. Geliştirilecek yazılımın işlevselliğinin gerçekleştiriminden (İng. Implementation) önce test durumları yazılır. Test durumları gerçekleştirilirken hangi fonksiyonun sağlanacağına ve nasıl davranması gerektiğine göre, tasarımın geçerliliği de doğrulanır. Kod, test durumunu başarılı bir şekilde geçtikten sonra döngü, ek ya da geliştirilmiş test durumuyla tekrar başlar ve fonksiyonun gerçekleştirimiyle devam eder. Geliştirici, bir kullanım senaryosu (İng. Use Case) için test durumu biter bitmez kodunu geliştirmeye başlar [21]. Şekil 2.8'de test güdümlü geliştirme döngüsü gösterilmiştir.

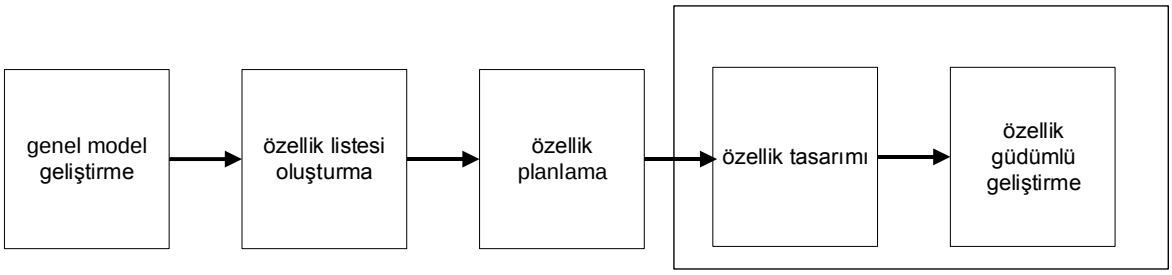


Şekil 2.8: Test güdümlü geliştirme döngüsü [21]

2.2.5. Özellik Güdümlü Geliştirme Modeli

Özellik Güdümlü Geliştirme (İng. Feature Driven Development-FDD), gelişmekte olan sistemler için çevik ve uyarlanabilir (İng. Adaptive) bir yaklaşımdır. FDD yaklaşımı, tüm yazılım sürecini kapsamaz; aksine tasarım ve geliştirme aşamalarına odaklanır. Bununla birlikte, bir yazılım geliştirme projesinin diğer faaliyetleriyle birlikte çalışmak üzere tasarlanmıştır [22, 23]. FDD, yazılımın pek çok farklı özelliğe (İng. Feature) bölünmesiyle elde edilen yinelemeli ve aşamalı bir yöntem olup her bir özelliği ayrı ayrı oluşturur. FDD yaklaşımında geliştirme süreci beş faaliyetten

oluşur: Genel model geliştirme, özellik listesi oluşturma, özellik planlama, özellik tasarımı ve özellik güdümlü geliştirme (Şekil 2.9) [23].



Şekil 2.9: Özellik-Güdümlü Geliştirme Süreçleri [23]

Geliştirme sürecinin ilk faaliyeti olan genel model geliştirmede, sorun alanının daha iyi anlaşılabilmesi için, projenin genel üst düzey görünümü oluşturulur. Bu adım takım için, takım üyeleri ve müşteri arasındaki ilişkiyi ve etkileşimi yönetmek bakımından önemli bir yere sahiptir. Özellik listesi oluşturma faaliyetinde, takım gereken özelliklerin ya da işlevlerin bir listesini oluşturmak için, ilk faaliyetten elde edilen bilgileri kullanır ve bunları iş konularına göre kategorilere ayırır. Özellik planlama aşaması proje yöneticisi ve geliştirme müdürü tarafından belirlenen geliştirme planı, özellik sahipliği (İng. Ownership of feature), zaman çizelgesi ve özellik geliştirme takımının sorumluluklarını içerir. Bir sonraki faaliyette özelliği tasarlarken odak; programlama işlerine, tanımlanan sınıflara ve metotlara, sunulan ardıl-işlem diyagramlarına (İng. Sequence Diagrams) göre, özelliklerin kendisine doğru kayar. Son faaliyette ise her bir özelliğin nerede geliştirileceği oluşturulur [24].

2.2.6. Dinamik Sistem Geliştirme Modeli

Dinamik Sistem Geliştirme (İng. Dynamic Systems Development-DSD) modelinin temelindeki ana fikir, bir üründeki işlevselliği tespit edip ardından bu işlevselliğe erişmek için zaman ve kaynakları ayarlamak yerine, zaman ve kaynakları tespit ettikten sonra işlevsellik miktarını buna göre ayarlamaktır.

DSD yöntemi beş aşamadan oluşur: Fizibilite çalışması, iş çalışması, işlevsel model yineleme, tasarım ve geliştirme yinelemesi ve gerçekleştirme. İlk iki aşama ardışıktır ve yalnızca bir kez yapılır. Asıl geliştirme çalışmalarının yapıldığı son üç aşama, yinelemeli ve artırımlıdır. DSD yönteminde yinelemeler (İng. Iteration), belirli bir görev için ayrılmış esnek olmayan zaman dilimleri ya da zaman kutuları (İng.

Timebox) halinde gerçekleştirilir. Tipik bir zaman kutusunun süresi genellikle, birkaç günden birkaç haftaya kadar uzar [22].

- Fizibilite çalışması (İng. Feasibility study): Önerilen yöntemin uygulanabilir olup olmadığı ve var olan problemlerin ortaya çıkarılması için kapsamlı bir araştırma yapılmaktadır.
- İş çalışması (İng. Business study): İş akışının ve süreçlerin birbiriyle nasıl ilişkili olduğunun net bir şekilde anlaşılabilmesi ile ilgilidir. Bu çalışma, projeye katılanların ve proje paydaşlarının belirlenmesini kapsar.
- İşlevsel model yineleme (İng. Functional model iteration): İş çalışması sırasında tanımlanan sistemlerin işlevlerini ve üst düzey iş bilgi gereksinimlerini tasfiye etmek üzerine çalışır. Bu aşamada riskler tanımlanır ve gelecekteki geliştirmelerde bu risklerle nasıl başa çıkılacağı üzerine bir plan yapılır. İşlevsel model yinelemenin sonucu olarak yazılımın standart analiz modeli ortaya çıkar.
- Tasarım ve geliştirme yinelemesi (İng. Design and build iteration): Bu aşamada gerçek sistem, önceki aşamada gerçekleştirilen işlevsel olmayan gereksinimlere dayanılarak oluşturulur ve test tamamlandıktan sonra yerleşik sistem bir sonraki aşamada uygulanır.
- Gerçekleştirim (İng. Implementation): Son olarak gerçekleştirme aşaması, sistemin geliştirme ortamından gerçek üretim ortamına aktarıldığı aşamadır. Nihai ürün hakkında gerekli eğitimler ya da bilgiler kullanıcılara verilir ve sistem onlara teslim edilir. DSD metodu olası dört geliştirme süreci tanımlamaktadır [22]: 1) Eğer sistem bütün gereksinimleri karşılıyorsa başka herhangi bir işe gerek kalmamıştır. 2) Ancak önemli bir miktarda gereksinim yerine getirilmemişse süreç, baştan sona tekrarlanabilir. 3) Eğer bazı kritik olmayan işlevsellikler ihmal edilmişse süreç, işlevsel model yineleme evresinden itibaren tekrar çalıştırılabilir. 4) Son olarak, zaman kısıtlamaları nedeniyle bazı teknik sorunlar çözülememişse tasarım ve geliştirme yinelemesi aşaması tekrarlanarak bu sorunların üstesinden gelinebilir.

2.3. Yazılım Ölçümü ve Metrikler

Fenton kitabında [25] bir zamanlar belirsiz ve anlaşılması zor olan yazılım ölçümünün, iyi bir yazılım mühendisliği için zorunlu hale geldiğini belirtmiştir.

Tasarımın istenilen kalitede olup olmadığı, ürünün piyasaya sürülmeye hazır olup olmadığı, gereksinimlerin tutarlı ve eksiksiz bir şekilde karşılanıp karşılanmadığı gibi birçok konuda bilgi sahibi olabilmek için, yazılımların özellikleri ölçülmektedir.

Ölçme, sayıları ya da sembolleri, gerçek dünyadaki varlıkların açıkça tanımlanmış kurallarla karakterize edilmiş niteliklerine atama sürecidir. Bu nedenle ölçüm:

- Varlıkları (ilgi nesneleri),
- Nitelikleri (varlıkların özellikleri),
- Niteliklere değer atamak için kuralları (ve ölçekleri) gerektirir [26].

Her ölçme faaliyeti, açıkça tanımlanmış ve kolay anlaşılır olan belirli bir hedef ya da ihtiyaç tarafından motive edilmelidir. Ölçme hedefleri kesin, açık ve yöneticilerin, geliştiricilerin ve kullanıcıların bilmeleri gerekenlerle bağlantılı olmalıdır [25]. Bu hedefler, ölçüm bilgisinin toplanmasından sonra nasıl kullanılacağı konusunda yol göstericidir.

Ölçme eylemi anlama, kontrol ve iyileştirme hedeflerine yönelik faaliyetler açısından önemli bir yere sahiptir [25]. Öncelikle geliştirme ve bakım boyunca neler olduğunun anlaşılmasına yardımcı olan metrikler (İng. Measure) vardır. Böylece mevcut durum değerlendirilerek geleceğe yönelik hedeflerin belirlenmesi sağlanır. Bu anlamda ölçme, faaliyetler ve bu faaliyetlerin etkiledikleri varlıklar arasındaki ilişkinin daha iyi anlaşılmasını sağlayarak süreç ve ürünün özelliklerini daha görünür kılar.

Ölçme, projelerde olan bitenin kontrol edilmesine olanak sağlar. Temel çizgileri ve hedefleri anlayarak, neler olabileceği öngörülebilir; süreçler ve ürünler üzerinde değişiklikler yapılarak hedeflere ulaşılmasında kolaylık elde edilmiş olur. Örneğin, kabul edilebilir sınırları aşan kod modüllerinin karmaşıklığı, kapsamlı bir gözden geçirme yaparak izlenebilir [25].

Ölçme, süreçlerin ve ürünlerin iyileştirilmesini teşvik eder. Örneğin, şartname kalitesinin ölçütlerine ve muhtemel tasarım kalitesinin öngörülerine dayanarak, yapılan tasarım incelemelerinin sayısı ya da türü artırılabilir [25].

Yazılım metrikleri maliyet ve çaba tahminleri, verimlilik metrikleri ve modelleri, performans değerlendirmesi ve modelleri, yetkinlik-olgunluk değerlendirmesi gibi yazılım ölçümü içeren birçok kavramı ve etkinliği ilgilendiren bir terimdir [25]. Yazılım

metriği verileri belirli bir amaç gözetilerek toplanmalıdır. Veriler toplanıp kullanılmadan önce, bilginin ölçeğinin göz önünde bulundurulması önemlidir [27].

Yazılım metrikleriyle ilgili olarak beş temel ölçek türü kullanılmaktadır: Nominal, sıralı, aralıklı, oransal ve mutlak.

- Nominal ölçek (İng. Nominal Scale): Bu ölçek türü, nesnelerin niteliklerine göre sınıflandırılmasında kullanılır. Ampirik ilişki sistemi yalnızca farklı sınıflardan oluşur; sınıflar arasında sıralama yoktur. Örnek olarak programlama dil adları, iş fonksiyonları, problem türleri verilebilir.
- Sıralı ölçek (İng. Ordinal Scale): Sınıflar, niteliklerine göre sıralanır. Nicel karşılaştırma yoktur. Örneğin programcı becerilerinin düşük, orta, yüksek olarak sıralanması.
- Aralıklı ölçek (İng. Interval Scale): Bir noktadan diğerine olan uzaklığı tanımlar; böylece ardışık sayılar arasında eşit aralıklar bulunur. Bu özellik, aralıklar üzerinde ortalama değeri hesaplama gibi sıralı ölçekte bulunmayan hesaplamalara izin verir. Aralıklı ölçekte farklılıklar anlamlı görünmekle birlikte mutlak bir sıfır noktası yoktur [28]. Örneğin karmaşıklık değeri 6 olan bir program, karmaşıklığı 2 olan programdan 4 birim daha karmaşıktır; ancak ilk programın, ikinci programın 3 katı kadar karmaşık olduğunu söylemek muhtemelen anlamlı değildir [27].
- Oransal ölçek (İng. Ratio Scale): Veri değerleri, mutlak sıfır değerine sahip olan ve anlamlı oranlar hesaplanmasına izin veren bir ölçekle ilişkilendirilir. Örneğin, 2.000 satırlık bir program, 1.000 satırlık bir programın 2 katı kadardır ve bu programlar, bu ölçeğe göre 0 (sıfır) uzunluğa sahip olabilir.
- Mutlak ölçek (İng. Absolute Scale): Bu ölçekte yapılan ölçüm, basit bir şekilde varlık kümesindeki öğeleri sayarak elde edilir. Yalnızca bir ölçüm eşleştirmesi (İng. Measurement Mapping) gerçek sayı vardır. Elde edilen sayı üzerinde tüm aritmetik analizler anlamlıdır. Örneğin, hata sayısı, bir programda tespit edilen hataların mutlak ölçekte bir ölçümüdür.

2.3.1. ISO/IEC 15939 Yazılım Ölçme Süreci

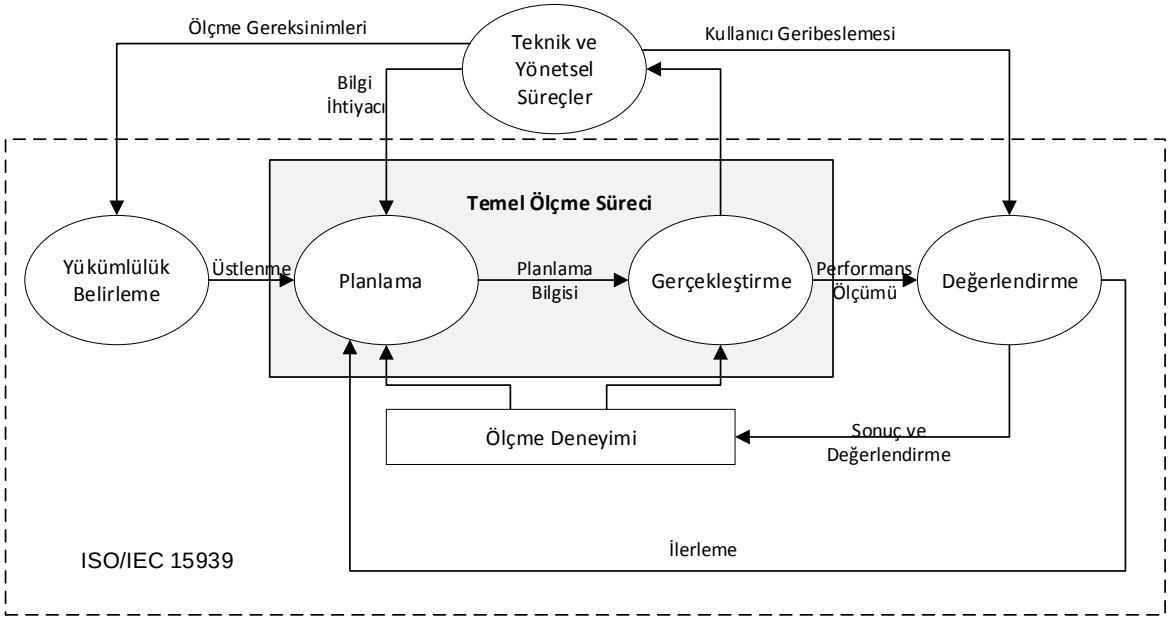
ISO/IEC 15939 standardı [29], bir yazılım ölçme sürecinin uygulanması için gerekli faaliyetleri ve görevleri tanımlar. Faaliyet, yazılım ölçme sürecinin amacına ve

sonuçlarına ulaşılmasına katkıda bulunan bir dizi ilgili görev içerir. Görev, iyi tanımlanmış bir iş bölümüdür. Her faaliyet bir ya da daha fazla görevi kapsar.

Yazılım ölçme süreci, Şekil 2.10'daki süreç modelinde gösterildiği gibi dört faaliyeti kapsar. Faaliyetler, sürekli bir geri bildirim ile ölçme sürecinin geliştirilmesi için tekrar eden bir döngüde sıralanır. Şekil 2.10'daki ölçme süreci modeli, kalite geliştirme için temelde yaygın olarak kullanılan Planla-Uygula-Denetle-Harekete Geç (İng. Plan-Do-Check-Act) döngüsünün bir uyarlamasıdır.

