T.C.
BEYKENT ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
İŞLETME ANABİLİM DALI
İŞLETME YÖNETİMİ BİLİM DALI

ÇEVİK YAZILIM GELİŞTİRME YÖNETİMİNİN FİRMA PERFORMANSINA ETKİSİ

Yüksek Lisans Tezi

Tezi Hazırlayan:

Mustafa Samet ÖZKAN

İSTANBUL, 2019

T.C.
BEYKENT ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
İŞLETME ANABİLİM DALI
İŞLETME YÖNETİMİ BİLİM DALI

ÇEVİK YAZILIM GELİŞTİRME YÖNETİMİNİN FİRMA PERFORMANSINA ETKİSİ

Yüksek Lisans Tezi

Tezi Hazırlayan:

Mustafa Samet ÖZKAN

Öğrenci No:

150745381

Danışman:

Dr. Öğr. Üyesi Canan URHAN

İSTANBUL, 2019

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum **"Çevik Yazılım Geliştirme Yönetiminin Firma Performansına Etkisi"** başlıklı bu çalışmanın, bilimsel ahlâk ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 27.09.2019

Mustafa Samet ÖZKAN

T.C.
BEYKENT ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ MÜDÜRLÜĞÜ
TEZLİ YÜKSEK LİSANS SINAV TUTANAĞI

27/09/2019

Enstitümüz ***İşletme*** Anabilim Dalı ***İşletme Yönetimi*** Programı yüksek lisans öğrencilerinden ***150745381*** numaralı ***Mustafa Samet ÖZKAN***'ın *"Beykent Üniversitesi Lisansüstü Eğitim – Öğretim Yönetmeliği"*nin ilgili maddesine göre hazırlayarak, Enstitümüze teslim ettiği ***"Çevik Yazılım Geliştirme Yönetiminin Firma Performansına Etkisi"*** konulu tezini, Yönetim Kurulumuzun 28/05/2019 tarih ve 2019/22 sayılı toplantısında seçilen ve Taksim Yerleşkesinde toplanan biz jüri üyeleri huzurunda, Beykent Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 29. maddesinin 3. fıkrası gereğince (45.) dakika süre ile aday tarafından savunulmuş ve sonuçta adayın tezi hakkında ~~***oyçokluğu/oybirliği***~~ ile ***Kabul/~~Red~~ veya ~~Düzeltme~~*** kararı verilmiştir.

İşbu tutanak, 4 nüsha olarak hazırlanmış ve Enstitü Müdürlüğü'ne sunulmak üzere tarafımızdan düzenlenmiştir.

DANIŞMAN
Dr. Öğr. Üyesi Canan URHAN
(Beykent Üniversitesi)

ÜYE
Doç. Dr. Erkut ALTINDAĞ
(Beykent Üniversitesi)

ÜYE
Dr. Öğr. Üyesi Güldenur ÇETİN
(İstanbul Ticaret Üniversitesi)

Adı ve Soyadı : Mustafa Samet ÖZKAN
Danışmanı : Dr. Öğr. Üyesi Canan URHAN
Türü ve Tarihi : Yüksek Lisans, 2019
Alanı : İşletme Yönetimi
Anahtar Kelimeler : Bilgi Teknolojileri, Performans, Proje Yönetimi, Çevik Yazılım Geliştirme, Yazılım Geliştirme Metodolojileri, Scrum

ÖZ

ÇEVİK YAZILIM GELİŞTİRME YÖNETİMİNİN FİRMA PERFORMANSINA ETKİSİ

Teknolojide yaşanan gelişmeye ayak uydurma ve rekabet avantajını sağlama ihtiyacından dolayı işletmeler, yeni yöntemler kullanmaya ve yaratıcı uygulamalar geliştirmeye çalışmaktadır. Uygulama geliştirme sürecinde kullanılan metodoloji sürecin başarılı ya da başarısız olmasında en önemli etkiye sahiptir. Aynı şekilde çalışanların daha etkili ve verimli çalışabilmesi, performans ve motivasyonun arttırılması için uygulanacak metodolojinin seçimine dikkat edilmesi de önem arz etmektedir.

Bu tez çalışması, bankacılık sektöründe faaliyet gösteren şirketlerin yazılım geliştirme süreçlerinde tercih ettiği yöntemleri araştırmakta ve bu yöntemlerin firma performansına olan etkisini ölçmeyi amaçlamaktadır. Bu maksatla yazılım geliştirme yöntemleri incelenmiş, yöntemlerin çalışan performansına etkisi üzerine araştırma yapılmıştır. Araştırma kapsamında analiz yapabilmek adına anket/ölçek soruları hazırlanmıştır. Bu çalışmada geleneksel ve çevik yazılım geliştirme yöntemleri karşılaştırılmış, firma performansına etkisi SPSS programı kullanılarak istatistiki analiz yöntemleriyle incelenmiş, çevik yöntemlerin çalışanlar tarafından kabul gördüğü ve iş süreçlerinde başarılı bir şekilde uygulanmasının sonucunda firma performansına olumlu yönde etki ettiği sonucu ortaya konmuştur.

Name and Surname : Mustafa Samet ÖZKAN
Supervisor : Dr. Lecturer Canan URHAN
Degree and Date : Master, 2019
Major : Business Administration
Key Words : Information Technologies, Performance, Project Management, Agile Software Development, Software Development Methodology, Scrum

ABSTRACT

THE EFFECT OF AGILE SOFTWARE DEVELOPMENT MANAGEMENT ON COMPANY PERFORMANCE

Due to the need to adapt to the developments in technology and to provide competitive advantage, enterprises are trying to use new methods and to develop creative applications. The methodology used in the application development process has the most important effect on the success or failure of the process. In the same way, it is also important to pay attention to the selection of the methodology that will be applied to increase the performance and motivation of the employees.

This thesis investigates the methods preferred by software companies in the banking sector and measures the effects of these methods on firm performance. For this purpose, software development methods were examined and the effect of these methods on employee performance was investigated. Questionnaire / scale questions were prepared in order to conduct analysis. In this study, traditional and agile software development methods were compared, the effect on firm performance was analyzed with statistical analysis methods using SPSS program, and it was revealed that agile methods were accepted by the employees and as a result of successful application in business processes, they had a positive effect on firm performanc

İÇİNDEKİLER

Sayfa No.

ÜÇÜNCÜ BÖLÜM
METODOLOJİ

TABLOLAR LİSTESİ

Sayfa No.

ŞEKİLLER LİSTESİ

Sayfa No.