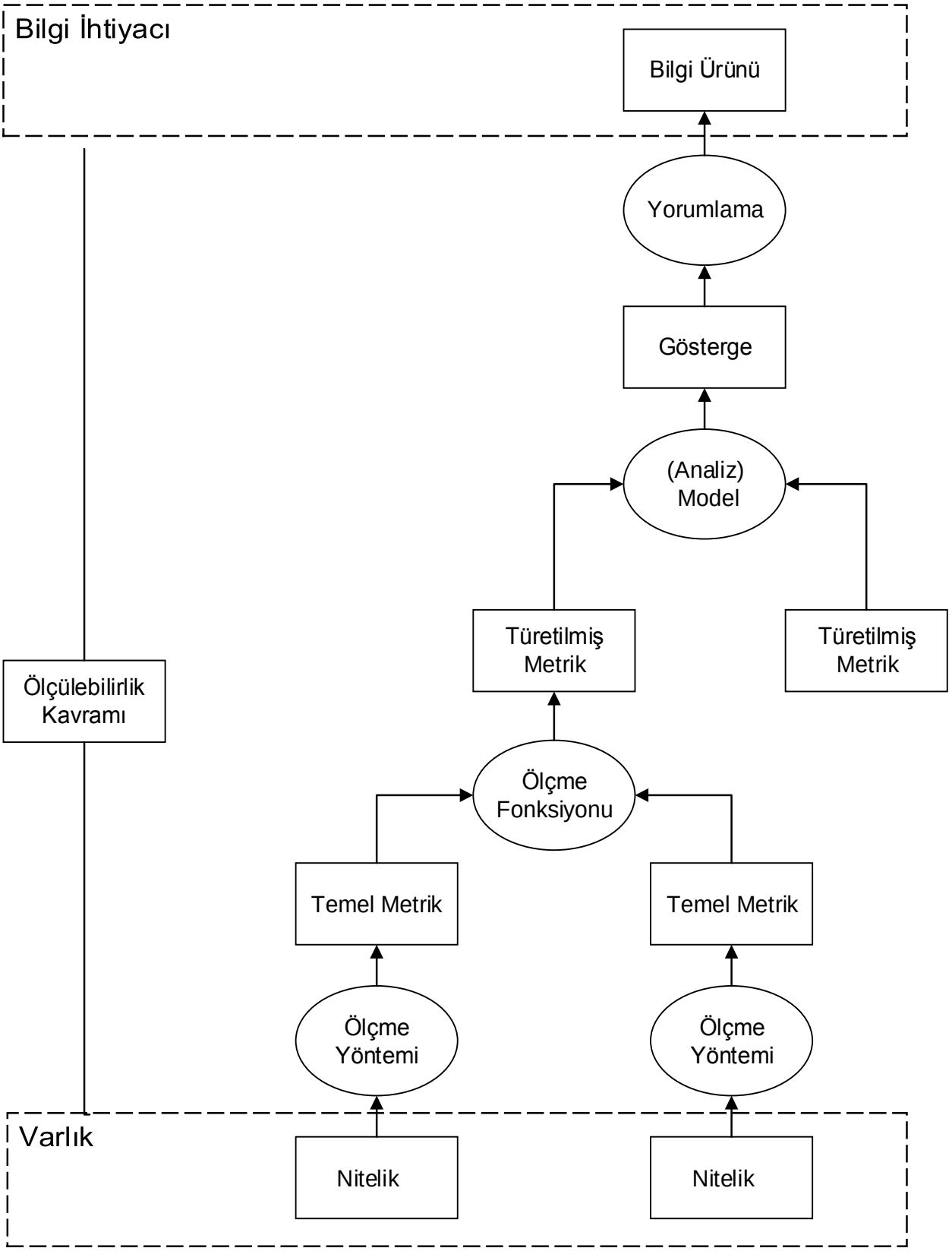
Faaliyetlerin ana hat yapısı ve görevleri şunlardır:

1. Ölçme taahhüdünü oluşturmak ve sürdürmek: Bu faaliyet temel olarak ölçme gereksinimlerinin belirlenmesi ve kaynakların atanması şeklinde iki görevden oluşur. Ölçme gereksinimlerinin belirlenmesi aşamasında ölçme kapsamı belirlenir, yönetim ve personelin ölçmeye bağlılığı taahhüt edilir ve taahhüt, organizasyonel birime bildirilir. Kaynakların atanması aşamasında; organizasyonel birim içerisindeki ölçme işlemi için bireylere sorumluluklar atanır ve ataması yapılan bireylere, ölçme sürecini planlamaları için kaynak sağlanır.
2. Ölçme sürecini planlamak: Bu faaliyet içinde organizasyonel birimi karakterize etme, bilgi ihtiyaçlarını belirleme, metrik seçimi, veri toplama, analiz ve raporlama prosedürlerini tanımlama, bilgi ürünlerini ve ölçme sürecini değerlendirme kriterlerini tanımlama, ölçme görevleri için gözden geçirme, onaylama ve kaynak sağlama, destek teknolojileri edinme ve konuşlandırma görevleri yapılmaktadır.
3. Ölçme sürecini gerçekleştirmek: Bu faaliyette işlemleri bütünleştirme, veri toplama, verileri analiz etme ve bilgi ürünlerini geliştirme, sonuçları bildirme görevleri yer almaktadır.
4. Ölçme sonuçlarını değerlendirmek: Bu faaliyet kapsamında bilgi ürünleri ve ölçüm süreci değerlendirilir ve olası iyileştirmeler tanımlanır.



Şekil 2.10: ISO/IEC 15939 Yazılım Ölçme Süreci Modeli [29]

ISO/IEC 15939 standardında, bilgi ihtiyaçlarını ilgili varlıklara ve bu varlıkların niteliklerine bağlayan bir yapı olarak Ölçme Bilgi Modeli [29] tanımlanmıştır. Ölçme Bilgi Modeli, ilgili niteliklerin nasıl nicelleştirildiğini ve karar verme aşamasına temel oluşturan göstergelere nasıl dönüştürüldüğünü açıklamaktadır. Şekil 2.11'de Ölçme Bilgi Modeli'nde yer alan bileşenler ve ilişkileri gösterilmektedir. Bileşenlerin açıklamaları izleyen paragraflarda verilmiştir.



Şekil 2.11: ISO/IEC 15939 Ölçme Bilgi Modeli [29]

- Varlık (İng. Entity): Niteliklerin ölçülmesiyle tanımlanan nesnelerdir. Tipik yazılım mühendisliği nesneleri ürünler (örn. Tasarım belgesi, kaynak kodu ve test durumu), süreçler (örn. Tasarım süreci ve gereksinim analiz süreci), projeler ve kaynaklar (örn. Programcılar ve test edenler) olarak sınıflandırılabilir. Bir varlık, bilgi ihtiyaçlarını karşılamak için bir ya da birden fazla niteliğe sahip olabilir.
- Nitelik (İng. Attribute): Bir varlığın, niceliksel ya da nitel olarak insan ya da otomatik araçlarla ayırt edilebilen karakteristiğidir. Bir varlık birden çok niteliğe sahip olsa da bunlardan ancak bazıları ölçüm için kullanılabilir. Ölçme Bilgi Modeli'nin belirli bir örneğini tanımlamanın ilk adımı, ölçüm kullanıcısının bilgi gereksinimlerine uygun nitelikleri seçmektir. Belirli bir nitelik, farklı bilgi ihtiyaçlarını destekleyen, çok sayıda ölçme bilgi modeline dâhil edilebilir.
- Temel Metrik (İng. Base Measure): Bir nitelik ve onu nicelleştirme yöntemi açısından tanımlanan bir ölçüdür. Temel metrik, diğer metriklerden fonksiyonel olarak bağımsızdır ve tek bir nitelik hakkında bilgi içerir.
- Ölçme Yöntemi (İng. Measurement Method): Bir niteliğin belirli bir ölçeğe göre sayısallaştırılmasında kullanılan mantıksal işlemler dizisidir. İşlemler, olayların sayılması ya da zamanın gözlemlenmesi gibi faaliyetleri içerebilir. Aynı ölçme yöntemi birden fazla nitelik için uygulanabilir. Bununla birlikte, bir nitelik ile yöntemin her eşsiz kombinasyonu, farklı bir temel metrik üretir. Ölçme yönteminin türü, bir niteliğin nicelleştirilmesi için kullanılan işlemlerin tipine bağlı olarak ikiye ayrılır: Öznel (İng. Subjective) ölçme, insan yargısı içeren nicelleştirme ve nesnel (İng. Objective) ölçme, sayma gibi sayısal kurallara dayalı nicelleştirme olarak tanımlanmaktadır.
- Ölçme Birimi (İng. Unit of Measurement): Aynı türdeki iki büyüklüğün bir sayı olarak ifade edilerek bunların karşılaştırılmasını sağlayan, genel kabul ile tanımlanmış büyüklüktür.
- Türetilmiş Metrik (İng. Derived Measure): Temel metriklerin iki ya da daha fazla değerinin bir fonksiyonu olarak tanımlanan bir ölçüdür. Türetilmiş metrikler, birden fazla varlığın birden fazla niteliği ya da aynı niteliği hakkında bilgi toplar.

- Ölçme Fonksiyonu (İng. Measurement Function): İki ya da daha fazla temel metriği birleştirmek için gerçekleştirilen bir algoritma veya hesaplamadır.
- Gösterge (İng. Indicator): Tanımlanmış bilgi ihtiyaçlarına göre bir modelden türetilen, belirli niteliklerin tahmin ya da değerlendirilmesini sağlayan bir öğedir. Göstergeler, analiz ve karar verme için temel oluşturur.
- Model: Bir ya da daha fazla temel ve/veya türetilmiş metriği ilişkili karar kriterleriyle birleştiren algoritma ya da hesaplamadır. Modeller, tanımlanan bilgi ihtiyaçlarıyla ilgili tahminler veya değerlendirmeler üretir. Ölçek ve ölçme yöntemi, göstergeler üretmek için kullanılan analiz teknikleri veya modellerin seçimini etkiler.
- Karar Kriteri (İng. Decision Criteria): Eylem ya da ileri araştırmanın gerekliliğini belirlemek ya da belirli bir sonuca duyulan güven düzeyini tanımlamak için kullanılan sayısal eşikler (İng. Threshold) ya da hedeflerdir. Karar kriterleri, ölçüm sonuçlarının yorumlanmasına yardımcı olur. Karar kriterleri hesaplanabilir veya beklenen davranışın kavramsal bir anlayışına dayanabilir. Karar kriterleri geçmiş verileri, planlar ve sezgiselliklerden türetilebilir veya istatistiksel kontrol limitleri ya da güven sınırları olarak hesaplanabilir.
- Ölçülebilir Kavram (İng. Measurable Concept): Varlıkların nitelikleri ve bilgi ihtiyaçları arasındaki soyut ilişkidir. Örneğin, bir proje grubunun yazılım geliştirme üretkenliğini bir hedefe kıyasla karşılaştırmak için bir bilgi ihtiyacı tanımlanmış olabilir. Bu durumda Ölçülebilir Kavram, “yazılım geliştirme üretkenlik oranı”dır.
- Bilgi Ürünü (İng. Information Product): Bilgi ihtiyaçlarını karşılayan ölçme sürecinin bir sonucudur.

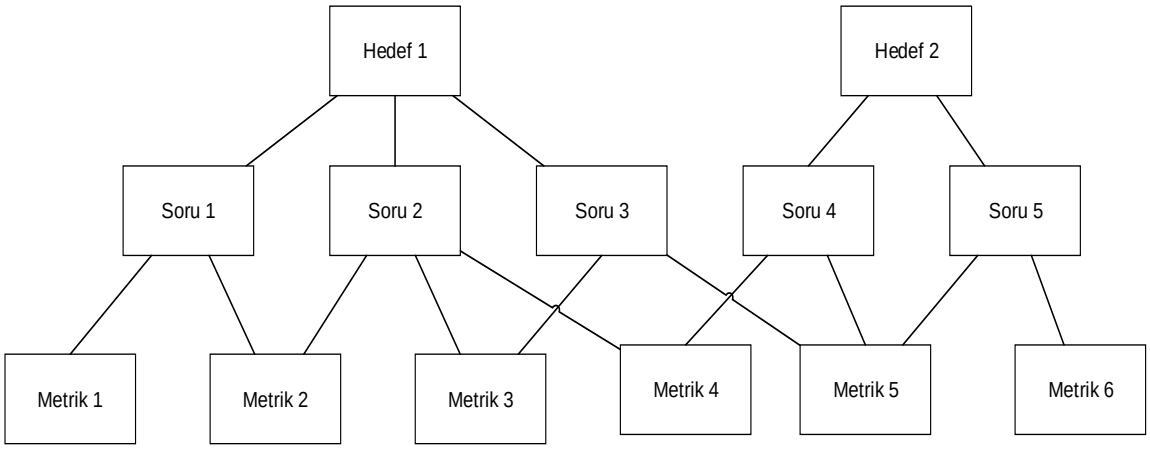
2.3.2. Hedef-Soru-Metrik Çatısı

Hedef-Soru-Metrik Çatısı (İng. Goal-Question-Metric Framework-GQM) [31], yazılım mühendisliği alanındaki süreçlerin ve ürünlerin ölçümünü destekleyecek, hedef odaklı bir yaklaşımın gerekliliğine yanıt olarak geliştirilmiştir. Hedef-Soru-Metrik Çatısı, yazılım süreçleri ve ürünlerini ölçmenin ardındaki hedefleri tanımlamak için, yukarıdan aşağıya yaklaşımı (İng. Top-down Approach) destekler ve tam olarak ne ölçüleceğine (metrik seçimine) karar vermek için bu hedefleri

kullanır. Ayrıca bu çatı, daha önce tanımlanmış amaçlara ve sorulara dayalı olarak verileri yorumlamak için, aşağıdan yukarıya yaklaşımı (İng. Bottom-up Approach) da desteklemektedir [30].

Hedef-Soru-Metrik yaklaşımını uygulamanın sonucunda, belirli bir soru grubunu hedefleyen ölçüm sisteminin spesifikasyonu ve ölçüm verilerinin yorumlanması için bir kurallar seti oluşur [31]. Şekil 2.12'de gösterilmekte olan Hedef-Soru-Metrik yaklaşımının hiyerarşik yapısı, üç seviyeden oluşur:

1. Kavramsal Seviye (Hedef): Belirli bir ortamla ilgili olarak, çeşitli bakış açılarından, çeşitli kalite ve yönetim modellerine göre ve çeşitli nedenlerle, bir nesne (örn. ürün, süreç, proje ya da kaynak) için hedef tanımlanır.
2. Operasyonel Seviye (Soru): Belirli bir hedefin değerlendirilmesi ya da ona ulaşma şeklini karakterize etmek için bir dizi soru tanımlanır. Sorular, seçilen konuya göre ölçüm nesnesini (örn. ürün, süreç, kaynak) karakterize etmeye ve seçilen bakış açısından ölçüm nesnesinin özelliğini saptamaya yardımcı olur.
3. Nicel Seviye (Metrik): Soruları nicel bir şekilde cevaplandırabilmek için her soruyla bir metrik kümesi ilişkilendirilir.



Şekil 2.12: Hedef-Soru-Metrik Çatısı'nın hiyerarşik yapısı [31]

3. SİSTEMATİK LİTERATÜR TARAMASI

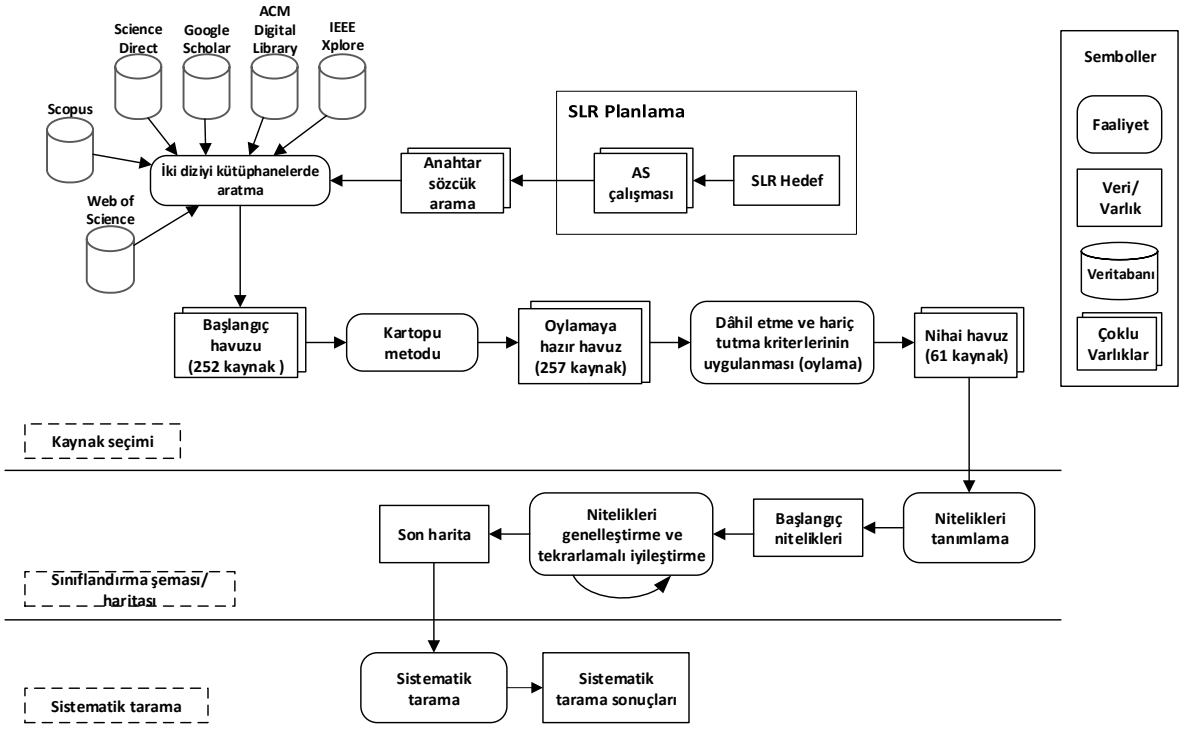
Sistematik literatür tarama (sistematik tarama olarak da anılır) (İng. Systematic Literature Review-SLR), belirli bir araştırma sorusu ya da konu alanı ile ilgili mevcut tüm araştırmaların tanımlanması, değerlendirilmesi ve yorumlanmasıdır [32]. Sistematik taramaya katkıda bulunan her bir çalışmaya, birincil çalışma (İng. Primary Study) adı verilir. Sistematik literatür taraması, ikincil bir çalışma türüdür. Sistematik tarama yapmanın birçok nedeni vardır [32]. Bu nedenler şu şekilde özetlenebilir:

- Bir işlem ya da teknoloji ile ilgili mevcut kanıtları özetlemek. Örneğin, belirli bir çevik yöntemin faydaları ve sınırlamaları hakkında deneysel kanıtları özetlemek.
- Mevcut araştırmalardaki boşlukları belirleyerek gelecek çalışmalar için önerilerde bulunmak.
- Yeni araştırma faaliyetlerini uygun bir şekilde yerine getirmek için arka plan bilgisi sağlamak.

Bu tez çalışmasında, yazılım kurumlarında çevik dönüşümün etkilerini ölçen deneysel kanıtları özetlemek amacıyla sistematik bir literatür taraması gerçekleştirilmiştir. Yürütülen literatür çalışmasının ayrıntıları izleyen alt başlıklarda sunulmuştur.

3.1. Tarama Yöntemi

Sistematik tarama, birçok farklı faaliyeti kapsamaktadır. Sistematik taramalar için mevcut kılavuzlar, tarama faaliyetlerinin sayısı ve sırası hakkında farklı önerilerde bulunmaktadır [33-35]. Bu sistematik literatür tarama çalışmasının yürütülmesinde Kitchenham tarafından önerilen kılavuz [35] kullanılmıştır. Sistematik tarama çalışmasının araştırma süreci Şekil 3.1'de gösterilmiştir.