KISALTMALAR

AM : Çevik Modelleme

BT : Bilgi Teknolojileri

COBIT : Bilgi ve İlgili Teknolojiler İçin Kontrol Amaçları

CPM : Kritik Yol Metodu

DSM : Doğrusal Sıralı Model

FDD : Özellik Odaklı Geliştirme

IPMA : Uluslararası Proje Yönetimi Derneği

ITIL : Bilgi Teknolojileri Altyapı Kütüphanesi

LD : Yalın Geliştirme

OMG : Nesne Yönetim Grubu

OOAD : Nesneye Dayalı Analiz ve Tasarım

PERT : Program Değerlendirme ve Gözden Geçirme Tekniği

PM : Proje Yöneticisi

PMBOK : Proje Yönetimi Bilgi Grubu

PMI : Proje Yönetimi Enstitüsü

PO : Proje Ofisi

RAD : Hızlı Uygulama Geliştirme

RUP : Birleşik Rasyonal İşlem

SDLC : Yazılım Geliştirme Yaşam Döngüsü

SR : Sprint Retrospektif

TDK : Türk Dil Kurumu

TOGAF : Açık Grup Mimarisi Çerçevesi

UML : Birleşik Modelleme Dili

UP : Birleştirilmiş İşlem

XP : Ekstrem Programlama

WBS : İş Dağılımı Yapısı

GİRİŞ

20. yüzyılda teknolojinin gelişmesine paralel bir şekilde bilgisayarlar insan hayatında önemli bir yer edinmiştir. Aynı yüzyılın ikinci çeyreğinden itibaren internetin ortaya çıkması ve yaygınlaşmasıyla birlikte bu cihazların kullanım amacında değişiklikler gözlenmiş, insanlar bilgisayarlar vasıtasıyla neler yapabileceklerini keşfetmeye başlamıştır. Bilgisayarlar, akıllı telefonlar ve tabletler hayatımızı kolaylaştırmakta, tüm işlemler mekan kısıtlaması olmadan kolay ve hızlı bir şekilde sonuçlanmaktadır. İşlemlerin hızlı bir şekilde sonuçlanabilmesi için elbette "donanım" olarak adlandırdığımız cihazlara ihtiyacımız bulunmaktadır. Fakat bu cihazlar içerisindeki "yazılım" olmadan aynı işlevi gerçekleştiremeyeceği düşünüldüğünde donanım ve yazılım ayrılmaz bir bütün haline gelmektedir.

Teknolojik gelişimin zamanla bu denli hızlanması ve özellikle 20. yy' da başlayan bu trendin 21. yy'da yukarı doğru ivmelenmesiyle birlikte değişim artık yıllar, aylar değil günler seviyesine inmiş bir olgu olarak karşımıza çıkmaktadır.

Teknolojide yaşanan bu değişimden finans sektöründe büyük bir rol oynayan bankalar da etkilenmiştir. Geçmişi yüzyıllar öncesine dayanan bankacılık ve finans kurumlarında artık tüm işlemler bilgisayarlar vasıtasıyla yapılmaktadır. Bankaların iş yapma şekilleri bilgisayarların etkisiyle değişmeye zorlanmış bu etki bankaların teknolojik alt yapılarındaki gelişimi tetiklemiştir. Günümüzde insanlar evlerinden ya da internete ulaşabildikleri herhangi bir yerden tüm bankacılık işlemlerini saniyeler içinde yapabilmektedir. Modern bankacılık anlayışı zaman içinde bankaları birer teknoloji şirketi olmaya doğru yönlendirmektedir. Bankaların yüksek maliyetli teknoloji içeren yazılımlar geliştirmesi, Bilgi Teknolojileri alanlarına yatırım yapmaları da gelişen teknolojiye ayak uydurma uğraşından dolayı gerçekleşmektedir.

Teknolojinin gelişmesine paralel olarak yöntemler değişmiş, yazılımlar iyileştirilmiş ya da çağa ayak uyduramadığı dönemde ise değişmeye zorlanmıştır. Yazılım geliştirme süreci bu noktada büyük bir önem arz etmektedir. Sürecin düzgün ve belirli bir plana göre yürütülebilmesi için birden fazla yöntem kullanılmaktadır. Bazı yöntemler yıllardır kült olarak kullanılmış, mevcut teknolojiler ile birlikte başarılı sonuçlar elde edilmiştir. Zamanla teknolojinin olduğu gibi yöntemlerin de değişmesi gerektiği fark edilmiştir.

Çalışmanın birinci bölümünde proje ve proje yönetimi kavramı, tarihçesi ve yaklaşımları incelenmiştir. Bu bölümde proje kavramının kökenleri, proje yönetiminin tarihsel gelişimi ve proje yönetimi yaklaşımları detaylı olarak açıklanmıştır.

Çalışmanın ikinci bölümünde "Yazılım Geliştirme Yaşam Döngüsü" kavramı ele alınmış, kavramın ne olduğu, nasıl çalıştığı, yararları ve aşamaları ifade edilmiştir. Aynı bölümde bu kavramı temel alan popüler modeller açıklanmış ve günümüzde kullanılan uygulamalar örneklendirilmiştir. Geleneksel ve çevik yazılım geliştirme yöntemleri anlatılmıştır. Geleneksel yaklaşımın temelleri, doğrusal sıralı modelin (Waterfall) özellikleri ve sınırlamaları açıklanmış, bunun haricinde en çok tercih edilen birkaç geleneksel yöntemden de örnekler verilmiştir. Çevik yöntemlerin oluşumu ve bu yöntemlere temel olan "Agile Manifesto" kavramı ele alınmıştır. Manifesto ile deklare edilen maddeler üzerinden örneklemeler yapılmış ve detaylı şekilde ifade edilmiştir. Çevik yöntemlerin arasında en çok tercih edilen beş yöntem detaylı bir şekilde anlatılmıştır.

Çalışmanın üçüncü bölümünde araştırma yöntemi açıklanmış, elde edilen verilen istatistiki analiz yöntemleriyle değerlendirilmiştir.

Çevik yazılım geliştirme yöntemi ve etkilerini ortaya koyan Şahin (2016), Kıranç (2018), Kır (2014), Derindere (2013) ve Çulha'nın (2014) çalışmaları gibi literatürde çalışmalar yapılmıştır.

Bu çalışma kapsamında çevik yazılım geliştirme yöntemi ve firma performansı üzerindeki etkileri, teorik ve ampirik olarak incelenmektedir. Çalışma kapsamında geliştirilen hipotezler, araştırma bölümünde İstanbul bölgesinde bankacılık süreçleriyle alakalı yazılım firmalarında çalışmakta olan 260 iş görenden toplanan verilerin istatistiksel analizlere tabi tutulması ile test edilmiştir. Sonuç kısmında ise bulgular yorumlanmıştır.

Bu kapsamda çalışmanın hipotezleri aşağıdaki gibi belirlenmiştir:

$\boldsymbol{H_1}$**:** Çevik proje geliştirme yönteminin firma performansına doğrudan ve pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1A}}$**:** Kapsam yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1B}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1C}}$**:** Maliyet yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

Hipotezleri doğrulamak için SPSS 21.0 analiz programı yardımıyla yöntem kısmından sırasıyla güvenilirlik, geçerlilik, faktör, korelasyona ve regresyon analizleri yapılmıştır. Bu amaçla çevik yöntemlerin firma performansına etkisi belirleyebilmek için durumla ilgili alan literatür taraması yapılarak soruların formu oluşturulmuştur. Birincil verilere ulaşabilmek için en fazla kullanılan metotlardan olan anket tekniği kullanılarak belirlenen çalışan gurubunun demografik özelliklerine göre çevik yöntemlerin firma performansına etkisi belirlenecektir.

Bu çalışma kapsamında *"Çevik yöntemlerin firma performansına etkisi nedir?"* sorusuna cevap aranmakta ve bu çerçevede yöntemler incelenmekte ve çalışan performansına olan etkileri araştırılmaktadır.

BİRİNCİ BÖLÜM
PROJE YÖNETİMİ

1.1. Proje Kavramı

Kelime kökeni olarak "proje", Latince "projectum" kelimesinden gelmekte olup, dilimize Fransızcadan geçtiği düşünülmektedir. Türk Dil Kurumu tarafından "değişik alanlarda önceden plan ve programa alınmış, maliyeti hesaplanmış, kurum ve kuruluşların yönetim organları tarafından onaylanmış, kısa ve uzun vadeye bağlanarak özel kurum veya devlet adına gerçekleştirilmesi kabul edilmiş bilimsel çalışma tasarısı" olarak ifade edilmektedir (Türk Dil Kurumu, 2019).

Proje, rutin bir operasyonel işlem değil, belirli bir hedefi gerçekleştirmek için tasarlanmış işlemler bütünüdür. Projeler önceden belirlenmiş bir başlangıç ve bitiş tarihine, sınırlı kaynaklara sahiptir. Proje ekipleri genellikle farklı kurumlardan ve birden fazla coğrafyada birlikte çalışmayan insanları içermektedir. Gelişmiş bir iş süreci için "yazılımın geliştirilmesi, bir binanın ya da köprünün inşası, doğal bir felaket sonrası rahatlama çabası, satışların yeni bir coğrafi pazara yayılması" hepsi bir proje olarak değerlendirilebilir. Projelerin, kuruluşların ihtiyaç duyduğu zaman aralığında, bütçeye uygun sonuçları elde etmek, öğrenmeyi ve entegrasyonu sağlamak için profesyonel bir şekilde yönetilmesi gerekmektedir.

Proje Yönetim Enstitüsü (PMI) projeyi "eşsiz bir ürün, hizmet veya sonuç elde etmek için yapılan geçici bir çaba" olarak tanımlamıştır (PMI, 2017: 32). Proje hedefleri, projenin sonunda hedefe ulaşılıp ulaşılmadıklarına bakılmaksızın durumu ortaya koymaktadır. Bu hedefler SMART Kriterleri olarak adlandırılmıştır (Doran, 1981: 1-2). SMART, proje yönetimi, çalışan-performans yönetimi ve kişisel gelişim gibi amaçların belirlenmesinde rehberlik edecek kriterleri veren bir anımsatıcı / kısaltmadır (Mind Tools, 2019).

SMART kelimesi bu kriterleri oluşturan hedeflerin baş harflerinin bir araya gelmesiyle elde edilmiştir.

- (S)pecific: Belirli
- (M)easurable: Ölçülebilir
- (A)chievable: Ulaşılabilir
- (R)ealistic: Gerçekçi
- (T)ime-Based: Zamana Dayalı

1.2. Proje Özellikleri

Çoğu zaman rutin yapılması gereken faaliyetler de proje olarak nitelendirmektedir. Projeler sadece bir kere yapılırken, günlük operasyonlar devamlılığı olan ve tekrarlanan işlerdir. Ayrıca, projeler tamamlandığında yeni bir ürün veya hizmet ortaya çıkmaktadır (Gray ve Larson, 2011: 5-7). Bu tanım bize projenin sahip olduğu özellikler hakkında fikir vermektedir. Projelerin belirli bir süresinin olduğu, sürekli tekrarlanan aktiviteler olmadığı ve bir hedefi olduğu anlaşılmaktadır.

Proje kavramının, literatürde (Oxford, 2016: 991) (Business Dictionary, 2018) (Manning, 2008: 31) yer alan tanımları incelenmiş, proje ile ilgili ortak özellikler Tablo 1'de özetlenmiştir.

Tablo 1 : Proje Özellikleri

Projeler tekrarlanmayan faaliyetlerden oluşan özgün çabalardır.
Projelerin başlangıç ve bitiş zamanları belirlidir.
Proje geçicidir, süresi içerisinde tamamlanır veya iptal edilir.
Projelerin tanımlanan bir amacı bulunur.
Projeler önemli belirsizlikler ve risk içerir. Proje ilerledikçe belirsizlikler azalır.
Projeler birbirini takip eden veya eşzamanlı yürütülen faaliyetlerden oluşur. Bu faaliyetler arasında teknoloji ve kaynak kısıtları nedeniyle karmaşık bir ilişki bulunur.
Projeler çeşitli kaynaklara ihtiyaç duyar ve maliyet kısıtları altında yürütülür.

Kaynak: (Oxford, 2016: 991) (Business Dictionary, 2018) (Manning, 2008: 31) (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

Tablo 1'de paylaşılan özellikler incelendiğinde proje, "belirli bir amacı ve kaynak kısıtı (zaman, maliyet ve insan) bulunan işlemler bütünü" olarak ifade edilebilir. Projelerin amacına uygun hareket edebilmesi ve kaynakların doğru şekilde kullanılabilmesi için sürecin profesyonel bir yaklaşımla idare edilmesi gerekmektedir. Bu noktada ise "Proje Yönetimi" kavramı ortaya çıkmaktadır. Bu kavram ve detayları bir sonraki bölümde anlatılmıştır.

1.3. Proje Yönetimi

Proje yönetimi, şirket kaynaklarının, belirli hedef ve amaçların yerine getirilmesi için belirlenmiş olan kısa vadeli bir çalışma için planlanması, organize edilmesi, yönetilmesi ve kontrol edilmesidir. Proje yönetiminin öncelikli zorluğu, proje kısıtlamaları dâhilinde tüm proje hedeflerine ulaşmaktır (Kerzner, 2009: 35).

Proje kısıtları genellikle geliştirme sürecinin başında oluşturulan proje belgelerinde açıklanmaktadır. Birincil kısıtlamalar kapsam, zaman, kalite ve bütçedir. İkincil kısıtlama ise gerekli girdilerin tahsis edilmesini optimize etmek ve bunları önceden tanımlanmış hedeflere ulaşmak için uygulamaktır. Proje yönetiminin asıl amacı, müşterinin hedeflerine uygun eksiksiz bir proje üretmektir. Müşterinin hedefleri açıkça belirlendikten sonra, proje yöneticileri, tasarımcılar ve proje paydaşları gibi projeye katılan diğer kişiler tarafından alınan tüm kararlar projenin ilerleyişini etkileyecektir. Kötü tanımlanmış veya çok katı bir şekilde belirlenmiş proje yönetimi hedefleri karar verme noktasında olumsuz etki bırakcaktır (PMI, 2017: 40).