Şekil 3.1: Sistematik tarama çalışmasında kullanılan araştırma süreci

3.1.1. Hedef ve Araştırma Soruları

Tez kapsamında yürütülen bu sistematik literatür taramasının amacı, bilimsel literatürde bildirilen ölçüm araçlarını ve vaka çalışmalarının sonuçlarını ortaya koyarak yazılım kuruluşlarındaki çevik dönüşümün etkilerini ölçen çalışmaların sentezini sağlamaktır. Bu amaca dayalı olarak, beş araştırma sorusu (AS) tanımlanmıştır.

- AS1: Çevik dönüşümün etkilerini ölçmek için önerilen vasıtalar nelerdir?
- AS2: Hangi metrikler ile ölçüm yapılmıştır?
- AS3: Hangi varlıklar ölçülmüştür?
- AS4: Dönüşümü bildiren çalışmalarda hangi araştırma yöntemleri kullanılmıştır?
- AS5: Çalışmalar tarafından bildirilen dönüşümler olumlu mu olumsuz mu sonuçlanmıştır?

3.1.2. Birincil Çalışmaları Arama

Birincil çalışmaları aramak için ACM, Google Scholar, IEEE Xplore, ScienceDirect, Scopus ve Web of Science dijital kütüphaneleri kullanılmıştır. Çizelge 3.1'de

aramada kullanılan dijital kütüphanelerde başlangıçta eşleşen yayın sayısı gösterilmektedir. Taramanın mümkün olduğunca ilgili tüm birincil çalışmaları kapsamasını sağlayabilmek için, ileriye ve geriye doğru kartopu tekniği (İng. Snowballing) uygulanmıştır [36]. Potansiyel birincil çalışmaların elde edilmesinde aşağıdaki iki cümle kullanılmıştır.

1. `(software AND (agile OR lean) AND (transformation OR transition OR adoption OR migration))`
2. `(software AND (agile OR lean) AND (transformation OR transition OR adoption OR migration) AND (measure* OR metric*))`

Çizelge 3.1: Dijital kütüphanelerde anahtar sözcüklerle eşleşen yayın sayısı

Dijital Kütüphane	URL	Eşleşme sayısı	
		ilk anahtar kelime	ikinci anahtar kelime
ACM	dl.acm.org	250	16
Google Scholar	scholar.google.com	392000	246000
IEEE Xplore	ieeexplore.ieee.org	398	49
ScienceDirect	www.sciencedirect.com	5950	4908
Scopus	www.scopus.com	1057	122
Web of Science	apps.webofknowledge.com	137	12

3.1.3. Dâhil Etme ve Hariç Tutma Kriterleri

Bu sistematik literatür tarama çalışmasında, ilgili tüm çalışmaları dahil etmek ve kapsam dışı çalışmaları da hariç tutmak için aşağıda belirtilen kriterler dikkatle tanımlanmış ve her bir makale için göz önünde bulundurulmuştur.

- Kaynak, bu çalışma kapsamıyla doğrudan ilgili mi?
- Kaynak, İngilizce olarak yazılmış mı ve tam metin olarak mevcut mu?
- Kaynakta sunulanlar, geçerli bir kanıt (örn. deneysel çalışmalar) ile desteklenmiş mi?
- Kaynak dergi, konferans ya da atölye çalışması kapsamında yayınlanmış mı?

Yukarıda yer alan her soru kaynak seçimi aşamasında, cevabı olumlu olanlar için ‘1’ ve olumsuz olanlar için ‘0’ olarak yanıtlanmıştır. Dört ölçütün hepsi için ‘1’ alan kaynaklar çalışma havuzuna dâhil edilmiştir.

3.1.4. Çalışmaların Sınıflandırılma Şeması

Birincil çalışmalar araştırıldıktan ve kartopu tekniği uygulandıktan sonra, çalışmaların özetleri ve tam metinleri hızlıca gözden geçirilerek 257 çalışma elde edilmiştir. Sistematik taramaya dâhil edilecek çalışmaları seçmek için çalışmalar önce dâhil etme/hariç tutma kriterlerine göre oylanmıştır. Başlangıçtaki makale havuzu içinden 196 kaynak, tam metinlerinin detaylı incelenmesiyle kaynak seçim sürecinde hariç tutulmuştur. Sonuç olarak, 61 kaynak veri çıkarımı için son havuza dâhil edilmiştir. Çizelge 3.2'de araştırma sorularıyla ilgili cevapları standartlaştırmak için, gözden geçirme sırasında oluşturulan sınıflandırma şeması gösterilmektedir.

Çizelge 3.2: Sistematik tarama sınıflandırma şeması

AS	Bakış açısı	Veri değer kategorileri
1	Katkı tipi	Metot/teknik, model, metrik, araç, süreç
2	Metrik tipi	Olgunluk seviyesi, hata ile ilişkili, çaba ve çalışma verimliliği
3	Varlık tipi	İş, süreç, ürün, kaynak
4	Araştırma tipi	Doğrulama araştırması, değerlendirme araştırması, deneyim makalesi
5	Dönüşüm sonucu	Olumlu (nicel ve nitel), Olumsuz (nicel ve nitel)

3.2. Tarama Sonuçları

3.2.1. AS1: Çevik dönüşümün etkilerini ölçmek için önerilen vasıtalar nelerdir?

Bu alanda yapılan birincil çalışmaların çoğu (45 çalışma, havuzun %73.7'si) yalnızca ampirik (deneysel) çalışmalar içermektedir. Makale havuzundaki çalışmalardan hiçbiri metot/teknik, araç ya da süreç tipinde katkı yapmamışlardır. Çevik dönüşümün etkisini ölçmek için çalışmalardan ikisi [37, 38] model ve altı tanesi [37-42] metrik önermişlerdir. Bu araştırma sorusunun cevap seti içine dâhil edilemeyen çalışmalar 'diğer' kategorisinde değerlendirilmiştir. 'Diğer' kategorisine dâhil edilen dokuz çalışmanın [43-51] hepsi 'deneyim raporu' (İng. Experience Report) olarak tanımlanmıştır. İzleyen paragrafta, bahsi geçen kategorilerin her biri için çalışmaların katkılarından örnekler sunulmuştur.

Korhonen çalışmasında [42], hata verilerindeki ve hata raporlama uygulamalarında meydana gelen değişiklikleri takip etmek için metrikler sunmuştur. İki çalışma [37, 38], yazılım geliştirme organizasyonundaki çevik dönüşümün etkilerini nicel olarak

ölçmek için metrik modeli sağlamıştır. Heidenberg ve arkadaşları [37], yazılım kuruluşunun çevik ve yalın dönüşüm öncesinde ve sonrasındaki durumunu karşılaştırmak amacıyla, yazılım süreç iyileştirme için metrik modeli önermişlerdir. Olszewska ve arkadaşları çalışmalarında [38], önerdikleri ölçüm modelinin mevcut geliştirme, teşvik edilen öz değerlendirme ve sürekli iyileştirmenin mevcut durumunu gösterdiğini öne sürmüşlerdir.

3.2.2. AS2: Hangi metrikler ile ölçüm yapılmıştır?

Bu araştırma sorusunu adreslemek için metrikler olgunluk seviyesi, hata ile ilişkili metrikler, çaba ve çalışma verimliliği metrikleri olarak sınıflandırılmıştır. Çizelge 3.3'de bu metrikler özetlenmektedir.

Çizelge 3.3: Metrik kategorileri

Metrik kategorisi	**Metrikler**
Olgunluk seviyesi	Takımın çevikliği [5], Uygulama seviyesi [52]
Hata ile ilişkili	Hata sayısı [8, 40, 42, 43, 48, 53-58], Harici sorun raporları sayısı [4, 37, 38, 41, 46], Hata/sorun raporları [59], Kümülatif açılmış ve kapatılmış hata sayısı [60], Kaynak kodu kontrol havuzunda değiştirilen, eklenen ya da silinen her bin kod satırındaki hata (İng. Bug) sayısı [61], Tasarım kusurlarının (İng. Design Flaw) sayısı [62], Sızıntı hatalarının (İng. Leaking-defects) oranı [63], Hata yoğunluğu [64, 65], Hataların/kusurların önem derecesi [66, 67]
Çaba ve çalışma verimliliği	İş bitim grafiği (İng. Burndown) [3, 54, 68, 69], Çevrim süresi (İng. Cycle Time) ve akış zamanı (İng. Lead Time) [37, 38, 41], İş değeri / çaba [8, 38, 41], Ürün geliştirmeye harcanan zaman karşılaştırması / eski ve çevik yolla kusurların giderilmesi [8, 70], Çaba sapması (İng. Effort Deviation) [40],

	Yükü test etme ve yeniden işleme (İng. Testing overhead and rework) [63], Test çabası [66, 71-73], Planlama çabası [48, 74], Toplam iş çabası [65] Üretkenlik [40, 54]

Birincil çalışmalardan iki tanesi [5, 52] olgunluk seviyesi kategorisine dâhil edilmiştir. İlk çalışmada [5], inovasyon artışı ve pazarlama zamanı azaltılması gibi iş hedeflerine ulaşılmasını sağlamak için uygulama seviyesi ölçülmüştür. İkinci çalışmada [52] ise her bir takımın çevikliğini ölçmek için metrikler kullanılmıştır.

Hata ile ilişkili ölçümlerin 26 çalışmada [4, 8, 37, 38, 40-43, 46, 48, 53-67] yer aldığı gözlemlenmiştir. Örneğin [53], [40], and [54] no'lu çalışmalar hata sayısı ölçümü ile yazılım kalitesini değerlendirmişlerdir. Başka bir çalışmada [60], yazılım kuruluşunda haftada üç kez teslim edilen, kümülatif açılan ve kapatılan hataların sayıları, hata ile ilişkili metrik olarak toplanmıştır.

Son olarak 20 çalışma [3, 8, 18, 37, 38, 40, 41, 48, 54, 63, 65, 66, 68-74] çaba ve çalışma verimliliği kategorisiyle ilişkili bulunmuştur. Örneğin, bu kategori içindeki dört çalışma [3, 54, 68, 69], projelerin izlenmesini ve denetlenmesini sağlayan ve verimlilik hesaplamasını kolaylaştıran iş birim grafiklerini (İng. Burndown chart) kullanmışlardır. Bir başka çalışmada [40] ise çaba sapması (İng. Effort deviation), tahmini çaba ve gerçek çaba arasındaki yüzde farkı hesaplanarak ölçülmüştür.

3.2.3. AS3: Hangi varlıklar ölçülmüştür?

Çizelge 3.4'de gösterildiği gibi birçok birincil çalışma, çevik dönüşümü metriklerle ölçmüştür. Araştırmacıların ve uygulayıcıların gelecekte bu alanda yapacakları çalışmalara öngörü sağlaması açısından, dönüşümün birçok yönden ölçümünün yapılması faydalı olacaktır. Ürün metrikleri, diğer metrik kategorilerine kıyasla daha yüksek miktarda (39 çalışmada) kullanılmıştır. Süreç metrikleri, ikinci derecede en çok kullanım miktarına (32 çalışma ile) sahiptir. Kaynak metrikleri, 31 çalışmada kullanılmıştır. İş metrikleri, 28 çalışma ile en düşük kullanım miktarına sahip metrik kategorisidir. Bütün varlık kategorileri ve referansları Çizelge 3.4'de gösterilmektedir. Ürün ölçümüyle ilgili olarak Goodman ve Elbaz makalelerinde [53],

ürün kod kalitesi ölçümü için hata takibi, kod karmaşıklığı, test kapsamı metriklerini kullanıldıklarından ve çevik dönüşümün bu konuda iyileşmeye katkı sağladığından bahsetmişlerdir. Sureshchandra ve Shrinivasavadhani çalışmalarında [40], ürün ile ilgili olarak yazılımın hata sayısı bakımından kalitesini ve kaynakla ilgili olarak takımın genel üretkenliğini ölçmüşlerdir. Dört varlığı da adresleyen O'Connor'un çalışmasında [68] hız (İng. Velocity), kod kapsamı, çevrimsel karmaşıklık, iş bitim grafikleri ile ölçümler yapılmıştır. İş kategorisini adresleyen KeyCorp firmasında yapılan çalışmada [50] ise akış zamanı ölçülmüştür.

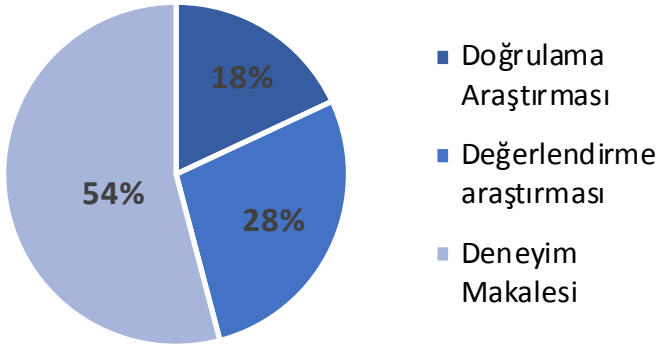
Çizelge 3.4: Varlık kategorileri ve çalışma sayıları

Varlık kategorileri	Çalışma sayısı	Çalışma referansları
İş metrikleri	28	[4, 5, 37-39, 41, 43-45, 50, 53, 54, 58, 59, 63, 67, 68, 70, 73-82]
Süreç metrikleri	32	[3-5, 8, 37, 38, 41, 43, 49, 51-53, 55, 56, 59, 60, 65, 66, 68, 70, 71, 77, 79, 80, 82-89]
Ürün metrikleri	39	[4, 8, 37, 38, 40-42, 44, 46, 48, 51, 53-69, 71-74, 76, 85-88, 90]
Kaynak metrikleri	31	[3, 5, 37, 38, 40-43, 46-48, 50-52, 54, 55, 59, 63, 65, 67, 68, 74, 77, 79-81, 83-85, 87, 89]

3.2.4. AS4: Dönüşümü bildiren çalışmalarda hangi araştırma yöntemleri kullanılmıştır?

Petersen ve arkadaşlarının yaptığı çalışmada [91], araştırma türleri hakkındaki tanımlamalara göre, makale havuzumuzdaki çalışmalar; doğrulama araştırması (İng. Validation Research), değerlendirme araştırması (İng. Evaluation Research) ve deneyim makalesi (İng. Experience Paper) olarak üç grupta sınıflandırılmıştır. Doğrulama araştırmasında incelenen teknikler yeni olup henüz pratikte (sektörde) uygulanmamıştır; ancak daha dar kapsamda (örn. laboratuvar ortamında) kullanılmıştır. Değerlendirme araştırmasında teknikler pratikte uygulanır ve tekniğin pratikte nasıl uygulandığı, uygulama sonucunda yararlarının ve dezavantajlarının neler olduğu gösterilerek değerlendirmesi yapılır. Deneyim makaleleri, pratikte neler yapıldığını ve nasıl yapıldığını açıklar; ayrıca yazarın kişisel tecrübesini anlatır. Bu sistematik literatür çalışması kapsamında hiçbir birincil çalışma çözüm önerisi (İng. Solution Proposal), felsefi makale (İng. Philosophical Paper) ya da görüş bildiri (İng. Opinion Paper) olarak sınıflandırılmamıştır.

Şekil 3.2'de görüldüğü üzere çalışmaların büyük çoğunluğu (33 çalışma [3-5, 39, 42-53, 56, 62, 63, 65, 66, 68, 70, 72-76, 78, 80-82, 89]) deneyim makalesi olarak sınıflandırılmıştır. Değerlendirme araştırması, ikinci en yüksek orana (17 çalışma [8, 37, 38, 41, 42, 55, 58, 67, 77, 79, 83, 84, 86-88, 92, 93] sahiptir. Örneğin [41, 83] no'lu çalışmalar araştırma sorusu ve geçerliliğe yönelik tehditler bölümlerini kapsadıkları için değerlendirme araştırması olarak sınıflandırılmıştır. 11 çalışma [40, 54, 57, 59-61, 64, 69, 71, 85, 90] doğrulama araştırması olarak sınıflandırılmıştır.



Şekil 3.2: Araştırma yöntemine göre çalışmaların dağılımı

3.2.5. AS5: Çalışmalar tarafından bildirilen dönüşümler olumlu mu olumsuz mu sonuçlanmıştır?

Çevik dönüşümün olumlu ve olumsuz sonuçları, 'yalnızca nitel', 'yalnızca nicel' ve 'nitel ve nicel' olarak kategorize edilmiştir. Çalışma kategorileri ve referansları Çizelge 3.5'de gösterilmektedir.

Sırasıyla 36, 50 ve 25 çalışma, dönüşümün 'nitel', 'nicel' ve 'nitel ve nicel' olmak üzere olumlu çıktılarını bildirmişlerdir. Örneğin, Goodman ve arkadaşlarının yaptıkları çalışmada [53] , çevik yöntemin iş-yaşam dengesini arttırdığı ve metrikleri kullanarak kod kalitesinde iyileşmeler görüldüğü bildirilmiştir. [38] no'lu çalışmada, çevik dönüşümün olumlu nicel bir sonucu olarak müşteri hizmetleri talebi için dönüş sürecinde, yaklaşık %24'lük bir azalma olduğu raporlanmıştır. Altı ve iki çalışma sırasıyla, nitel ve nicel olumsuz sonuçlar bildirmiştir. Makale havuzundaki birincil çalışmalardan hiçbiri, dönüşümün olumsuz sonuçları olarak 'nitel ve nicel' kategorisi

altında sonuç bildirmemiştir. Coffin çalışmasında [82], projenin toplam maliyetinin ilk beklentileri aştığını belirtmiştir. Başka bir çalışmada [38], harcanan para başına ürünün ortalama özellik (İng. Feature) sayısının, yeni çalışma yönteminde (çevik ve yalın geliştirme) eski çalışma yöntemine göre (plan güdümlü geliştirme) %483 daha yüksek olduğu bildirilmiştir.

Çizelge 3.5: Dönüşüm sonuç kategorileri

Sonuç Tipi	Yalnızca Nitel Sonuç Referansları	Yalnızca Nicel Sonuç Referansları	Nitel ve Nicel Sonuç Referansları	Çalışma sayısı
Olumlu	[39, 47, 51, 52, 57, 72, 74, 76, 78, 84, 93]	[5, 37, 38, 40-43, 45, 49, 54, 56, 58, 59, 64-66, 70, 75, 77, 82, 83, 85, 89, 90]	[3, 4, 8, 42, 44, 46, 48, 50, 53, 60-63, 67-69, 71, 73, 79-81, 86-88, 92]	61
Olumsuz	[8, 53, 67, 70, 71, 81]	[38, 82]	-	8

3.3. SLR'ın Geçerliliğine Yönelik Tehditler

SLR'ın geçerliliği açısından olası tehditler ile ilgili başlıca sorunlar; arama cümleleri nedeniyle makale havuzundaki olası eksik kaynaklar, araştırmacının dâhil etme/ dışlama kriterleri ve veri çıkarımına yönelik önyargılarıdır. Bahsi geçen ilk tehditle baş etmek için Bölüm 3.1.2'de açıklanan sistematik yaklaşım, kaynak arama ve seçiminde uygulanmıştır. Bu sistematik tarama çalışmasının tekrarlanabilir olduğunu doğrulamak için arama motoru, dijital kütüphaneler ve dâhil etme/dışlama kriterleri titizlikle tanımlanmış ve raporlanmıştır. Ayrıca makale havuzunun ilgili tüm birincil kaynakları içerdiğinden emin olmak için kartopu tekniği uygulanmıştır. İkinci tehditle baş etmek için dâhil etme/dışlama kriterleri, özet ve tam metin inceleme süreçlerinde temel alınmıştır. Dâhil etme/dışlama kriterleri uygulanırken ilgili herhangi bir kaynağı dışlama riskini azaltmak ve makale havuzunun kalitesini korumak için, sorulara verilen yanıtlar kriterlerle değerlendirilmiştir. Veri çıkarımı sürecinde, insan faktörü sebebiyle veriler istemeden yanlış yorumlanmış olabilir. Veri çıkarımının doğruluğunu sağlamak ve bu tehdidi en aza indirmek için seçilen kaynaklar üzerinde akran denetimi (İng. Peer Review) uygulanmıştır.

3.4. Bulguların ve Sonuçların Özeti

Bu sistematik literatür taramasının amacı, bilimsel literatürde bildirilen ve yazılım kuruluşlarında çevik dönüşümün etkilerini ölçen çalışmaların sentezini oluşturmaktı.

Bu sistematik tarama, ayrıntılı olarak incelediğimiz 61 çalışmanın seçimiyle ve analiziyle sonuçlanmıştır.

AS1'e göre katkı tipi bulguları, önerilen metrikler kıt olmasına rağmen, çalışmaların çoğunda bildirilmiştir. Bununla birlikte, dönüşümün etkilerini ölçmek için herhangi bir yöntem/teknik, araç ya da süreç önerilmemiştir. Sonuçlar, çevik dönüşümün etkilerini ölçmede bu kategorilerle ilgili katkıda bulunma fırsatı olduğunu göstermektedir. AS2'de yer alan metrik kategorileri içinde hata ile ilişkili metrikler; olgunluk seviyesi, çaba ve çalışma verimliliği metriklerine kıyasla en fazla çalışılan metriklerdir.

AS3 ile varlık ölçümüne bakıldığında, en çok ölçülenin yazılım ürünü olduğu gözlenmiştir. Ayrıca, diğer varlıkların da önemli miktarda ölçüldüğü fark edilmiştir. Birçok çalışmada dönüşümün etkilerinin çeşitli varlıklar açısından ölçülmesi tatmin edici bir durumdur. AS4 tarafından tanımlanan araştırma yöntemleri ile ilgili olarak çok sayıda çalışma, deneyim makalesi olarak sınıflandırılmıştır. Bununla birlikte daha az sayıda değerlendirme araştırmasının bulunması, araştırmacıların çevik dönüşümün etkilerini doğru ve yüksek bir güvenle gözlemlemesini zorlaştırmaktadır.

Son olarak AS5'e göre çevik dönüşümlerin olumlu ve olumsuz sonuçları, 'yalnızca nitel', 'yalnızca nicel' ve 'nitel ve nicel' olarak sınıflandırılarak araştırılmıştır. Çoğu çalışmada çevik dönüşüm sonuçlarının olumlu yönde raporlandığı bulunmuştur. Bununla birlikte olumsuz sonuçlar, daha az sıklıkla raporlanmış olabilir.

3.5. Vaka Çalışmasına Temel Olarak Alınan Metrikler

Çevik dönüşümün etkilerini ölçen çalışmaları kapsayan bu sistematik literatür taraması sonucunda, çalışılan firmanın bilgi ihtiyaçları göz önünde bulundurularak ölçüm tasarımına dahil edilen metrikler; çevrim ve bekleme süresi [37, 38, 41], akış zamanı [37, 38, 41], açılan ve kapatılan hataların durumu [60], hata yoğunluğu [64, 65], hatanın önem derecesi [66, 67] ve üretkenlik [40, 54] metrikleridir. Çevrimsel karmaşıklık, test işlem, test durumu etkililiği, hatanın çözülme zamanı ve kurallara uyum indeksi metrikleri, firmanın ölçüm için uygun gördüğü veriler baz alınarak bu metriklere eklenmiştir. Vaka çalışmasında kullanılan tüm metriklerin açıklamalarına, Bölüm 4.3'te yer verilmiştir.

4. VAKA ÇALIŞMASI TASARIMI

Bu tez çalışmasında yürütülen vaka çalışmasında keşif amaçlı araştırma (İng. Exploratory Study) [94] yöntemi uygulanmıştır. Sistematik tarama sonucunda elde edilen metrikler vaka çalışmasının araştırma sorularıyla ilişkilendirilerek ve firmanın bilgi ihtiyaçları göz önünde bulundurularak, ayrıca ISO 15939 Yazılım Ölçme Standardı [95] rehberliğinde ve firmanın işbirliği ile tekrarlamalı çalışarak, bir ölçüm tasarımı ortaya çıkarılmıştır. İzleyen alt bölümlerde vaka çalışmasının bağlamı ve kapsamı, hedefi ve araştırma soruları ve çalışmada kullanılacak ölçüm tasarımı açıklanmıştır.

4.1. Bağlam ve Kapsam

Vaka çalışması kapsamında müşterilerine kurumsal ürünler sunan bir yazılım firmasında, çevik dönüşüm sırasında temel bankacılık alanında gerçekleştirilmiş bir büyük ürünün farklı müşteriler için geliştirilmiş iki ayrı alt parçasından (İng. Sub-module) toplanan veriler ile geçişin etkisi değerlendirilmiştir. Değerlendirmeye tabi tutulan iki alt parça, işlevsellik ve büyüklük bakımından birbirine yakınlık göstermektedir. Bu iki alt parçadan izleyen paragraflarda ürün 1 ve ürün 2 olarak bahsedilecektir.

Ürün 1 için çevik dönüşüm öncesine ait bir sürüm ve çevik dönüşüm sonrasına ait dört sürüm, ürün 2 için çevik dönüşüm öncesine ait üç sürüm ve çevik dönüşüm sonrasına ait üç sürüm çalışmaya dâhil edilmiştir. Firma bünyesinde toplam 800 kişi çalışmaktadır. Bu çalışmada çevik dönüşüm etkilerinin ölçülmesi ve değerlendirilmesi kapsamında yalnızca 8'er kişiden oluşan 4 takımın ürettiği veriler kullanılmıştır. Firmada takımlara çevik dönüşüm öncesinde iki günlük temel seviyede çevik geliştirme ve Scrum eğitimleri verilmiştir. Eğitimler boyunca iki takıma birer Scrum Master eşlik etmiştir.

Firma, üretkenlik ve müşteriye dönüş hızı artışının sağlanabilmesi amacıyla, çevik dönüşüm gerçekleştirmiştir. Çevik dönüşüm esnasında dışarıdan herhangi bir danışmanlık hizmeti alınmadan, literatürdeki kaynaklardan yararlanılmıştır. Çevik dönüşüm ile birlikte her iki üründe de ölçekli Scrum (İng. Scaled Scrum) kullanılmaya başlanmıştır. Scrum geliştirme modeli içinde ürün vizyonunu ve iş listesini birlikte oluşturma, ürün iş listesini uygulama için efor tahmini, ürün iş

listesindeki ögeler arasındaki bağımlılıkları açıklama, müşteriyi Scrum etkinliklerine davet etme ve günlük toplantı etkinlikleri gerçekleştirilmiştir.

Firma içerisinde deneyselliği teşvik etmek amacıyla, uç programlama pratiklerinden eşli programlama ve mob programlama (İng. Mob Programming) denenmiş ve kullanılmaya devam edilmiştir. Mob programlama, tüm ekibin aynı şey üzerine aynı anda, aynı alanda ve aynı bilgisayarda çalıştığı bir yazılım geliştirme yaklaşımıdır. Bu yaklaşım, eşli programlama kavramını birlikte çalışan iki kişiden, her seferinde tek bir iş öğesi sunmak için sürekli olarak tek bir bilgisayarda işbirliği yapan tüm ekibe kadar genişletir. Kodlamaya ek olarak mob programlama takımları hikâyeler tanımlama, müşterilerle çalışma, yazılım tasarımı, test etme ve dağıtma da dâhil olmak üzere, tipik bir yazılım geliştirme takımının hemen hemen tüm çalışmalarında birlikte çalışırlar [96]. Ayrıca firmada test güdümlü geliştirme de denenmiş, ancak zorlanıldığı için devamlılık sağlanamamıştır.

Çevik dönüşüm öncesinde her iki üründe de çağlayan geliştirme modeli, Proje Yönetim Enstitüsü (İng. Project Management Institute-PMI) standartları rehber alınarak uygulanmıştır. Proje Yönetimi Enstitüsü (PMI), proje yönetiminin profesyonel alanını geliştiren uluslararası bir organizasyondur. Bunu, sertifikalı eğitim ve geliştirme yoluyla standartlar belirleyerek ve mesleki konferanslar yürüterek yapar. PMI, genel kabul görmüş proje yönetimi uygulamalarının yanı sıra, ortak bir proje yönetimi dili ya da sözlüğü geliştirmeye yönelik bir çaba göstererek Proje Yönetimi Bilgi Tabanı (İng. Project Management Body of Knowledge Guide-PMBOK Guide) yayımlamaktadır. Rehberin yanı sıra PMI projeler, programlar, kişiler, kuruluşlar ve meslek konularını ele alan 11 küresel standart belgesinin mevcut bir kütüphanesini de tutar.

4.2. Vaka Çalışmasının Hedefi ve Araştırma Soruları

Vaka çalışmasında, çevik dönüşüm gerçekleştirmiş bir firmada dönüşümün etkilerinin nicel olarak ölçülmesi ve değerlendirilmesi hedeflenmiştir. Bu hedefe ve SLR çalışmasıyla belirlenen metrik sınıflarına uygun olarak aşağıdaki araştırma soruları tanımlanmıştır:

AS1: Çevik dönüşüm yazılım kurumunda işi (İng. Business) nasıl etkilemiştir?

AS2: Çevik dönüşüm yazılım kurumunda süreci nasıl etkilemiştir?

AS3: Çevik dönüşüm yazılım kurumunda ürünü nasıl etkilemiştir?

AS4: Çevik dönüşüm yazılım kurumunda kaynağı nasıl etkilemiştir?

Ölçüm tasarımının, firmanın çevik dönüşüm öncesi ve sonrasındaki durumunun karşılaştırılmasında yararlı olmasını sağlamak amacıyla, kullanılacak metrikler aşağıdaki kriterler göz önünde bulundurularak seçilmiştir:

- Metrikler, hem plan güdümlü (şelale vb.) hem de çevik geliştirme yöntemine uygulanabilir olmalı.
- Metriklere ilişkin veriler, geçmişteki (çevik dönüşüm öncesindeki) ve devam etmekte olan projelerden toplanabilir olmalı.
- Metrikler objektif olarak değerlendirmeye olanak sağlamalı.

Ölçüm tasarımında kullanılacak metriklerin belirlenmesinden sonraki aşamada, firma tarafından ilgili veriler sağlanmış ve analiz edilerek sonuçları değerlendirilmiştir. Ölçüm tasarımı izleyen alt bölümde, analize ve sonuçlara ilişkin detaylar ise 5. Bölümde verilmiştir.

4.3. Ölçüm Tasarımı

Ölçüm tasarımı yapılandırılırken iş, süreç, ürün ve kaynak açılarından çevik dönüşümün etkilerini ölçmeyi sağlayacak bilgi göstergeleri (İng. Indicator) ve ilişkili ölçme yapıları (türetilmiş ve temel metrikler, ölçme fonksiyonları vb.) ISO 15939 Yazılım Ölçme Standardı rehberliğinde tanımlanmıştır.

Ölçüm yapısı, geleneksel yöntemlerle geliştirilmeye başlanılmış ve dönüşüm sonrasında çevik yöntemlerle gerçekleştirilmiş ürünler üzerinde uygulanması hedeflenerek oluşturulmuştur. İzleyen alt bölümlerde iş, süreç, ürün ve kaynak varlıkları bazında her bir ölçüm yapısı açıklanmış ve ilişkili gösterge tablosunda bulunan ‘Gösterge Yorumu’ kısmında, beklenen durumların ya da çıktıların tanımı verilmiştir.

4.3.1. İş Ölçümü Tasarımı

Çevik dönüşümü gerçekleştiren kurumların hedeflerinden biri de müşteriye dönüş hızını artırmaktır. Müşteriden gelen talebe karşılık verme esnekliğinin ölçülmesi, çevik dönüşümün işe yansıyan etkilerinin açıkça görülmesini mümkün kılar.

4.3.1.1. Çevrim Süresi ve Bekleme Süresi

Bu çalışmada çevrim süresi (İng. Cycle Time) müşteriden gelen talebin işleme konulduğu andan, tamamlamasına kadar geçen süre olarak ele alınmaktadır. Çevrim süresi, geliştirici bakış açısından değer taşıyan bir ölçüm olarak değerlendirilebilir. Bekleme süresi ise talebin işleme konulmadan önce işin yapılmaya hazır hale gelmesi için beklenen zaman dilimidir.

Bilgi İhtiyacı	Çevrim süresini ve bekleme süresini değerlendirmek
Analiz Birimi	Ürün bazında, talep başına
Ölçme Birimi	Zaman (gün)
Türetilmiş Metrik(ler)	1.Ortalama çevrim süresi 2.Ortalama bekleme süresi
Ölçme Fonksiyonu	1. Toplam çevrim süresi / çevrim sayısı 2. Toplam bekleme süresi / bekleme sayısı
Temel Metrik(ler)	Çevrim süresi, bekleme süresi
Gösterge Yorumu	Çevik yöntemlerle geliştirmede, çevrim ve bekleme süresinin kısa olması, ayrıca bekleme süresinin çevrim süresine oranının düşük olması beklenir.

4.3.1.2. Akış Zamanı

Müşteriden talebin geldiği andan, talebin işlenip ürünün müşteriye teslim edildiği âna kadar geçen zaman dilimi ‘akış zamanı’ (İng. Lead Time) olarak değerlendirilmiştir. Akış zamanı, talebin önceliklendirilmesi ve doğru ürüne yönlendirilmesi gibi süreçlerin toplam zamanı ile bekleme süresi ve çevrim süresi gibi metriklerin değerlerinin toplamı olarak hesaplanır. Akış zamanı, kullanıcı bakış açısından değer taşıyan bir ölçüm olarak değerlendirilebilir.

Bilgi İhtiyacı	Akış zamanını değerlendirmek
Analiz Birimi	Ürün bazında, talep başına
Ölçme Birimi	Zaman (gün)
Türetilmiş Metrik(ler)	Ortalama akış zamanı
Ölçme Fonksiyonu	Toplam akış zamanı / akış sayısı
Temel Metrik(ler)	Akış zamanı
Gösterge Yorumu	Çevik yöntemlerle geliştirilen üründe, akış zamanının kısa olması beklenir.

4.3.2. Süreç Ölçümü Tasarımı

4.3.2.1. Test İşlem (İng. Test Commit)

Fonksiyonel test, bir doğrulama aktivitesi olarak yazılımın tanımlı gereksinimleri sağlayıp sağlamadığının teyit edilmesi için yapılır. Bir geçerleme aktivitesi olarak yazılım test sürecinin son adımı olan kullanıcı kabul testinin (İng. User Acceptance

Testing-UAT) amacı, sistemin günlük iş ve kullanıcı senaryolarını destekleyip desteklemediğini değerlendirmek ve sistemin işletme kullanımı için yeterli ve doğru olmasını sağlamaktır. Kullanıcı kabul testi, geniş kapsamlı gereksinimleri karşılar; bu nedenle operasyonel ve fonksiyonel gereksinimler ile arayüz gereksinimlerini kapsamalıdır [97]. Operasyonel gereksinimler veri yakalama, veri işleme, veri dağıtımı ve veri arşivleme gereksinimlerini içerir. Fonksiyonel gereksinimler, tüm iş fonksiyonlarının iş kurallarına göre gerçekleştirilmesini sağlar. Arayüz gereksinimleri ise UAT'de yazılım sistemiyle bağlantılı tüm işletme sistemlerinin, gereksinim şartnamesinde tanımlanan şekilde veri geçişi ve alımına ya da kontrolüne ilişkin gereklerini içerir.

Bilgi İhtiyacı	Fonksiyonel ve kullanıcı kabul test işlemlerini değerlendirmek
Analiz Birimi	Ürün bazında
Ölçme Birimi	Test işlem
Türetilmiş Metrik(ler)	1.Toplam fonksiyonel test işlem sayısı 2.Toplam kullanıcı kabul test işlem sayısı
Ölçme Fonksiyonu	3.Toplam fonksiyonel test işlem sayısının, toplam kullanıcı kabul test işlem sayısına oranı 1.Fonksiyonel test işlem sayılarının toplanması 2.Kullanıcı kabul test işlem sayılarının toplanması 3.Toplam fonksiyonel test işlem sayısı / Toplam kullanıcı kabul test işlem sayısı
Temel Metrik(ler)	Fonksiyonel ve kullanıcı kabul testlerindeki işlem sayıları
Gösterge Yorumu	Çevik yöntemlerle geliştirilmiş üründe, toplam fonksiyonel test işlem sayısının toplam kullanıcı kabul test işlem sayısına oranının yüksek olması beklenir.

4.3.2.2. Test Durumu Etkililiği

Test durumu (İng. Test Case), test edilen bir sistemin gereksinimlerini yerine getirip getirmediğinin ya da doğru bir şekilde çalışıp çalışmadığının gösterilmesi için kullanılan girdiler, gerçekleşmesi beklenen adımlar ve beklenen sonuçlardan oluşan bir dizi koşul ya da değişkendir.