Bir proje, tipik olarak yararlı bir değişim ya da hedeflere ulaşmak için benzersiz amaçla gerçekleştirilen belirli bir başlangıç ve bitiş süresi (genellikle zaman kısıtlı ve genellikle fonlama veya personel tarafından kısıtlanan) ile benzersiz bir ürün, hizmet, katma değer veya sonuç üretmek için tasarlanmış geçici bir çabadır. Projelerin geçici niteliği, ürün veya hizmet üretmek için tekrarlanan, kalıcı veya yarı kalıcı işlevsel faaliyetleri olan normal faaliyetlerinin aksinedir. Uygulamada, bu tür farklı üretim yaklaşımlarının yönetimi, farklı teknik becerilerin ve yönetim stratejilerinin geliştirilmesini gerektirmektedir (Lock, 2007: 1-5).

Bankacılık sektörü özelinde bir örnekleme düşünüldüğünde mevcut uygulamalar üzerinde ortaya çıkan bir ihtiyaç ya da istek yeni bir talebi meydana getirmektedir. Bu talep fizibilite sonrasında üzerinde çalışılmaya yeterli ise proje olarak değerlendirilecektir. Talebin kapsamının belirlenmesi, planlanması, yürütülmesi ve sonuca bağlanması sürecine de proje yönetimi denilmektedir.

Proje yönetimi bilgilerin, becerilerin, araçların ve tekniklerin proje gereksinimlerini yerine getirmek amacıyla proje aktivitelerine uygulanması şeklinde tanımlanmaktadır (PMBOK, 2013: 6).

1.3.1 Proje Yönetimi Tarihçesi

1900 yılına kadar inşaat mühendisliği projeleri genellikle mimarlar, mühendisler ve usta inşaatçılar tarafından yönetilmiştir (Lock, 2007: 1-5). 1950' li yıllarda kuruluşlar proje yönetimi araç ve tekniklerini karmaşık mühendislik projelerine sistematik olarak uygulamaya başlamış, bir disiplin olarak proje yönetimi, sivil inşaat, mühendislik ve ağır savunma faaliyeti gibi çeşitli uygulama alanlarından geliştirilmiştir (Cleland ve Gareis, 2006: 4-5).

Henry Gantt ve Henri Fayol, proje yönetimi kavramının öncüleri olarak kabul edilmektedir. Henry Gantt, kendi tasarımı olan Gantt grafiğini proje yönetim aracı olarak kullanmasıyla planlama ve kontrol tekniklerini geliştirmiştir (Stevens, 2002: 15). Henri Fayol ise proje ve program yönetimi ile alakalı bilginin temelini oluşturan beş yönetim fonksiyonunu geliştirmesiyle bu övgüyü kazanmıştır (Witzel, 2003: 83-85).

Gantt ve Fayol için F. W. Taylor'un bilimsel yönetim teorileri öğrencileri denilmektedir. Çalışmaları, çalışma dağılımı yapısı (WBS) ve kaynak tahsisi dâhil olmak üzere modern proje yönetimi araçlarının öncüsü olarak görülmektedir (Kwak, 2003: 1-9).

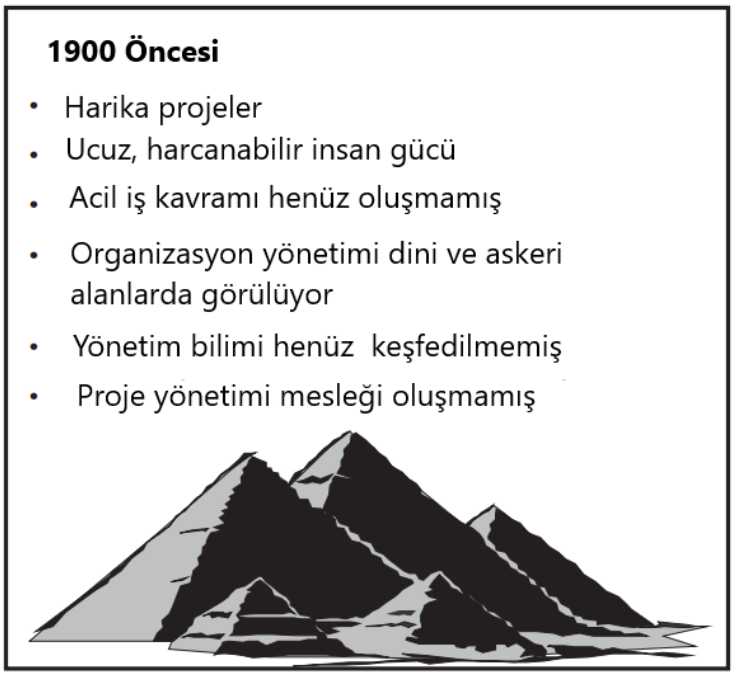
Snyder ve Kline'a (1987) göre, modern proje yönetimi dönemi 1958'de CPM/PERT'in gelişimi ile başlamıştır (Kwak, 2003: 1-9). CPM/PERT kavramlarına modern proje yönetiminin başlangıcında ayrı bir önem verilmiştir. CPM/PERT bazı noktalarda örtüşmesine rağmen aslında iki farklı alanda (donanma ve kimya endüstrisi) ve iki ayrı şekilde geliştirilmiştir. 1958'de ABD donanması, nükleer savaş başlıkları taşıyan ilk denizaltı tarafından başlatılan balistik füzeler (SLBM) olan Polaris projesine önderlik etmiştir. Polaris projesi sayesinde ABD donanmasına günümüzün en yaygın kullanılan tekniklerinden birini geliştirmek için kaynak ayrılmıştır. Bu teknik Program Değerlendirme İnceleme Tekniği (PERT) olarak adlandırılmaktadır. Projenin zamanlamasıyla ilgili yüksek karmaşıklık ve belirsizlik nedeniyle, PERT, projenin farklı zamanlama senaryolarını görselleştirmek için uygun bir yöntem olarak gösterilmektedir (Seymour, 2014: 233-238).

Kritik Yol Yöntemi (CPM), büyük bir inşaat projesi yürüten E.I du Pont de Nemours şirketinin çalışmaları sonucunda, PERT ile neredeyse aynı anda icat edilmiştir. Proje, büyük bir kimyasal tesisin inşasını içermektedir. CPM projenin maliyetini ve zamanını doğru bir şekilde tahmin etmesi gereken bir çalışmanın sonucu olarak elde edilmiştir. Başlangıçta, geliştirdikleri

yöntem Proje Planlama ve Çizelgeleme (PPS) olarak adlandırılsa da daha sonra teknik CPM yöntemine dönüştürülmüştür (Seymour, 2014: 233-238).

Modern proje yönetimi tarihi "proje yönetimi" kavramının ortaya çıkmasıyla başlamıştır. Proje yönetiminin tarihçesi yaşanan gelişmeler göz önüne alındığında beş ana başlıkta incelenecektir.