Bilgi İhtiyacı	Test durumu etkililiğini değerlendirmek
Analiz Birimi	Ürün bazında
Ölçme Birimi	Test durumu başına hata sayısı
Türetilmiş Metrik(ler)	Test durumu etkililiği
Ölçme Fonksiyonu	Toplam kaydedilen hata sayısı / Toplam test durum sayısı * 100
Temel Metrik(ler)	Kaydedilen hata sayısı, test durum sayısı
Gösterge Yorumu	Çevik yöntemlerin kullanımında az sayıda test durumu ile çok sayıda hata yakalanması, diğer bir deyişle test durumu etkililiğinin yüksek olması beklenir. Bununla

	birlikte dönüşüm sonrasında ürün hatalılığı düşmüşse tersi bir durum da gözlenebilir.

4.3.2.3. Hatanın Çözülme Zamanı

Müşteri tarafından bildirilen hataların ne kadar sürede çözüme kavuşturulduğu 'hata çözülme zamanı' olarak değerlendirilmiştir. Vaka çalışmasında yer alan firmada müşteri tarafından bildirilen hataların tamamı çözüldüğü için, 'hata çözme başarısı' metriği ölçüm tasarımında kullanılmamıştır.

Bilgi İhtiyacı	Bir hatanın ortalama çözülme süresini değerlendirmek
Analiz Birimi	Ürün bazında, hata başına
Ölçme Birimi	Hata başına gün sayısı (gün/hata)
Türetilmiş Metrik(ler)	Hatanın ortalama çözülme süresi
Ölçme Fonksiyonu	Hatanın çözülme süresi/Müşteriden dönen hata sayısı
Temel Metrik(ler)	Müşteriden dönen hata sayısı, hatanın çözülme süresi
Gösterge Yorumu	Çevik yöntemlerde hatanın daha kısa sürede çözülmesi, dolayısıyla hata başına gün sayısının düşük olması beklenir.

4.3.2.4. Açılan ve Kapatılan Hataların Durumu

Geliştirme süreci boyunca karşılaşılan sorunlar 'açılan hata', çözümlenen sorunlar 'kapatılan hata' [55] olarak ele alınmaktadır.

Bilgi İhtiyacı	Açılan ve kapatılan hataların durumunu değerlendirmek
Analiz Birimi	Ürün bazında, zamana göre
Ölçme Birimi	Hata
Türetilmiş Metrik(ler)	1.Ortalama açılan hata sayısı 2.Ortalama kapatılan hata sayısı 3.Hatanın kapatılma oranı
Ölçme Fonksiyonu	1. Toplam açılan hata sayısı/Ürün sayısı 2. Toplam kapatılan hata sayısı/Ürün sayısı 3. Toplam kapatılan hata sayısı/Toplam açılan hata sayısı
Temel Metrik(ler)	Açılan hata sayısı, kapatılan hata sayısı
Gösterge Yorumu	Çevik yöntemlerle geliştirmede kapatılan hata sayısının açılan hata sayısına daha yakın bir değerde olması (seyretmesi) beklenir.

4.3.3. Ürün Ölçümü Tasarımı

4.3.3.1. Hata Yoğunluğu

Hata yoğunluğu, ürünlerin kalite değerlendirmesinde kullanılan önemli metriklerden biridir. Hata yoğunluğunun yüksek olması, düşük kaliteli ürün olarak değerlendirilmektedir. Yazılım büyüklük ölçümü için çalıştığımız firmanın işlev puanı

(İng. Function Point) verilerine erişilememesi sebebiyle, hata sayısını normalize etmek için kod satır sayısı kullanılmıştır.

Bilgi İhtiyacı	Hata yoğunluğunu değerlendirmek
Analiz Birimi	Ürün bazında
Ölçme Birimi	Kod satırı başına hata (hata/kod satırı)
Türetilmiş Metrik(ler) **Ölçme Fonksiyonu**	Hata yoğunluğu Üründeki toplam hata sayısı / Ürünün kod satır sayısı
Temel Metrik(ler)	Hata sayısı, ürünün kod satır sayısı
Gösterge Yorumu	Çevik yöntemlerle geliştirmede hata yoğunluğunun düşmesi beklenir.

4.3.3.2. Hatanın Önem Derecesi

Hatalar, yazılım kalitesi üzerinde oluşturduğu olumsuz etkinin şiddetini vurgulamak için önem derecesine göre sınıflandırılır. Bu çalışmada hatalar düşük, orta, yüksek ve kritik olmak üzere dört sınıfa ayrılmıştır. Düşük önem derecesi, ürünün işlevselliğini ya da veriyi etkilemeyen, yazım denetimi/dilbilgisi hataları ya da tasarımdaki yüzeysel hataları temsil eder. Orta önem derecesindeki hatalar, ürünün birincil özelliklerini veya işlevselliğini engellemezken düşük dereceli hatalara göre daha büyük bir olumsuz etki oluşturur. Ürünün tanımlı gereksinimlerini karşılamasını veya bir özelliğini yerine getirmesini engelleyen sorunlar, yüksek önem dereceli hata olarak sınıflandırılır. Kritik önem derecesindeki hatalar, çözümlenmedikçe ürünün işlevselliğini yerine getiremediği, veri kaybına veya bozulmasına sebep olan hatalar olarak değerlendirilir.

Bilgi İhtiyacı	Hataların önem derecesini değerlendirmek
Analiz Birimi	Ürün bazında
Ölçme Birimi	Hata derecesi oranı
Türetilmiş Metrik(ler) **Ölçme Fonksiyonu**	Toplam hata sayısı, her hata derecesinin oranı Hata derecesine göre toplam hata sayısı/Tüm derecelerdeki hataların toplam sayısı
Temel Metrik(ler)	Hata derecesine göre hata sayısı
Gösterge Yorumu	Çevik yöntemlerle geliştirilen üründe, her bir alt hata derecesinin, bir üst hata derecesine göre daha düşük oranda hataya sahip olması beklenir. Örneğin, düşük hata derecesindeki hata oranının, diğer hata derecelerine göre en düşük oranda olması beklenir.

4.3.3.3. Kurallara Uyum İndeksi

Kurallara uyum indeksi (İng. Rules Compliance Index-RCI), teknik kalite değerlendirmesinde kullanılır. Sonar Qube aracı üzerinden verimlilik (İng. Efficiency), sürdürülebilirlik (İng. Maintainability), taşınabilirlik (İng. Portability),

güvenilirlik (İng. Reliability), kullanılabilirlik (İng. Usability) kategorileri için, belirlenen kural setine göre 'ihlal yoğunluğu' hesaplanır.

Bilgi İhtiyacı	Kurallara uyum indeksini değerlendirmek
Analiz Birimi	Ürün bazında
Ölçme Birimi	Ağırlıklandırılmış ihlal sayısı, kod satır sayısı
Türetilmiş Metrik(ler) **Ölçme Fonksiyonu**	Kurallara uyum indeksi 100 - (Ağırlıklandırılmış ihlallerin sayısı / Kod satır sayısı * 100)
Temel Metrik(ler)	Verimlilik, sürdürülebilirlik, taşınabilirlik, güvenilirlik ve kullanılabilirlik için ihlal sayıları
Gösterge Yorumu	Çevik yöntemlerle geliştirilmiş üründe, kurallara uyum indeksinin yüksek olması ve ihlal yoğunluğunun düşük olması beklenir.

4.3.3.4. Çevrimsel Karmaşıklık

McCabe'in çevrimsel karmaşıklığı (İng. Cyclomatic Complexity), bir yazılım programının karmaşıklığını nicelleştiren bir yazılım yapısı metriğidir [98]. Çevrimsel karmaşıklık, programa ilişkin kontrol akış diyagramını kullanarak doğrusal bağımsız yolların sayısının hesaplanmasıyla çıkarılır. Çevrimsel karmaşıklık değeri ne kadar yüksekse program o kadar karmaşık ve hata eğilimli olarak değerlendirilir. Çevrimsel karmaşıklık değerinin yüksek olması, programların bakım yapılabilirliğini ve test edilebilirliğini zorlaştırması sebebiyle istenmeyen bir durumdur.

Çevrimsel karmaşıklık, yazılım mühendislerinin bir programa özgü risklerini belirlemesine yardımcı olabilir. Birçok çalışma, çevrimsel karmaşıklık ile programın hataya yatkınlığı arasında anlamlı bir ilişki olduğuna dair deneysel kanıt sağlamıştır [99, 100]. Çevrimsel karmaşıklık, test edilebilirliğin de bir göstergesi olarak kabul edilmektedir [101, 102]. Bir programın çevrimsel karmaşıklığı ne kadar yüksekse test etme çabası da o kadar yüksek olur.

Bilgi İhtiyacı	Ürünün karmaşıklığını değerlendirmek
Analiz Birimi	Ürün
Ölçme Birimi	Kontrol akış diyagramı
Türetilmiş Metrik(ler) **Ölçme Fonksiyonu**	Çevrimsel karmaşıklık Kenar sayısı - Düğüm sayısı + 2 * Programdaki bileşen sayısı
Temel Metrik(ler)	Kenar sayısı, düğüm sayısı, programdaki bileşen sayısı
Gösterge Yorumu	Çevik yöntemlerle geliştirilmiş üründe, çevrimsel karmaşıklığın düşük olması beklenir [103].

4.3.4. Kaynak Ölçümü Tasarımı

4.3.4.1. Üretkenlik

Üretkenlik, takımın performans değerlendirmesinde kullanılan göstergelerden biridir [40]. Takımın üretkenliğin yüksek olması, en az kaynak ve çaba sonuncunda en fazla çıktının elde edilmesi olarak değerlendirilir.

Bilgi İhtiyacı	Takımın üretkenliğini değerlendirmek
Analiz Birimi	Takım bazında
Ölçme Birimi	Kişi-ay başına kod satır sayısı (kod satırı/kişi-ay)
Türetilmiş Metrik(ler) **Ölçme Fonksiyonu**	Ortalama üretkenlik Kod uzunluğu / Takımın iş gücü
Temel Metrik(ler)	Kod satır sayısı, takımın iş gücü
İndikatör Yorumu	Çevik geliştirme yöntemleriyle çalışan takımların üretkenliğinin yüksek olması beklenir.

5. VAKA ÇALIŞMASININ SONUÇLARI VE TARTIŞMA

Ölçüm tasarımında belirlenen metrikler, firmada çevik dönüşüm gerçekleştirmiş 2 üründen toplanan veriler üzerinde uygulanmıştır. Ölçümlerden elde edilen bilgi göstergeleri ve sonuçları, kategorilerine göre aşağıdaki bölümlerde sunulmuştur.

5.1. İş Ölçümü Sonuçları

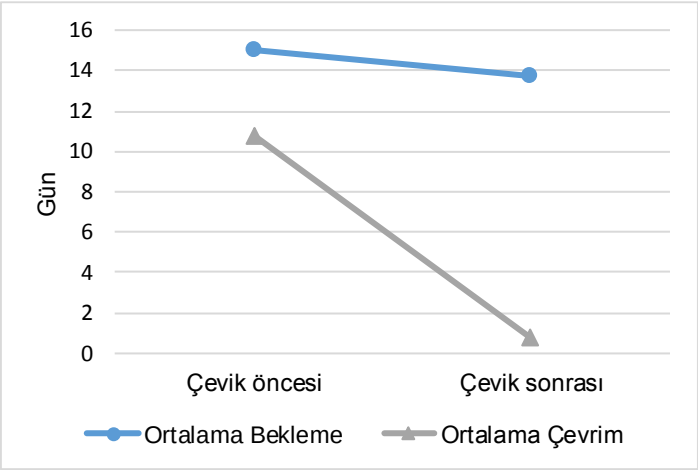
5.1.1. Çevrim Süresi ve Bekleme Süresi

Çevik dönüşüm öncesinde ve sonrasında elde edilen bekleme ve çevrim süresi değerlerinde yaşanan değişiklikler, aşağıda Çizelge 5.1'de gösterilmiştir. Bu metriğe ilişkin değerler iki ürünün çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.

Çizelge 5.1: Bekleme ve çevrim süresi değerleri (gün bazında)

Geliştirme yöntemi / Metrik	Çevik öncesi	Çevik sonrası	Geliştirme yöntemi / Metrik	Çevik öncesi	Çevik sonrası
Toplam bekleme süresi	270	262	Toplam çevrim süresi	43	14
Toplam bekleme sayısı	18	19	Toplam çevrim sayısı	4	19
Ortalama bekleme süresi	15	13,8	Ortalama çevrim süresi	10,8	0,7

Çevik dönüşüm sonrasında ortalama bekleme ve çevrim sürelerinin her ikisinde de bir düşüş gözlemlenmiştir. Şekil 5.1'de ortalama bekleme süresinde hafif azalan bir eğim, ortalama çevrim süresinde ise daha sert azalan bir eğim görülmektedir. Ölçüm tasarımında bölüm 4.3.1.1'deki gösterge yorumunda belirtildiği gibi, çevik yöntemlerle geliştirmede, çevrim ve bekleme süresinin daha kısa olması öngörüsüne uygun veriler elde edilmiştir. Ne var ki ortalama bekleme süresinin ortalama çevrim süresine oranı, çevik dönüşüm öncesinde 1,4 iken çevik dönüşüm sonrasında 18,7'ye yükselmiştir. Buna göre ortalama bekleme süresinin, ortalama çevrim süresine oranının düşük olması beklentisi karşılanmamıştır. Bu durumun sebebi, ortalama çevrim süresinin çok azalmış olması, ortalama bekleme süresinin ise (muhtemelen diğer kurumsal yapı ve süreçler nedeniyle) çok azalmamış olması olabilir.



Şekil 5.1: Ortalama bekleme ve çevrim süresi

5.1.2. Akış Zamanı

Akış zamanına ilişkin değerler iki ürünün, çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.

Çizelge 5.2: Akış zamanı değerleri (gün bazında)

Geliştirme yöntemi / Metrik	Çevik dönüşüm öncesi	Çevik dönüşüm sonrası
Toplam akış zamanı	394	351
Toplam akış sayısı	21	22
Ortalama akış zamanı	18,8	16,0

Bölüm 4.3.1.2'deki ölçüm tasarımında yer alan gösterge yorumuna paralel olarak, Çizelge 5.2'de toplam akış zamanı ve ortalama akış zamanı değerlerinde düşüş gözlenmektedir. Çevik dönüşüm sayesinde, müşteriden gelen talebin çağlayan geliştirme yöntemine göre daha hızlı bir şekilde işlendiği ve ürün haline dönüştürülüp teslim edildiği anlaşılmaktadır.

5.2. Süreç Ölçümü Sonuçları

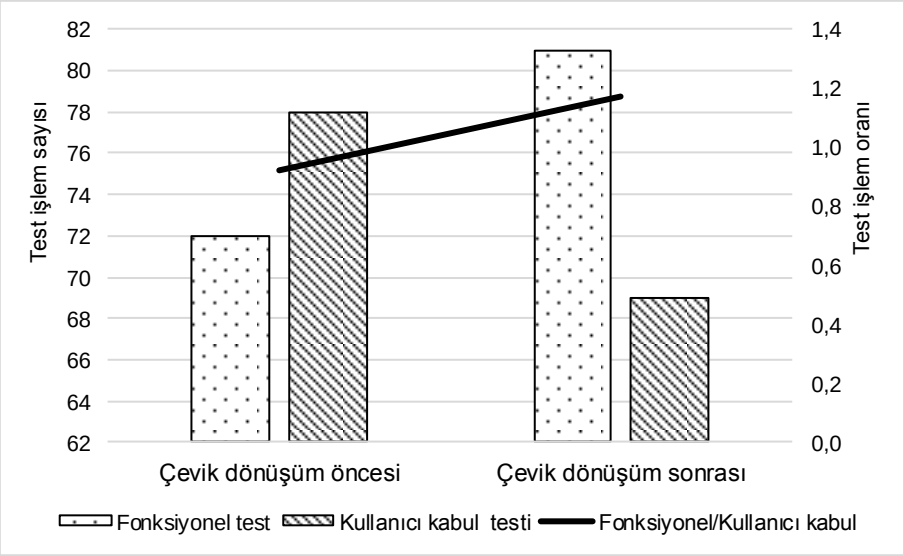
5.2.1. Test İşlem

Test işleme ilişkin değerler iki ürünün çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.

Çizelge 5.3: Fonksiyonel ve kullanıcı kabul test işlem sayıları

Geliştirme yöntemi / Metrik	Çevik dönüşüm öncesi	Çevik dönüşüm sonrası
Fonksiyonel test işlem sayısı	72	81
Kullanıcı kabul test işlem sayısı	78	69
Fonksiyonel/Kullanıcı kabul test işlem sayısı	0,92	1,17

Çizelge 5.3'de görüldüğü üzere, kullanıcı kabul test işlem sayısı çevik dönüşümle birlikte azalırken fonksiyonel test işlem sayısı artmaktadır. Ayrıca Şekil 5.2'de çevik dönüşüm gerçekleştirildikten sonra fonksiyonel test işlem sayısının, kullanıcı kabul test işlem sayısına oranında artış olduğu görülmektedir. Ölçüm tasarımında bölüm 4.3.2.1'deki gösterge yorumunda belirtildiği gibi çevik dönüşüm gerçekleştirildikten sonra bu oranın artış eğiliminde olması beklentisiyle örtüşen bir sonuç ortaya çıkmıştır.



Şekil 5.2: Fonksiyonel ve kullanıcı kabul test işlem grafiği

5.2.2. Test Durumu Etkililiği

Test durumu etkililiğine ilişkin değerler iki ürünün, çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır. Hatayla ilişkisi tutulan test durumları ölçüme dâhil edilmiştir.

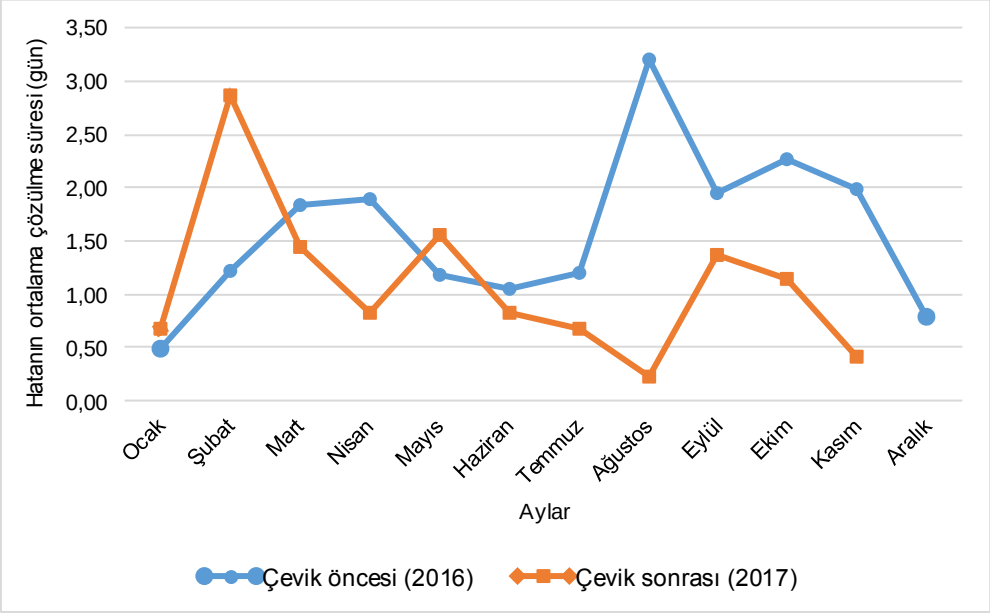
Çizelge 5.4: Test durumu etkililiği ile ilişkili değerler

Geliştirme yöntemi / Metrik	Çevik dönüşüm öncesi	Çevik dönüşüm sonrası
Toplam hata sayısı	411	259
Toplam test durumu sayısı	540	356
Test durumu etkililiği(Hata/Test durumu*100)	76%	73%

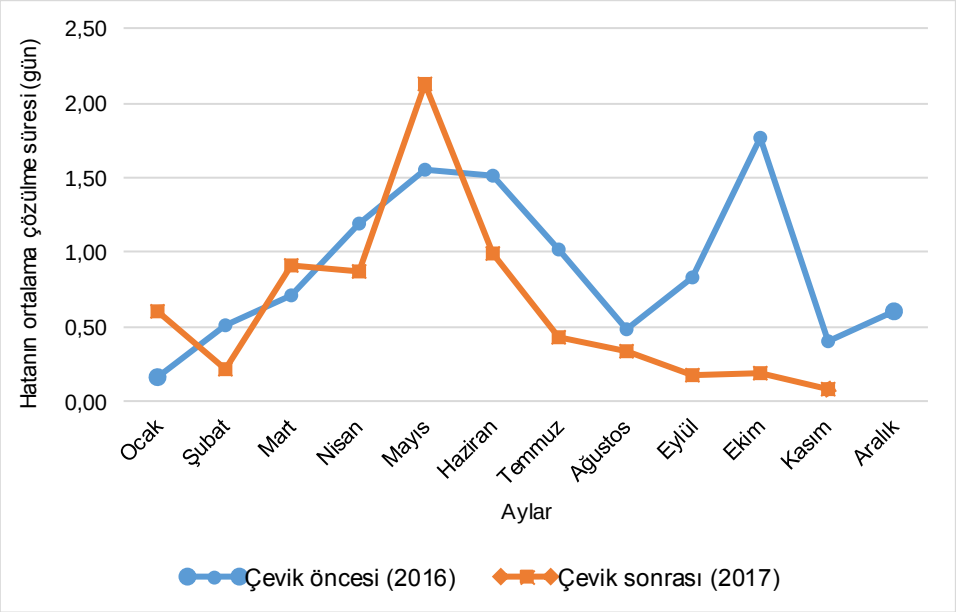
Ölçüm tasarımında, çevik dönüşüm sonrasında test durumu etkililiğinin artış göstermesine yönelik bir öngörü belirtilmişti. Oysa çevik dönüşüm ile test durumu etkililiği değerinde %3'lük bir azalma görülmektedir. Bu bulguya göre az sayıda test durumu ile çok sayıda hata yakalanması hedefine ulaşılamamıştır. Bununla birlikte, çevik dönüşüm sonrasında üründeki hata oranının azalması sebebiyle oransal olarak daha az hata yakalanmış olabilir.

5.2.3. Hatanın Çözülme Zamanı

Çevik dönüşüm öncesi ve sonrasında hatanın çözülme zamanına ilişkin veriler Ürün 1 ve Ürün 2 için ayrı ayrı hesaplanmıştır. Çevik öncesi ve sonrası için sırasıyla, 2016 yılına ait ve 2017 yılının Aralık ayına kadarki döneme ait veriler kullanılmıştır.



Şekil 5.3: Ürün 1 için hatanın ortalama çözülme süresi



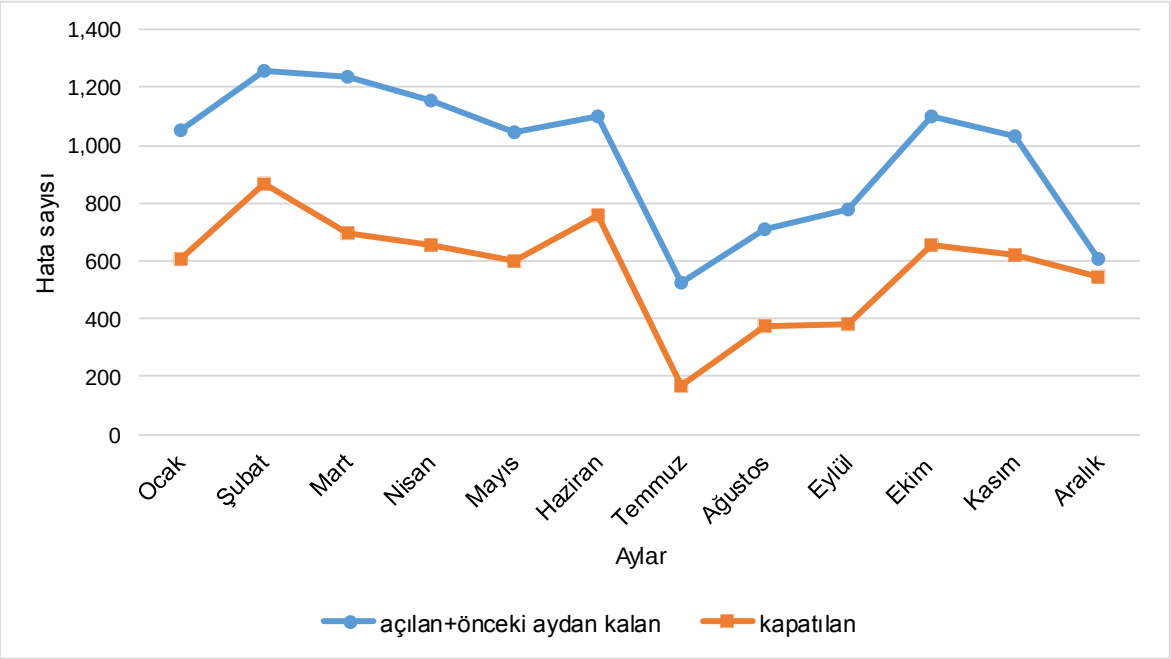
Şekil 5.4: Ürün 2 için hatanın ortalama çözülme süresi

Müşterinin bildirdiği hatalar Ürün 1 için çevik dönüşüm öncesinde ortalama 1,62 günde çözülürken çevik dönüşüm sonrasında ortalama 1,09 günde çözülmüştür. Ürün 2 için ise çevik dönüşüm öncesinde ortalama 0,89 günde ve çevik dönüşüm sonrasında ortalama 0,73 günde çözülmüştür. Her iki ürün için de çevik dönüşüm ile hataların ortalama çözülme süresi azalmıştır. Şekil 5.3 ve Şekil 5.4'de müşteri tarafından bildirilen hataların, aylara göre ortalama çözülme süresi verilmiştir. Şekil 5.3'deki grafikte, Ürün 1 için hataların ortalama çözülme zamanı çevik öncesinde aylar boyunca artan bir seyir izlerken çevik sonrasında azalan bir seyir izlemektedir. Şekil 5.4'deki grafikte, Ürün 2 için çevik dönüşüm öncesinde aylar boyunca artan bir seyir görülürken çevik dönüşüm sonrasında Mayıs ayına kadar artan, bu aydan sonra hızla azalan bir seyir mevcuttur. Bölüm 4.3.2.3'deki ölçüm tasarımında belirtildiği gibi çevik yöntemlerde hatanın kısa sürede çözülmesi beklentisine uygun sonuçlar elde edilmiştir.

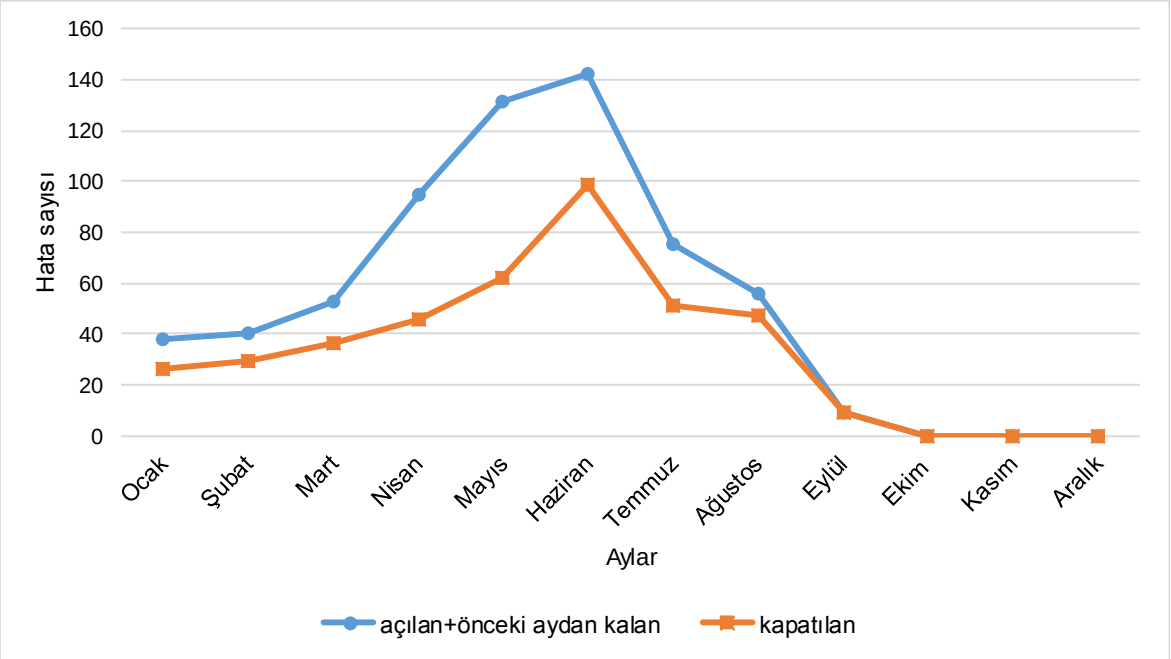
5.2.4. Açılan ve Kapatılan Hataların Durumu

Açılan ve kapatılan hataların durumu incelenirken iki ürünün toplam hata sayısı çevik dönüşüm öncesi 2016 yılına ait verilerden, çevik dönüşüm sonrası 2017 yılının ilk 9 ayına ait verilerden çıkarılmıştır. Genel olarak bakıldığında Şekil 5.6'daki çevik dönüşüm sonrası hata sayılarının, Şekil 5.5'deki çevik dönüşüm öncesindeki hata

sayılarından ciddi bir farkla az olmasının sebebi, çevik dönüşüm sonrasında bakım ve operasyonel işlerden kaynaklanan hataların 2017 yılına aktarılmamış olmasıdır.



Şekil 5.5: Çevik dönüşüm öncesi açılan ve kapatılan hataların dağılımı (2016)



Şekil 5.6: Çevik dönüşüm sonrası açılan ve kapatılan hataların dağılımı (2017 ilk 9 ay)

Çevik dönüşüm öncesinde, Haziran ayına kadar hatanın kapatılma oranı %50'nin üzerinde seyrederken Haziran-Temmuz arasında bir düşüş gözlenmiş, ardından hatanın açılma ve kapatılma sayılarında artış olmuştur (Şekil 5.5). Bu düşüşün sebebi firma genelinde çalışanların yaz tatiline çıkmış olmasıdır.

Çevik dönüşüm sonrasında, ilk 3 ay hatanın kapatılma oranı %60'ın üzerinde seyrederken Nisan ve Mayıs aylarında %50'nin altına düşmüş, ardından Haziran'dan itibaren %65'in üzerinde seyretmiştir (Şekil 5.6).

5.3. Ürün Ölçümü Sonuçları

5.3.1. Hata Yoğunluğu

Hata yoğunluğuna ilişkin değerler iki ürünün çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.

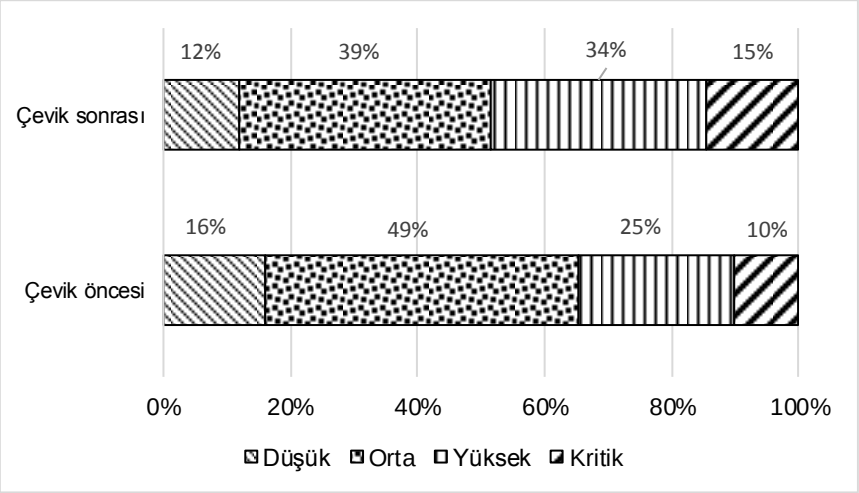
Çizelge 5.5: Hata yoğunluğu ile ilişkili değerler

Geliştirme yöntemi / **Metrik**	**Çevik dönüşüm öncesi**	**Çevik dönüşüm sonrası**
Toplam hata sayısı	13.978	2.048
Kod satır sayısı	1.825.312	2.000.720
Hata yoğunluğu	0,8%	0,1%

Çizelge 5.5'de çevik dönüşüm gerçekleştirildikten sonra hata yoğunluğu ‰8 değerinden ‰1 değerine düşmüştür. Öngörüldüğü şekilde çevik dönüşümün, hata yoğunluğu üzerinde olumlu etkisi gözlenmiştir. Hata yoğunluğunun düşmesi, test durumu etkililiğindeki %3'lük düşme ile birlikte değerlendirildiğinde geliştirme kodu bakımından, iki ürünün toplam kalitesinde iyileşme olduğu şeklinde yorumlanabilir.

5.3.2. Hatanın Önem Derecesi

Hatanın önem derecesine ilişkin değerler iki ürünün, çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.



Şekil 5.7: Hatanın önem derecesi

İki ürüne ait toplam hata sayısı sıralı ölçek kullanılarak düşük, orta, yüksek, kritik olarak sınıflandırılmıştır. Şekil 5.7'den çevik dönüşümle birlikte, düşük dereceli hata tipi oranında %4 ve orta dereceli hata tipi oranında %10'luk bir azalma gözlenmektedir. Yüksek dereceli hata oranı %9 ve kritik dereceli hata oranı %5'lik bir artış göstermiştir. Çevik geliştirmede düşük dereceli hataların oranı, yinelemeler boyunca daha kolay tespit edilebilmesiyle düşmüş olabilir. Genel olarak bakıldığında çevik dönüşüm, kritik hataların oranında önemli bir değişikliğe neden olmamıştır.

5.3.3. Kurallara Uyum İndeksi

Kurallara uyum indeksine ilişkin değerler iki ürünün, çevik dönüşüm öncesinde 2015 ve 2016 yıllarına ait ve çevik dönüşüm sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır. Bölüm 4.3.3.3'deki tabloda belirtilen verimlilik, sürdürülebilirlik, taşınabilirlik, güvenilirlik, kullanılabilirlik için ihlal sayıları gizlilik sebebiyle verilememiştir.

Sonar Qube aracından elde edilen verilere göre çevik dönüşüm öncesinde 2015 yılında kurallara uyum indeksi 82,9 değerinden %2,4 artarak 2016 yılında 84,9 değerine ulaşmıştır. Çevik dönüşüm gerçekleştirildikten sonra kurallara uyum indeksi 84,9 değerinden %4,6 artarak 2017 yılında 88,8 değerine yükselmiştir. Çevik dönüşüm ile kaynak kodunun teknik kalite bakımından iyileşme oranının arttığı anlaşılabilir.

5.3.4. Çevrimsel Karmaşıklık

Çevrimsel karmaşıklığa ilişkin değerleri iki ürünün çevik dönüşüm öncesinde 2016 yılına ait ve sonrasında 2017 yılının ilk 9 ayına ait verileri üzerinden hesaplanmıştır.

Çizelge 5.6: Çevrimsel karmaşıklık için referans değerleri [104, 105]

Çevrimsel karmaşıklık değeri	Anlamı
0-10	Risk taşımayan program
11-20	Orta derecede riskli program
21-50	Yüksek derecede riskli program
>50	Bakımı yapılamayacak kadar riskli program

Çevrimsel karmaşıklık değeri çevik dönüşüm ile birlikte 2,20'den %9,1 azalarak 2,0 değerine ulaşmıştır. Çevrimsel karmaşıklık için Çizelge 5.6'deki referans değerler göz önüne alındığında, her iki geliştirme yöntemi ile risk taşımayan programlar üretildiği görülmektedir.

5.4. Kaynak Ölçümü Sonuçları

5.4.1. Üretkenlik

Üretkenliğe ilişkin değerler dört takımın ve iki ürünün, çevik dönüşüm öncesinde 2015 ve 2016 yılına ait ve çevik dönüşüm sonrasında 2017 yılının ilk 9 ayına ait toplam değerleri üzerinden hesaplanmıştır. Bölüm 4.3.4.1'de belirtilen kod uzunluğu ve takımların iş gücü gizlilik sebebiyle verilememiştir.