- 1900 ve Öncesi
- 1900 – 1949 Dönemi
- 1950 – 1969 Dönemi
- 1970 – 1989 Dönemi
- 1990 – 2000+



Şekil 1. 1900 Öncesi

Kaynak: (Lock, 2007: 1-5)

Antik çağlardan kalma projeler, mimari ve endüstriyel kültürümüzde etkileyici miras bırakmıştır. Bu mirasın, bugün kolayca ve ucuz bir şekilde mevcut olan teknoloji olmadan nasıl başarıldığı da merak edilmektedir. Bununla birlikte, birkaç kayda değer hayırsever işveren dışında, işçilerin refahı ve güvenliği ile ilgili endişeler genellikle yoksun ve birçok erken proje

çalışanı yaralanma, hastalık ve saf fiziksel yorgunluk nedeniyle hayatlarını kaybetmiştir. İnsanlar genellikle ucuz ve harcanabilir bir kaynak olarak kabul edilmiştir (Lock, 2007: 1-5).

Resmi yönetim organizasyon yapıları erken zamanlardan beri var olmuş, ancak bu yapılar endüstriden ziyade askeri, kilise ve sivil idarelerde gelişmiştir. Sanayi alanına organizasyonel yapılanma çok sonradan gelmiştir. Ming hanedanlığına (1368-1644) kadar bilinen tarihte, köprü yapım işlemi için organizasyon şeması Çin ordusunda görev yapan askerler tarafından oluşturulmuştur (Lock, 2007: 1-5).

Şekil 2. 1900 – 1949 Dönemi

Kaynak: (Lock, 2007: 1-5)

Hızlı sanayileşme ve 1. Dünya Savaşı'ndaki mühimmat üretim talepleri, Elton Mayo ve Frederick Winston Taylor gibi fabrikalarda insan ve verimlilik üzerine çalışan kişilerin yönetim bilimi ve endüstri mühendisliğine yönelmesini sağlamıştır (Kanigel, 1997: 85-86). Henry Ford, Model T otomobiliyle ünlü üretim hattı imalatını yapmış ve özellikle proje yöneticileri için önemli olan, özellikle Taylor'da çalışan Henry Gantt (1861-1919), günümüzde hala popüler ve olan "Gantt Şemaları"nı geliştirmiştir (Lock, 2007: 1-5).

Kritik yol ağlarının ilk örneklerinin 1950'den önce geliştiği genel olarak kabul edilmese de, değerleri o sırada yaygın olarak takdir edilmemiştir. İlk örneklerin bilgisayarların varlığı olmadan, değişime karşı esnek olmadığı ve çalışma programlarına tercüme etmekte zorlandığı ifade edilmiştir. Bu nedenle kullanımı zor olarak sınıflandırılmıştır. Gantt'ın çubuk grafikleri tercih edilmiş, genellikle hareketli manyetik, geçmeli şeritler veya kartlar kullanılarak yeniden zamanlamaya izin veren özel listelerde kurulmuştur. İşlerin insanlara atanması, makinelere ve tatil programlarına tahsisine kadar her şey, genellikle ofis duvarlarında belirgin bir şekilde görüntülenen çizelgelerle kontrol edilmiştir (Seymour, 2014: 233-238).

Şekil 3. 1950 – 1969 Dönemi

Kaynak: (Lock, 2007: 1-5)

Terminal dijital bilgisayarlarının ortaya çıkışı, kritik yol ağlarının işlenmesini ve güncellenmesini daha hızlı ve kolay hale getirmiştir. Amerikan savunma endüstrisi ve Du Pont şirketi, 1950'lerde bu güçlü planlama ve çizelgeleme aracından istifade edebilecek organizasyonlar arasında gösterilmiştir. İmalat ve inşaat endüstrileri kısa sürede bu yeni yöntemlerin faydalarını benimsemiştir. Bu dönemde bilgisayarlar büyük, oldukça pahalı ve kendilerine özel tasarlanmış klimalı temiz odalarda muhafaza edilmiştir. Sermaye ve işletme maliyetleri, büyük organizasyonların dışındaki tüm şirketlerin bütçelerini aştığından dolayı pek çok planlama uzmanı bilgisayarlar proje yönetiminde kullanma şansı elde edememiştir. Proje

yönetimi, henüz saygın bir meslek olarak görülmese de tanınmış bir iş tanımı haline gelmiştir (Lock, 2007: 1-5).

Proje yönetiminin kurumsallaşma süreci, şu anda Uluslararası Proje Yönetimi Birliği (IPMA) olarak bilinen, dünyanın ilk proje yönetim birliğinin oluşturulmasıyla başlamış, dört yıl sonra, ABD'de Proje Yönetim Enstitüsü (PMI) kurulmuştur (Seymour, 2014: 233-238).

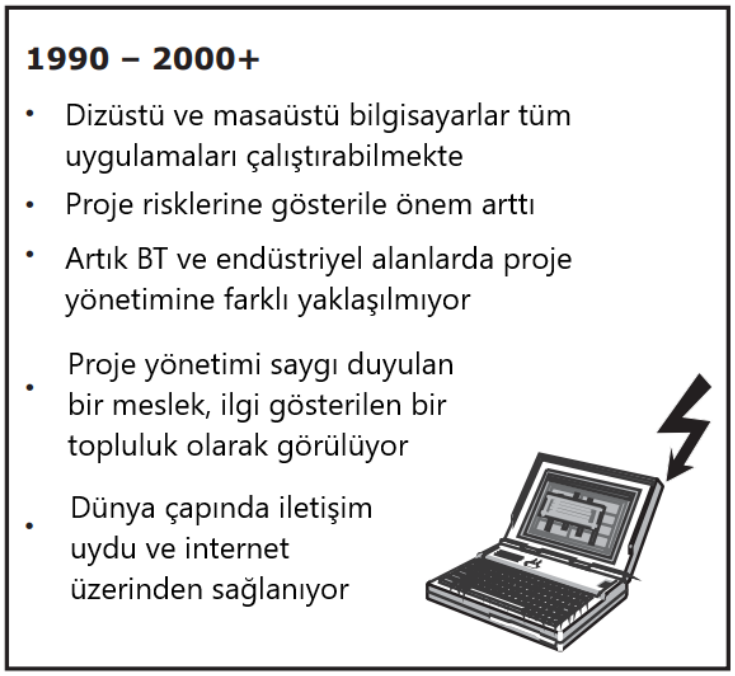
Şekil 4. 1970 – 1989 Dönemi

Kaynak: (Lock, 2007: 1-5)

Bu dönemde BT alanında hızlı bir büyüme gözlemlenmiştir. Endüstriyel proje yönetimi daha önce olduğu gibi devam etmiş, ancak daha fazla proje yönetimi yazılımı geliştirilmiş ve BT kavramının daha geniş tanınması ile devam etmiştir. Bu süreçte proje yöneticileri BT uzmanlarına çok daha az bağımlı hale gelmiştir. Proje yöneticileri, proje yönetimi yazılımını çalıştırabilecek masaüstü bilgisayarına sahip olmuş, ürettikleri renkli, karmaşık grafiklerle süreçlerini yürütebilir duruma gelmiştir (Lock, 2007: 1-5).