Üretkenlik hesaplanırken ölçüm tasarımında belirtildiği üzere, ürün büyüklüğünün (kod satır sayısı) harcanan çabaya (kişi-ay) bölünmesi formülü uygulanmıştır. Üretkenlik metriğinin hesaplanması sonucunda yazılım geliştirme takımlarının üretkenliğinin 2016 yılında bir önceki yıla göre %12,3 azaldığı, çevik dönüşüm sonrası 2017 yılında ise bir önceki yıla göre %46,1 oranında arttığı görülmüştür. 2015 yılına göre 2016 yılındaki üretkenliğin azalmasına, firmada kullanılan teknoloji değişikliği sebep olmuştur. Bununla birlikte çevik dönüşüm sonrası 2017 yılındaki üretkenliğin, 2015 yılına kıyasla %28,2 oranında arttığı görülmüştür.

5.5. Bulguların Özeti ve Tartışma

Çevik dönüşüm sırasında gerçekleştirilmiş iki ürüne ait veriler üzerinde yapılan ölçümler sonucunda, bazı metrikler için ölçüm tasarımındaki gösterge yorumlarında beklenen çıktılar ile karşılaşılırken bazıları için beklenen durumdan farklı bulgular elde edilmiştir. Elde edilen bulgular, vaka çalışmasının bağlam ve kapsamı çerçevesinde değerlendirilmiştir.

İş metrikleri için aşağıdaki bulgular elde edilmiştir:

- Çevrim ve bekleme sürelerinin ölçümü sonucunda ortalama çevrim ve bekleme sürelerinin her ikisinde de bir azalış olmuştur. Çevik dönüşüm ile birlikte ortalama bekleme süresi %8,1 azalırken ortalama çevrim süresi %93,1 azalış göstermiştir. Ortalama bekleme süresinin, ortalama çevrim süresine oranı ise %12,4 artış göstermiştir. Her iki sürenin ortalama değerlerinin azalış göstermesi, çevik dönüşümün işi olumlu yönde etkilediğini ifade eder. Ancak ortalama bekleme süresindeki azalışın ortalama çevrim süresine göre daha az oranda gerçekleşmesi, ürünün geliştirmeye alınmasından önceki ön analiz sürecinde daha uzun kaldığı ve dönüşümün bu süreyi kısaltmada çok etkili olmadığı anlamına gelmektedir.
- Ortalama akış zamanı çevik dönüşüm ile birlikte %15 azalmıştır. Bu azalış ile çevik yöntemle geliştirilen ürünlerin daha kısa sürede müşteriye teslim edildiği anlaşılmaktadır.

Süreç metrikleri için aşağıdaki bulgular elde edilmiştir:

- Test işlem metriği için fonksiyonel test işlem sayısı %12,5 artış gösterirken kullanıcı kabul test işlem sayısı %11,5 azalmıştır. Fonksiyonel test işlem sayısının kullanıcı kabul test işlem sayısına oranı %27,2 artmıştır. Ölçüm tasarımında belirtildiği gibi, fonksiyonel test işlem sayısının kullanıcı kabul test işlem sayısına göre daha fazla olması yönündeki yorumla örtüşen bir çıktı elde edilmiştir. Bu bulgu, çevik yöntemin firmada fonksiyonel test işlemlerini güçlendirdiğini ve kullanıcı kabul test işlemlerini azalttığını göstermektedir.
- Test durumu etkililiği metriği için çevik dönüşüm gerçekleştirildikten sonra %3'lük bir azalma görülmüştür. Ölçümün yapıldığı iki ürün için çevik yöntemin

test durumu etkililiğinde negatif yönde bir etki gösterdiği anlaşılmaktadır. Bununla birlikte bu azalma, çevik dönüşüm sonrası ilk dokuz aya ait verilerde ölçüm yapılması ve yazılım kurumunda ilk aylarda öğrenmeye bağlı üretim düşüşü olmasına, dolayısıyla daha az sayıda test durumu üretilmesine bağlanabilir. Ayrıca az sayıda test durumu ile çok sayıda hata yakalanamamış olması çevik dönüşüm sonrası hata yoğunluğundaki düşüş ile birlikte değerlendirilmelidir. Bu durumun, çevik dönüşüm öncesine kıyasla çevik dönüşüm sonrası geliştirmede, daha düşük hata yoğunluğuna sahip ürün çıkarılmasından kaynaklandığı değerlendirilmektedir.

- Müşteri tarafından bildirilen hataların çözülme zamanı, her iki üründe çevik dönüşüm ile birlikte azalan bir seyir göstermiştir. Çevik dönüşümden sonra hatanın ortalama çözülme süresi, gün bazında Ürün 1 için %32,8 azalırken Ürün 2 için %18,2 azalmıştır. Bu ölçüm sonucuna göre, müşterinin raporladığı hataların daha kısa sürede çözülmesi, müşteriyle olan ilişkilere olumlu yönde etki edeceği ve böylece müşteri memnuniyetinin artacağı şeklinde yorumlanabilir.
- Açılan ve kapatılan hataların durumuna genel olarak bakıldığında iki ürüne ait ortalama açılan hataların, çevik dönüşüm öncesinde %99,2'sinin kapatıldığı ve çevik dönüşüm sonrasında %100'ünün kapatıldığı gözlenmiştir. Hatanın kapatılma oranında çevik dönüşüm öncesi ve sonrasında ciddi bir fark olmadığı görülmektedir.

Ürün kategorisindeki metrikler için şu bulgular elde edilmiştir:

- Hata yoğunluğu metriğine bakıldığında, ürünlerin çevik dönüşümden olumlu yönde etkilendiği görülmektedir. Çevik geliştirme yöntemi kullanılan ürünlerde hata yoğunluğu ‰8'den ‰1'e düşmüştür. Sadece hata yoğunluğunun düşmesine bakılarak müşteriye daha kaliteli ürün teslim edildiği düşünülebilir.
- Hataların önem derecelerinin oransal dağılımı da ürün kalitesinin değerlendirilmesi bakımından önemlidir. Hatalar düşük, orta, yüksek, kritik olarak derecelendirilmiş; çevik dönüşüm öncesi ve sonrasındaki değerleri ölçülmüştür. Bu metrik için hata derecelerine en üst düzeyden başlayarak bakıldığında, kritik dereceli hata oranında %5'lik ve yüksek dereceli hata

oranında ise %9'lük bir artış olduğu görülmüştür. Orta dereceli hata oranında %10'lük azalış görülürken düşük dereceli hata oranında %4'lük bir azalma olmuştur. Bu bulgulardan hareketle çevik yöntemleri kullanmanın, köklü sorunları ortaya çıkarmada çok büyük bir etkisi olmadığı düşünülebilir.

- Sonar aracının yazılım kaynak kodunu kalite profil ayarlarında belirlenen kurallara göre ayrıştırmasıyla kurallara uyum indeksi değeri elde edilmiştir. Kurallara uyum indeksi metriği için önceden tanımlanan kuralların ihlal oranı çevik dönüşüm öncesi 2016 yılında 2015 yılına kıyasla %7,1 azalarak uyum indeksi 82,9'dan 84,9'a yükselmiştir. İhlal oranı çevik dönüşüm sonrası 2017 yılında, çevik dönüşüm öncesi 2016 yılına kıyasla %18,7 azalarak uyum indeksi %88,8'e ulaşmıştır. 2015-2017 yıllarına göre, tanımlanan kuralların ihlal oranının azalış ve kurallara uyum indeksinin artış yüzdeleri, takımın öğrenme eğrisi ve çevik dönüşüm ile ürün kalitesinde meydana gelen iyileşmeye bağlanabilir.
- Çevrimsel karmaşıklık değerinin her iki geliştirme yönteminde de düşük olduğu görülmüştür. Ancak çevik dönüşüm sonrasında karmaşıklık değeri biraz daha azalmıştır.

Kaynak kategorisinde yer alan üretkenlik metriği ölçümü sonucunda, yazılım geliştirme takımının üretkenliği, çevik dönüşümden olumlu yönde etkilenmiştir. Çevik dönüşüm ile üretkenlik artış oranında iyileşme görülmüştür. Böylece firmanın çevik dönüşüm gerçekleştirme amaçlarından biri olan üretkenliği artırmanın sağlanmış olduğu tespit edilmiştir.

5.6. Vaka Çalışmasına Yönelik Geçerlilik Tehditleri

Bu bölümde vaka çalışmasının geçerliliği Yin'in [106] kitabında belirttiği dört kritik koşul ile değerlendirilmiştir. Bu koşullar yapı geçerliliği (İng. Construct Validity), iç geçerlilik (İng. Internal Validity), dış geçerlilik (İng. External Validity) ve güvenilirliktir (İng. Reliability).

Yapı geçerliliği, incelenen kavramlar için doğru operasyonel önlemlerin belirlenmesidir [106]. Bu tez çalışmasında, Hedef-Soru-Metrik çatısı ve ISO 15939 yazılım ölçüm süreci kullanılarak metriklerin, çalışmanın amacına uygun şekilde ölçüm tasarımında yer alması sağlanmıştır. Yapılan sistematik literatür taraması ile

Bölüm 3.5'de verilen kaynaklardan yararlanılarak vaka çalışmasında kullanılan çevrim ve bekleme süresi, akış zamanı, açılan ve kapatılan hataların durumu, hata yoğunluğu, hatanın önem derecesi ve üretkenlik metrikleri elde edilmiştir. Çevrimsel karmaşıklık, test işlem, test durumu etkililiği, hatanın çözülme zamanı ve kurallara uyum indeksi metrikleri ise yazılım kurumunun bilgi ihtiyaçlarına yönelik olarak mevcut verilerin değerlendirilmesiyle ölçüm tasarımına eklenmiştir.

Ölçüm tasarımında kullanılan metriklerin hesaplanmasında, vaka çalışmasının gerçekleştirildiği yazılım kurumunun ölçümde kullanılacak verilerinin mevcudiyetiyle ilgili olarak kısıtlamalarla karşılaşılmıştır. Örneğin yazılım büyüklük ölçümü için yazılım kurumunda işlev puanı verilerine erişilememesi sebebiyle, hata yoğunluğu ve üretkenlik metriklerinin hesaplanmasında kod satır sayısı metriği kullanılmıştır. Ayrıca çevik dönüşüm öncesi ve sonrası ölçüm sonuçlarının değerlendirilmesinde, kurallara uyum indeksi ve üretkenlik metrikleri için 2015 yılı verilerine erişilerek dönüşümün ürün ve kaynak üzerindeki etkileri daha kapsamlı incelenmiştir.

İç geçerlilik yalnızca açıklayıcı (İng. Explanatory Research) ya da nedensel çalışmalar (İng. Casual Study) için uygulanmaktadır [106]. Vaka çalışması kapsamında yalnızca nicel verilere odaklanılmış olup firmada çalışan takımların öğrenme eğrisi (İng. Learning Curve), takım uyumu (İng. Team Cohesion) vb. insan faktörünü içeren etmenler kapsam dışında tutulmuştur. Özellikle hatanın çözülme zamanı, açılan ve kapatılan hataların durumu ve hatanın önem derecesi metrikleri için zamana bağlı olarak geliştiricilerin alan bilgisinde meydana gelen iyileşmeler, çevik dönüşümün etkisinin değerlendirilmesine dâhil edilmemiştir. Bununla birlikte çalışma kapsamına, karşılaştırma bağlamları bakımından benzer işlevsellik ve büyüklükteki ürünler ile yetkinliği benzer takımlar dâhil edilmiştir. Ayrıca yürütülen vaka çalışması keşif amaçlı araştırma kapsamında gerçekleştirilmiştir. Bu nedenle ölçüm sonuçlarının değerlendirilmesinde çevik dönüşüm öncesi ve sonrası karşılaştırmaya yoğunlaşılmış olup nedensellik analizi (İng. Causality Analysis) yapılmamıştır.

Dış geçerlilik, bir çalışmanın bulgularının genelleştirilebileceği alanı tanımlamakla ilgilidir [106]. Yürütülen vaka çalışması yalnızca bir yazılım kurumunda gerçekleştirilmiştir. Bu nedenle çevik dönüşümün yorumlanmasında, sonuçların ne

ölçüde genelleştirilebileceği çalışmanın dış geçerliliğine yönelik bir tehdit olarak görülebilir. Farklı kurumlarda yapılacak benzer çalışmalar ile bu kısıt giderilebilir. Ayrıca bu çalışmanın firma çevik geliştirmede olgunlaştıktan sonra tekrar yapılması, çevik dönüşümün uzun vadede firmayı nasıl etkileyeceğini değerlendirmek açısından önemlidir.

Son olarak güvenilirlik, bir çalışmanın adımlarının aynı sonuçlarla tekrarlanabileceğini göstermekle ilgilidir [106]. Bölüm 4'de vaka çalışmasının gerçekleştirilmesinde izlenen adımların sistematik olarak uygulanması ve ölçüm tasarımı ile çalışmanın tekrar edilebilirliği güvence edilmiştir.

6. SONUÇ

Çevik dönüşüm gerçekleştirmenin etkilerinin ölçülmesi ve değerlendirilmesi, hem bu dönüşümü gerçekleştiren hem de gerçekleştirmeyi planlayan firmalar ve araştırmacılar açısından önemli bir yere sahiptir. Bu alanda yapılan deneysel çalışmalar, çevik dönüşümün etkilerinin anlaşılabilmesine ve değerlendirilebilmesine olanak tanır.

Bu tez çalışmasında bir yazılım kurumunda çevik dönüşüm sırasında gerçekleştirilmiş iki üründen toplanan veriler kullanılarak çevik dönüşümün etkilerinin ölçümü ve değerlendirmesi gerçekleştirilmiştir. Ölçüm tasarımının oluşturulmasından önce, sistematik literatür tarama yürütülmüştür. Literatürdeki konuyla ilgili mevcut çalışmaların incelenmesiyle ölçmeye temel olacak tanımlar belirlenmiş, ardından çalışmalardan çıkartılan metrikler ölçtükleri iş, süreç, ürün ve kaynak varlıklarına göre gruplanmıştır. Yazılım kurumunda bu varlıklar açısından çevik dönüşümün etkilerini ölçmeyi sağlayacak bilgi ihtiyaçları, ISO 15939 Yazılım Ölçme Standardı rehber alınarak belirlenmiş ve literatürden derlenen metriklerle ilişkilendirilmiştir. İş, süreç, ürün ve kaynak açılarından çevik dönüşümün etkilerini ölçmeyi sağlayacak bilgi göstergeleri ve ilişkili ölçme yapıları (türetilmiş ve temel metrikler, ölçme fonksiyonları vb.) tanımlanmıştır. Bu bilgi ihtiyaçları üzerinden, orta ölçekli bir yazılım kurumunda çevik dönüşümün etkileri ölçülmüş ve ölçüm sonuçları ilişkili olduğu bağlam ve kapsamda değerlendirilmiştir.

Vaka çalışması sonucunda, çevik dönüşüm yazılım kurumunda, birinci araştırma sorusu (AS1) kapsamında işi; çevrim ve bekleme süresi ve akış zamanı metrikleri bakımından ve ikinci araştırma sorusu (AS2) kapsamında süreci; test işlem ve test durumu etkililiği, hatanın çözülme zamanı, açılan ve kapatılan hataların durumu metrikleri bakımından olumlu yönde etkilediği saptanmıştır. Üçüncü araştırma sorusu (AS3) kapsamında çevik dönüşüm ürünü; hata yoğunluğu, hatanın önem derecesi, kurallara uyum indeksi, çevrimsel karmaşıklık metrikleri bakımından ve son olarak, dördüncü araştırma sorusu (AS4) kapsamında çevik dönüşüm kaynağı üretkenlik metriği bakımından olumlu yönde etkilemiştir. Çevik dönüşüm sayesinde firmanın amaçladığı şekilde, üretkenlik ve hız artışı sağlandığı görülmüştür. Ölçüm sonuçlarına genel olarak bakıldığında çevik dönüşümün kurumu olumlu yönde

etkilediği tespit edilmiştir. Bununla birlikte geliştiricilerin alan bilgisinde meydana gelen iyileşmeler, takımların öğrenme eğrisi, takım uyumu vb. faktörler de vaka çalışmasının sonuçları üzerinde etkili olmuş olabilir; dolayısıyla elde edilen sonuçlar bu çerçevede değerlendirilmelidir.

Gelecek çalışmalarda çevik dönüşümün etkilerini daha geniş kapsamda değerlendirebilmek amacıyla farklı kurumlarda ölçüm yapılması hedeflenmektedir. Ayrıca sistematik taramadan elde edilen, ancak bu tez çalışmasında kullanılmayan diğer metriklerin kullanılması da çevik dönüşümün etkilerini farklı bakış açılarından değerlendirmeye olanak sağlayacaktır.

KAYNAKLAR

[1] *Agile Transformation: Understanding What it Means to be Agile*. URL: http://www.castsoftware.com/research-labs/agile-transformation-what-is-it-definition [Son ziyaret tarihi:25 Ekim 2017]

[2] V. One, "11th Annual State of Agile Report," **2017**.

[3] G. Cloke, "Get your agile freak on! agile adoption at yahoo! music," in *Agile Conference (AGILE),* pp. 240-248, **2007**.

[4] B. Schatz and I. Abdelshafi, "Primavera gets agile: a successful transition to agile development," *IEEE software,* vol. 22, pp. 36-42, **2005**.

[5] J. Prochazka, M. Kokott, M. Chmelar, and J. Krchnak, "Keeping the Spin--From Idea to Cash in 6 Weeks: Success Story of Agile/Lean Transformation," in *Global Software Engineering (ICGSE), 6th IEEE International Conference*, pp. 124-130, **2011**.

[6] A. Tarhan and S. G. Yilmaz, "Systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process," *Information and Software Technology,* vol. 56, pp. 477-494, **2014**.

[7] N. Ozkan, A. Tarhan, and C. Kucuk, "Scrum at Scale in a COBIT Compliant Environment: The Case of Turkiye Finans IT," **2017**.

[8] J. Li, N. B. Moe, and T. Dybå, "Transition from a plan-driven process to scrum: a longitudinal case study on software quality," in *Proceedings of the ACM-IEEE international symposium on empirical software engineering and measurement*, p. 13, **2010**.

[9] L. Lagerberg and T. Skude, "The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two software development projects at Ericsson," ed, **2013**.

[10] N. Solutions. (September). *New Product Development Glossary*. URL: http://www.npd-solutions.com/glossary.html#s [Son ziyaret tarihi:25 Ekim 2017]

[11] I. Sommerville, *Software Engineering*, Ninth Edition ed.: Pearson, **2011**.

[12] E. M. Simão, "Comparison of software development methodologies based on the SWEBOK," **2011**.

[13] B. Agarwal, S. Tayal, and M. Gupta, *Software engineering and testing*: Jones & Bartlett Learning, **2010**.

[14] R. MALL, *Fundamentals Of Software Engineering*: PHI Learning Pvt. Ltd., **2009**.

[15] B. W. Boehm, "A spiral model of software development and enhancement," *Computer,* vol. 21, pp. 61-72, **1988**.

[16] Kent Beck, James Grenning, and R. C. Martin. *Çevik Yazılım Geliştirme Manifestosu*. URL: http://agilemanifesto.org/iso/tr/manifesto.html [Son ziyaret tarihi:25 Ekim 2017]

[17] E. M. Burke and B. M. Coyner, *Java extreme programming cookbook*: " O'Reilly Media, Inc.", **2003**.

[18] K. Beck, *Extreme programming explained: embrace change*: addison-wesley professional, **2000**.

[19] R. S. Pressman, *Software engineering: a practitioner's approach*: Palgrave Macmillan, **2005**.

[20] L. E. Institute. *Lean*. Available: https://www.lean.org/ [Son ziyaret tarihi:25 Ekim 2017]

[21] T. Stober and U. Hansmann, *Best Practices for Large Software Development Projects*: Springer, **2010**.

[22] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *arXiv preprint arXiv:1709.08439,* **2017**.

[23] S. R. Palmer and M. Felsing, *A practical guide to feature-driven development*: Pearson Education, **2001**.

[24] M. Al-Zewairi, M. Biltawi, W. Etaiwi, and A. Shaout, "Agile Software Development Methodologies: Survey of Surveys," *Journal of Computer and Communications,* vol. 5, p. 74, **2017**.

[25] N. E. Fenton and S. L. Pfleeger, "Software metrics: a rigorous and practical approach," *Methods,* vol. 5, p. 26, **2009**.

[26] R. E. Park, W. B. Goethert, and W. A. Florac, "Goal-Driven Software Measurement. A Guidebook," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst., **1996**.

[27] E. E. Mills and K. H. Shingler, "Software Metrics-SEI Curriculum Module SEI-CM-12-1.1," **1988**.

[28] S. Alexandre, "Software Metrics: An Overview (Version 1.0)," *CETIC asbl—University of Namur, Software Quality Lab, Belgium,* **2002**.

[29] I. O. f. Standardization. *ISO/IEC 15939:2002*. URL: https://www.iso.org/standard/29572.html [Son ziyaret tarihi:25 Ekim 2017]

[30] C. Differding, B. Hoisl, and C. M. Lott, *Technology package for the goal question metric paradigm*: Fachbereich Informatik, Univ., **1996**.

[31] V. Caldiera and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering,* vol. 2, pp. 528-532, **1994**.

[32] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, ed: sn, **2007**.

[33] M. Pai, M. McCulloch, and J. Colford, "Systematic Review: A Road Map Version 2.2. Systematic Reviews Group, UC Berkeley, 2002," ed, **2004**.

[34] K. S. Khan, G. Ter Riet, J. Glanville, A. J. Sowden, and J. Kleijnen, *Undertaking systematic reviews of research on effectiveness: CRD's guidance for carrying out or commissioning reviews*: NHS Centre for Reviews and Dissemination, **2001**.

[35] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University,* vol. 33, pp. 1-26, **2004**.

[36] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, p. 38, **2014**.

[37] J. Heidenberg, M. Weijola, K. Mikkonen, and I. Porres, "A metrics model to measure the impact of an agile transformation in large software development organizations," in *International Conference on Agile Software Development*, pp. 165-179, **2013**.

[38] M. Olszewska, J. Heidenberg, M. Weijola, K. Mikkonen, and I. Porres, "Quantitatively measuring a large-scale agile transformation," *Journal of Systems and Software,* vol. 117, pp. 258-273, **2016**.

[39] D. Poon, "A self funding agile transformation," in *Agile Conference,* pp. 9 pp.-350, **2006**.

[40] K. Sureshchandra and J. Shrinivasavadhani, "Adopting agile in distributed development," in *Global Software Engineering, ICGSE, IEEE International Conference on*, pp. 217-221, **2008**.

[41] M. Olszewska, J. Heidenberg, M. Weijola, K. Mikkonen, and I. Porres, "Did it actually go this well? A Large-Scale Case Study on an Agile Transformation."

[42] K. Korhonen, "Evaluating the effect of agile methods on software defect data and defect reporting practices-a case study," in *Quality of Information and Communications Technology (QUATIC), Seventh International Conference on the*, pp. 35-43, **2010**.

[43] R. Rasmussen, T. Hughes, J. Jenks, and J. Skach, "Adopting agile in an FDA regulated environment," in *Agile Conference, AGILE'09.*, pp. 151-155, **2009**.

[44] P. A. Beavers, "Managing a Large" Agile" Software Engineering Organization," in *Agile Conference (AGILE),* pp. 296-303, **2007**.

[45] D. Tudor and G. A. Walter, "Using an agile approach in a large, traditional organization," in *Agile Conference,* pp. 7 pp.-373, **2006**.

[46] K. Long and D. Starr, "Agile supports improved culture and quality for healthwise," in *Agile, AGILE'08. Conference*, pp. 160-165, **2008**.

[47] G. Benefield, "Rolling out agile in a large enterprise," in *Hawaii international conference on system sciences, proceedings of the 41st annual*, pp. 461-461, **2008**.

[48] K. Sureshchandra and J. Shrinivasavadhani, "Moving from waterfall to agile," in *Agile, AGILE'08. Conference*, pp. 97-101, **2008**.

[49] M. Block, "Evolving to Agile: A story of agile adoption at a small SaaS company," in *Agile Conference (AGILE),* pp. 234-239, **2011**.

[50] T. R. Seffernick, "Enabling agile in a large organization our journey down the yellow brick road," in *Agile Conference (AGILE),* pp. 200-206, **2007**.

[51] K. Nottonson and K. DeLong, "Baby steps: Agile transformation at babycenter. com," *IT Professional,* vol. 10, **2008**.

[52] E. C. Lee, "Forming to performing: Transitioning large-scale project into agile," in *Agile, 2008. AGILE'08. Conference*, pp. 106-111, **2008**.

[53] D. Goodman and M. Elbaz, " It's Not the Pants, it's the People in the Pants" Learnings from the Gap Agile Transformation What Worked, How We Did it, and What Still Puzzles Us," in *Agile, 2008. AGILE'08. Conference*, pp. 112-115, **2008**.

[54] E. Kim and S. Ryoo, "Agile adoption story from NHN," in *Computer Software and Applications Conference (COMPSAC), IEEE 36th Annual*, pp. 476-481, **2012**.

[55] K. Korhonen, "Evaluating the impact of an agile transformation: a longitudinal case study in a distributed context," *Software Quality Journal,* vol. 21, pp. 599-624, **2013**.

[56] C. Fry and S. Greene, "Large scale agile transformation in an on-demand world," in *Agile Conference (AGILE),* pp. 136-142, **2007**.

[57] C. T. Schmidt, S. Ganesha Venkatesha, and J. Heymann, "Empirical insights into the perceived benefits of agile software engineering practices: A case study from SAP," in *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 84-92, **2014**.

[58] K. Petersen and C. Wohlin, "The effect of moving from a plan-driven to an incremental software development approach with agile practices," *Empirical Software Engineering,* vol. 15, pp. 654-693, **2010**.

[59] N. L. Russo, G. Fitzgerald, and S. Shams, "Exploring adoption and use of agile methods: A comparative case study," **2013**.

[60] K. Korhonen, "Migrating defect management from waterfall to agile software development in a large-scale multi-site organization: A case study," in *International Conference on Agile Processes and Extreme Programming in Software Engineering*, pp. 73-82, **2009**.

[61] C. Pinheiro, F. Maurer, and J. Sillito, "Moving towards agility in a bureaucratic environment–RUP as a bridge between Waterfall and Agile processes," *AGILE* **2008**.

[62] M. Giblin, P. Brennan, and C. Exton, "Introducing agile methods in a large software development team: The impact on the code," *Agile Processes in Software Engineering and Extreme Programming,* pp. 58-72, **2010**.

[63] A. Anwar, A. A. Kamel, and E. Ahmed, "Agile Adoption Case Study, Pains, Challenges & Benefits," in *Proceedings of the 2nd Africa and Middle East Conference on Software Engineering*, pp. 60-65, **2016**.

[64] I. Turnu, M. Melis, A. Cau, M. Marchesi, and A. Setzu, "Introducing tdd on a free libre open source software project: a simulation experiment," in *Proceedings of the workshop on Quantitative techniques for software agile process*, pp. 59-65, **2004**.

[65] P. Abrahamsson, "Extreme programming: First results from a controlled case study", p. 259, **2003**.

[66] R. Tufail and A. A. Malik, "A case study analyzing the impact of software process adoption on software quality," in *Frontiers of Information Technology (FIT), 10th International Conference*, pp. 254-256, **2012**.

[67] L. Layman, L. Williams, and L. Cunningham, "Exploring extreme programming in context: an industrial case study," in *Agile Development Conference,* pp. 32-41, **2004**.

[68] C. P. O'Connor, "Anatomy and physiology of an Agile Transition," in *Agile Conference (AGILE),* pp. 302-306, **2011**.

[69] J. L. H. Jie, "Industrial Case Study of Transition from V-Model into Agile SCRUM in Embedded Software Testing Industries," *ACM SIGSOFT Software Engineering Notes,* vol. 41, pp. 1-3, **2016**.

[70] N. Russo, S. Shams, and G. Fitzgerald, "A Tale Of Two Agile Implementations: A Cross-Case Exploratory Analysis," in *the UK Academy for Information Systems Conference*, **2013**.

[71] L. Layman, L. Williams, and L. Cunningham, "Motivations and measurements in an agile case study," *Journal of Systems Architecture,* vol. 52, pp. 654-667, **2006**.

[72] R. K. Gupta, P. Manikreddy, and K. Arya, "Pragmatic Scrum Transformation: Challenges, Practices & Impacts During the Journey A case study in a multi-location legacy software product development team," in *Proceedings of the 10th Innovations in Software Engineering Conference*, pp. 147-156, **2017**.

[73] R. K. Gupta, P. Manikreddy, and G. Abhinandan, "Challenges in Adapting Agile Testing in a Legacy Product," in *Global Software Engineering (ICGSE), IEEE 11th International Conference*, pp. 104-108, **2016**.

[74] U. Viswanath, "Lean Transformation: How Lean Helped to Achieve Quality, Cost and Schedule: Case Study in a Multi Location Product Development Team," in *Global Software Engineering (ICGSE), IEEE 9th International Conference*, pp. 95-99, **2014**.

[75] D. Duka, "Adoption of agile methodology in software development," in *Information & Communication Technology Electronics & Microelectronics (MIPRO), 36th International Convention,* pp. 426-430, **2013**.

[76] C. Maples, "Enterprise agile transformation: the two-year wall," in *Agile Conference, AGILE'09.*, pp. 90-95, **2009**.

[77] R. Chen, R. Ravichandar, and D. Proctor, "Managing the Transition to Agile Product Development----Lessons from Cisco Systems," in *Academy of Management Proceedings*, p. 11327, **2015**.

[78] D. Wilby, "Roadmap transformation: from obstacle to catalyst," in *Agile Conference, AGILE'09.*, pp. 229-234, **2009**.

[79] K. Power, "Sensemaking and Complexity in Large-Scale Lean-Agile Transformation: A Case Study from Cisco," in *System Sciences (HICSS), 49th Hawaii International Conference*, pp. 5417-5426, **2016**.

[80] U. Viswanath, "Lean Transformation: Adapting to the change, factors for success and lessons learnt during the journey: A case study in a multi location software product development team," in *Proceedings of the 9th India Software Engineering Conference*, pp. 156-162, **2016**.

[81] R. Noordeloos, C. Manteli, and H. Van Vliet, "From RUP to Scrum in global software development: A case study," in *Global Software Engineering (ICGSE), IEEE Seventh International Conference*, pp. 31-40, **2012**.

[82] R. Coffin, "A tale of two projects [agile projects]," in *Agile Conference,* pp. 7 pp.-164, **2006**.

[83] K. Korhonen, "Adopting agile practices in teams with no direct programming responsibility–A case study," in *International Conference on Product Focused Software Process Improvement*, pp. 30-43, **2011**.

[84] E. Papatheocharous and A. S. Andreou, "Evidence of agile adoption in software organizations: An empirical survey," in *European Conference on Software Process Improvement*, pp. 237-246, **2013**.

[85] Y. Dubinsky, A. Yaeli, Y. Feldman, E. Zarpas, and G. Nechushtai, "Governance of software development: The transition to agile scenario," in *Software Applications: Concepts, Methodologies, Tools, and Applications*, ed: IGI Global, pp. 309-327, **2009**.

[86] M. Paasivaara, C. Lassenius, V. T. Heikkilä, K. Dikert, and C. Engblom, "Integrating global sites into the lean and agile transformation at ericsson," in *IEEE 8th International Conference on Global Software Engineering*, pp. 134-143, **2013**.

[87] H. Svensson and M. Host, "Introducing an agile process in a software maintenance and evolution organization," in *Software Maintenance and Reengineering, CSMR, Ninth European Conference*, pp. 256-264, **2005**.

[88] Y. Dubinsky, O. Hazzan, D. Talby, and A. Keren, "System Analysis and Design in a Large-Scale Software Project: The Case of Transition to Agile Development," in *ICEIS (3)*, pp. 11-18, **2006**.

[89] G. Lifshitz, A. Kroskin, and Y. Dubinsky, "The Story of Transition to Agile Software Development," in *International Conference on Agile Processes and Extreme Programming in Software Engineering*, pp. 212-214, **2008**.

[90] F. Ji and T. Sedano, "Comparing extreme programming and Waterfall project results," in *Software Engineering Education and Training (CSEE&T), 24th IEEE-CS Conference*, pp. 482-486, **2011**.

[91] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *EASE*, pp. 68-77, **2008**.

[92] M. Laanti, O. Salo, and P. Abrahamsson, "Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation," *Information and Software Technology,* vol. 53, pp. 276-290, **2011**.

[93] M. Paasivaara, B. Behm, C. Lassenius, and M. Hallikainen, "Towards rapid releases in large-scale xaas development at ericsson: A case study," in *Global Software Engineering (ICGSE), IEEE 9th International Conference*, pp. 16-25, **2014**.

[94] C. Robson and K. McCartan, *Real world research*: John Wiley & Sons, **2016**.

[95] I. O. f. Standardization. *ISO/IEC 15939:2002 Software engineering -- Software measurement process*. URL: https://www.iso.org/standard/29572.html [Son ziyaret tarihi:26 Ekim 2017]

[96] AgileAlliancce. *Mob Programming*. URL: https://goo.gl/STeF1F [Son ziyaret tarihi:4 Aralık 2017]

[97] eTestingHub. *Scope of User Acceptance Testing*. URL: https://goo.gl/mbxSXo [Son ziyaret tarihi:4 Aralık 2017]

[98] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering,* pp. 308-320, **1976**.

[99] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," *Software process: Improvement and practice,* vol. 14, pp. 39-62, **2009**.

[100] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on software engineering,* vol. 32, pp. 771-789, **2006**.

[101] M. Bruntink and A. Van Deursen, "Predicting class testability using object-oriented metrics," in *Source Code Analysis and Manipulation, Fourth IEEE International Workshop*, pp. 136-145, **2004**.

[102] M. Bruntink and A. van Deursen, "An empirical study into class testability," *Journal of systems and software,* vol. 79, pp. 1219-1232, **2006.**

[103] D. Sato, A. Goldman, and F. Kon, "Tracking the evolution of object-oriented quality metrics on agile projects," in *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pp. 84-92, **2007**.

[104] GMetrics. *Cyclomatic Complexity (McCabe) Metric*. URL: https://goo.gl/c6cKoS [Son ziyaret tarihi:6 Aralık 2017]

[105] Aivosto. *Complexity metrics*. URL: https://goo.gl/5VwhSX [Son ziyaret tarihi:6 Aralık 2017]

[106] R. K. Yin, *Case study research: Design and methods*: Sage publications, **2013**.

ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı : Feyza Nur KILIÇASLAN

Dğum Yeri : Eskişehir

Medeni Hali : Evli

E-posta : feyzanur.kilicaslan@gmail.com

Adresi : Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü, ofis Z10

Eğitim

Lise : Sema Yazar Anadolu Lisesi, Kayseri, 2009

Lisans : Vistula University, Computer Engineering, Varşova, 2014

Yabancı Diller

İngilizce

İş Deneyimi

Araştırma Görevlisi, Hacettepe Üniversitesi, 2015-

Deneyim Alanları

Yazılım Mühendisliği, Yazılım Kalite Yönetimi, Yazılım Metrikleri

Tezden Üretilmiş Projeler ve Bütçesi

-

Tezden Üretilmiş Yayınlar

Kılıçaslan, F.N, Tarhan, A., ve Altunel, H. “Çevik dönüşümün etkilerini nasıl ölçeriz? Orta ölçekli bir yazılım kurumu için bilgi ihtiyaçları ve metrikler”, XI. Ulusal Yazılım Mühendisliği Sempozyumu’17, CEUR Proceedings, sf. 12-23, Alanya, Türkiye, 2017.

Tezden Üretilmiş Tebliğ ile Katıldığı Toplantılar

-

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

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

Tarih: 09/01/2018

Tez Başlığı / Konusu: YAZILIM KURUMLARINDA ÇEVİK DÖNÜŞÜMÜN ETKİLERİNİN ÖLÇÜLMESİ VE DEĞERLENDİRİLMESİ : DENEYSEL BİR ÇALIŞMA

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam 64 sayfalık kısmına ilişkin, 08/01/2018 tarihinde şahsım/tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 5 'tür.

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

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

Gereğini saygılarımla arz ederim.

Tarih ve İmza
09/01/2018

Adı Soyadı: FEYZA NUR KILIÇASLAN
Öğrenci No: N14229761
Anabilim Dalı: BİLGİSAYAR MÜHENDİSLİĞİ
Programı: BİLGİSAYAR MÜHENDİSLİĞİ
Statüsü: ☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Yrd. Doç. Dr. Ayça Zarhan
(Unvan, Ad Soyad, İmza)