Bilgisayarların verimliliğinin artması projeleri yönetmek için gereken karmaşık verileri idare edebilecek ve düzenleyebilecek bir yazılım geliştirmeye izin vermiştir. 1980'lerde proje yönetimi programları Projeler Kaynak Organizasyonu Yönetim Planlama Tekniği II

(PROMPT II) proje yönetimi modeline dayanmaktayken; daha sonraki bir zamanda Kontrollü Ortamlarda Amaçlar (PRINCE) modeline dahil edilmiştir (Historic Projects, 2019).



Şekil 5. 1990 - 2000 Sonrası

Kaynak: (Lock, 2007: 1-5)

Bu dönemde teknoloji değişim için itici bir güç olmaya devam etmiş ve proje yöneticilerinin yaptıklarını da büyük ölçüde etkilemiştir. 1996 yılında PRINCE, PRINCE2'ye yükseltilmiş, 1997'den kısa bir süre sonra Kritik Zincir Proje Yönetimi (CCPM) adı verilen alternatif bir yöntem tanıtılmıştır (Seymour, 2014: 233-238). CCPM, Eliyahu M. Goldratt tarafından geliştirilen "projeleri planlama ve yönetme" yöntemidir. TOC'den türetilmiş, CPM ve PERT'den farklı olarak yöntem, esas olarak projeyi tamamlamak için gereken özellikler yerine gerekli kaynakları vurgulamıştır (Goldratt, 1997: 120).

Bu dönemde artık proje yönetimi iki ayrı dal (endüstri ve BT) olarak görülmemektedir. Projelerin daha hızlı ve daha iyi sonuçlar getirebilmesi için şirketler daha geniş ve memnuniyet verici bir bir yaklaşımı tercih etmektedir (Lock, 2007: 1-5).

2001 yılında Çevik Manifesto yazılmıştır. Çevik Manifesto, yazılım geliştirme ekiplerinin bir takım olarak iyi performans göstermelerini amaçlayan bir dizi temel değer üzerine kurulmuştur (Fowler ve Highsmith, 2001).

Proje Yönetimi Yaklaşımları

2017 yılında yapılan bir araştırmaya göre, herhangi bir proje başarısının, projeyi etkileyen bağlamsal dinamiklerle dört ana unsur arasındaki etkileşimden kaynaklandığı öne sürülmektedir (Olivier, 2017: 145).

Bu dört unsur aşağıdaki şekilde ifade edilmiştir.

- **Plan:** Planlama ve tahmin faaliyetleri.
- **Süreç:** Tüm faaliyetlere ve proje yönetimine genel yaklaşım.
- **İnsan:** İş birliğinin ve iletişimin nasıl yapıldığı.
- **Güç:** Yetki sınırları, karar vericiler, uygulama politikaları ve benzerleri.

Proje planlamasının, çıktılara (ürün odaklı) veya faaliyetlere (süreç odaklı) dayalı olarak birkaç uzantısı bulunmaktadır. Kullanılan metodolojiden bağımsız olarak, tüm proje hedeflerine, zaman çizelgesine ve maliyetine, ayrıca tüm katılımcıların ve paydaşların rol ve sorumluluklarına dikkat edilmesi gerekmektedir (Jensen, 2017).

İKİNCİ BÖLÜM
YAZILIM GELİŞTİRME METODOLOJİLERİ

2.1.Yazılım Geliştirme Kavramı

Yazılım geliştirme; çeşitli yazılım dilleri kullanılmak suretiyle bir yazılım ürünü, uygulaması ve yazılım iskeletinin oluşturulma sürecidir. Araştırma, geliştirme, yeniden kullanım, bakım ve test aşamalarını içeren ideal olarak yapılandırılmış ve planlanmış bir işlemler bütünüdür (Fabe, 2010). Bu işlemler Yazılım Geliştirme Yaşam Döngüsü (SDLC) olarak adlandıracağımız bir süreci oluşturmaktadır.

"Yazılım geliştirici" mesleği, ilk bilgisayarlardan ve onların operatörlerinden, ENIAC ve vakum tüplerinin günlerine kadar devam etmiştir. Yazılım geliştirmeye yönelik pratikler ve yöntemler, bilgisayarın icadından bu yana geçen yıllar boyunca gelişmiştir. Bu yöntemler, bilgisayar donanımı, geliştirme araçları ve yazılım geliştirme ekiplerinin yönetiminde modern düşünceye uyum sağlamıştır. Bu ilerlemeyle birlikte özel ve halka açık şekilde yürütülen yazılım geliştirme çalışmaları sonucunda dünya çapında yeni yazılım geliştirme yöntemleri oluşmuştur (Columbia, 2019).

IBM tarafından yapılan araştırmalara göre yazılım geliştirme kavramı "Yazılımı oluşturma, tasarlama, dağıtma ve destekleme sürecine adanmış bir dizi bilgisayar bilimi etkinliği anlamına gelir." şeklinde ifade edilmiştir. Yazılımın kendisi, bilgisayara ne yapması gerektiğini söyleyen talimatlar veya programlardır. Donanımdan bağımsızdır ve bilgisayarları programlanabilir yapmaktadır. Bu araştırma yazılım geliştirme kavramını dört başlık altında toplamaktadır (IBM, 2017).

- **Sistem Yazılımı :** İşletim sistemleri, disk yönetimi, yardımcı programlar, donanım yönetimi ve diğer operasyonel ihtiyaçlar gibi temel işlevleri sağlayan yazılım çeşididir.
- **Program Yazılımı :** Programcılara, metin editörlerine, derleyicilere, bağlayıcılara, hata ayıklayıcılara kod oluşturmak için gerekli araçlar vermek amacıyla yazılan yazılım çeşididir.

- **Uygulama Yazılımı :** Kullanıcıların görevleri gerçekleştirmesine yardımcı olmak için geliştirilen ofis verimlilik paketleri, veri yönetimi yazılımı, medya oynatıcılar ve güvenlik programları bunlara örnek olarak verilebilir.
- **Gömülü Yazılım :** Gömülü sistemler yazılımı, tipik olarak bilgisayar olarak kabul edilmeyen makineleri ve cihazları, telekomünikasyon ağlarını, arabaları, endüstriyel robotları ve daha fazlasını kontrol etmek için kullanılmaktadır.

Yazılım geliştirme işlemi öncelikle programcılar, yazılım mühendisleri ve yazılım geliştiriciler tarafından gerçekleştirilmektedir. Bu roller birbirleriyle etkileşime girip örtüşmekte, aralarındaki dinamik gelişim departmanları ve topluluklar arasında büyük farklılıklar gösterebilmektedir (IBM, 2017).

- **Programcılar/Kodlayıcılar :** Programcılar veya kodlayıcılar, bilgisayarları veritabanlarını birleştirmek, çevrimiçi siparişleri işlemek, iletişimleri yönlendirmek, arama yapmak veya metin ve grafikleri görüntülemek gibi belirli görevler için program kodlarını yazmaktadır.
- **Yazılım Mühendisleri :** Yazılım mühendisleri, sorunları çözmek için yazılımlar ve sistemler oluşturmak için mühendislik ilkelerini uygular. Yalnızca belirli bir örnek veya müşteriyi çözmek yerine, sorunlara genel olarak uygulanabilecek çözümler geliştirmek için modelleme dili ve diğer araçlar kullanmaktadır.
- **Yazılım Geliştiricileri :** Yazılım geliştiriciler, mühendislerden daha az resmi bir role sahiptir ve kod yazma gibi belirli proje alanlarıyla yakından ilgilenebilirler. Aynı zamanda, gereklilikleri özelliklere dönüştürmek için fonksiyonel ekipler arasında çalışmak, geliştirme ekiplerini ve süreçlerini yönetmek ve yazılım test ve bakımını yürütmek de dahil olmak üzere genel yazılım geliştirme yaşam döngüsünü yönlendirmektedir.

IBM tarafından yapılan tanımlamada sadece sürecin "kod geliştirme" aşamasında yer alan kişiler ve üstlendikleri görevler detaylandırılmıştır. Yazılım geliştirme sürecinde yukarıda belirtilen rollerin haricinde analistler, test uzmanları, proje yöneticileri, iş paydaşları vb.. rollere sahip kişiler de bulunabilmektedir.

2.1.1 Yazılım Geliştirme Yaşam Döngüsü (SDLC)

Yazılım Geliştirme Yaşam Döngüsü, geliştirme sürecinin temel aşamalarını ve faaliyetlerini tanımlayan bir iş akışı sürecidir. Uygulamayı planlamak ve uygulamak, sistemleri veya ürünleri zamanında ve bütçe dahilinde sunmak için sistem analistleri, tasarımcılar ve geliştiriciler tarafından kullanılmaktadır (Tiky, 2016: 4).

Şekil 6. Yazılım Geliştirme Yaşam Döngüsü

Kaynak: Synotive (13 Mart 2019) tarihinde https://www.synotive.com/blog/software-development-client-questionnaire 'den alındı.

Yazılım geliştirme süreçleri uygulamaya, metodolojiye ve uygulama alanına göre farklılık gösterse de tipik olarak aşağıdaki adımları içermektedir (IBM, 2017).

- **Metodolojinin Seçimi :** Bu aşama yazılım geliştirme adımlarının uygulandığı bir süreci oluşturmak için kullanılacak metodolojinin seçilmesini içermektedir. Metodoloji seçimi proje için genel bir iş sürecini veya yol haritasını açıklamaktadır.

- **Gereksinimlerin Toplanması :** Kullanıcılar ve diğer paydaşlar için neyin gerekli olduğunu anlamak ve bu bilgiyi belgelendirme aşamasıdır.
- **Mimari Seçimi veya Geliştirilmesi :** Yazılımın çalışacağı temel yapının belirlenme/oluşturulma aşamasıdır.
- **Tasarım Geliştirme :** İhtiyaçların ortaya koyduğu sorunların çözümü için gerekli olan bu aşamada genellikle süreç modellerini ve durum panolarını içermektedir.
- **Kodlama :** Kodlamanın uygun programlama diliyle yapılmasını ifade eden bu süreç erken sorunları gidermek ve kaliteli yazılımı daha hızlı üretmek için gerekli olan ekip çalışmasını içermektedir.
- **Test :** Yazılım tasarımının ve kodlamanın bir parçası olarak önceden planlanmış senaryolarla uygulama testi ve uygulamadaki yük testini simüle etmek için performans testi aşamalarını içermektedir.
- **Yapılandırma ve Hataları Yönlendirme :** Yazılımın tüm aşamalarını anlamak ve farklı sürümleri oluşturmak için yapılandırma ve hataları yönetme aşamasıdır.
- **Dağıtma (Yayınlama) :** Kullanıcı sorunlarına çözüm bulması ve kullanılması amacıyla yazılımın yayınlanma aşamasıdır.
- **Veri Taşıma :** Gerekli durumlarda verilen mevcut uygulamadan veya veri kaynaklarından yeni veya güncel uygulamaya aktarımını içermektedir.
- **Projeyi Yönetme ve Ölçümleme :** Uygulama yaşam döngüsü boyunca kaliteyi ve teslimatı korumak ve geliştirme sürecini farklı modellerle değerlendirmek için projeyi yönetme ve ölçme aşamasıdır.

Yazılım geliştirme sürecinin adımları SDLC adımlarıyla da uyum içindedir. Bir başka ifadeyle SDLC adımları sürecin sentezlenmiş hali olarak ifade edilebilir (IBM, 2017). Bu adımlar aşağıdaki şekilde sınıflandırılmaktadır. Başlıkların detayları "**2.1.2 Yazılım Geliştirme Yaşam Döngüsü Aşamaları**" bölümünde anlatılmıştır.

- Planlama
- Gereksinim Analizi
- Tasarım
- Yazılım Geliştirme
- Test
- Bakım ve Destek

SDLC, bir yazılımın nasıl tasarlanacağını, geliştirileceğini, korunacağını, değiştirileceğini, geliştirileceğini, test edileceğini ve hatta başlatılacağını açıklayan ayrıntılı adımlar ve etkinliklerden oluşmaktadır (Tiky, 2016: 7).

Yazılım geliştirme süreçleri lineer olduğu gibi döngüsel de olabilmektedir. Sürecin şekli ve uygulama biçimi kullanılacağı alana, projenin büyüklüğüne ve karmaşıklığına göre değişkenlik göstermektedir (Awad, 2005: 35). Bu sebepten yazılım geliştirme süreçlerinde birden fazla yazılım geliştirme yöntemi kullanılmaktadır. Kullanılan yöntemler göz önünde bulundurulduğunda, SDLC, yazılım geliştirme yöntemleri için en genel terimdir. Sürecin adımları ve sıralaması uygulanan metoda göre değişiklik gösterebilir. Yönteme bakılmaksızın süreç her bir yinelemeden başlayarak, döngü halinde çalışmaktadır.

Bir yazılım uygulaması geliştirilirken üretilen ilk sürüm çoğu zaman talep sahibinin isteklerini "tam anlamıyla" karşılamayacaktır. Son kullanıcının deneyimleri ve ihtiyaçları doğrultusunda yeniden tasarlanmayı, geliştirilmeyi ve değiştirilmeyi bekleyen ek özellikler ve hata düzeltmeleri ortaya çıkacaktır. Son kullanıcılar tarafından yapılan bildirimler yeni özellik istekleri ve mevcut özelliklerde iyileştirme talepleri halinde gelmektedir. Tüm yöntemler yaklaşımda geniş ölçüde farklılık gösterebilir ancak ortak bir hedefi paylaşırlar: amaç yazılımı mümkün olduğunca ucuz, verimli ve etkili bir şekilde geliştirmektir.

Yazılım Geliştirme Yaşam Döngüsü Kullanımının Yararları

Küçük veya orta ölçekli bir proje örneği düşünüldüğünde planlama veya tasarım yapılmadan yazılım geliştirmeye başlamak sürecin paydaşlarına (tasarımcı, programcı) cazip gelebilmektedir. Programcılar, planlama için harcanan zamanın, programlama işini yapıp ürünü teslim etmeleri için yeterli olduğunu iddia ederken, yöneticiler "aynı" sonucu elde etmek için verimli çalışmayı ve en az miktarda kaynak kullanmaya odaklanma eğilimindedir.

Yazılım geliştirme süreçlerinde SDLC kullanımının yararları ve neden SDLC kullanılması gerektiği aşağıda maddeler halinde açıklanmıştır (Tiky, 2016: 10).

- **Kalite Güvencesi ve Kalite Kontrolü:** Kalite Güvencesi, ürün geliştirme sürecinde kalitenin sağlanması için bir dizi faaliyettir. Kalite kontrolü ise geliştirme süreci tamamlanmış ürünün kalitesini sağlamak için uygulanan bir dizi faaliyet olarak

tanımlanmaktadır. KG, ürün geliştirme sürecine odaklanarak kusurları önlemeyi amaçlarken, KK, bitmiş ürünü inceleyerek hataları tespit etmeyi amaçlamaktadır.

- **Daha Kolay Uygulama Kontrolü:** SLDC' nin beş temel aşamasında programcıların, test uzmanlarının ve yöneticilerin süreci takip edebilmesi amacıyla kılavuzlar, talimatlar gibi dokümantasyon işleri yapılmaktadır. Yapılan bu işlem sonucunda beklenmeyen alanlarda çıkması muhtemel sorunlar en aza indirgenmekte ve proje takviminin beklendiği ve planlandığı gibi karşılanabileceği öngörülmektedir.

- **Gereksinimlerin Yerine Getirilmesi/Beklentilerin Karşılanması:** Kullanıcı beklentilerinin değişmesi nedeniyle teslim edilen ürününün geliştirmek istenmesi veya teknik yükseltmeler yapılmak istenmesi gibi birçok durumla karşılaşılmaktadır. Böyle durumlarda tasarım aşamasında, tasarımcılar sadece gereksinimler için çözüm vermekle kalmaz, aynı zamanda benzer sistemlerle entegrasyon, esneklik ve geliştirmelerin kullanılabilirliği gibi durumları da dikkate alırlar.

Bir çeşit yapısal plan olmadığı durumlarda yazılım geliştirme ekipleri adeta ne yapacağını bilmeyen bir "kedi sürüsü" haline gelme eğilimindedir. Böyle bir durumda geliştiriciler neyi geliştirmeleri gerektiğini bilmiyorken, proje yöneticileri de, bir projenin tamamlanmasına yönelik ne kadar ilerleme kaydedildiğini bilmemektedir. Bir plan olmadan, işletmenin nihai ürünün gereksinimleri karşılayıp karşılamadığına karar vermesinin bir yolu ise bulunmamaktadır (Raygun, 2019).

Yazılım geliştirme yaşam döngüsü özelliklerini taşıyan bir yöntem yazılım geliştirme süreci için birçok avantaj sağlamaktadır. Bu avantajlar aşağıda maddeler halinde paylaşılmıştır.

- Her adım için ortak bir kelime tanımlanır.
- Geliştirme ekipleri ve paydaşlar arasında tanımlanmış iletişim kanalları bulunur.
- Geliştiriciler, tasarımcılar, iş analistleri ve proje yöneticileri arasındaki rolleri ve sorumluluklar netleştirilir.
- Bir adımdan diğerine açıkca tanımlanmış girdi ve çıktılar tanımlanır.
- Bir adımın gerçekten tamamlanıp tamamlanmadığını doğrulamak için kullanılabilecek deterministik bir "iş tanımı" belirlenir.

2.1.2 Yazılım Geliştirme Yaşam Döngüsü Aşamaları

Yazılım geliştirme süreci belirli adımlardan oluşmaktadır. Tercih edilen yönteme göre bu adımlar sıralı ya da döngüsel olabildiği gibi aynı anda paralel işleyen süreçlerde tercih edilmektedir. YGYD adımları aşağıda görüldüğü gibi altı aşamada ele alınabilir.

- Planlama
- Gereksinim Analizi
- Tasarım
- Yazılım Geliştirme
- Test
- Bakım ve Destek

Çevik yöntemler, bu adımların tümünü sıkı ve hızlı bir şekilde yinelenen bir döngüyle bir araya getirirken, Şelale yöntemi ise bu adımların her birini sırasıyla ele alma eğilimindedir. Şelale yönteinde bir adım sonlandığında elde edilen çıktı, bir sonraki adıma girdi olacaktır.

Planlama genellikle bir grup son kullanıcı, bir ihtiyaç veya fırsatı tanımlayan sponsordan gelen bir inovasyon girişimi veya inovasyonun ardından gerçekleşir. Planlama aşamasında, kavramların kapsamı veya sınırı tanımlanmaktadır. Finansal, operasyonel ve teknik alanlarda ürün fizibilite çalışması, iş paydaşlarının katkısı ile ekibin kıdemli üyeleri tarafından yapılmaktadır. Öngörülemeyen riskleri en aza indirmek için planlama aşamasında Kalite Güvencesi ve Risk Yönetimi planı da hazırlanır (Tiky, 2016: 8).

Planlama aşaması, aşağıda belirtilen proje ve ürün yönetimi konularını içermektedir (Raygun, 2019).

- Kaynak Tahsisi (hem insan hem de malzeme)
- Kapasite Planlaması
- Proje Takvim Planlanması
- Maliyet Tahmini
- Karşılama

Planlama aşaması sonrasında elde edilen çıktılar aşağıda paylaşılmıştır (Raygun, 2019).

- Proje Planı ve Programı
- Maliyet Tahmini
- Tedarik Gereksinimleri
- Gereksinim Analizi

Yazılım geliştirme süreci başladıktan sonra iş paydaşları geliştirme taleplerini ve gereksinimlerini iletmek için BT ekipleriyle iletişim halinde olmalıdır. Gereksinimlerin toplanma aşamasında, gerekli bilgiler iş paydaşlarından ve konunun mesleki uzmanlarından elde edilmektedir. Yazılım Mimarları, Geliştirme Ekipleri ve Ürün Yöneticileri, yazılımla otomatikleştirilmesi gereken iş süreçlerini belgelemek için mesleki anlamda uzman kişilerle birlikte çalışır. Metodoloji olarak Şelale yönteminin tercih edildiği bir projede bu aşamanın çıktısı genellikle gereksinimlerin listelendiği bir doküman olmaktadır. Çevik yöntemler ise bu durumun aksine gerçekleştirilecek görevler listesini üretmektedir.

Tasarım aşamasında; gereksinimler anlaşıldıktan sonra, yazılım mimarı ve yazılım mühendisleri uygulamayı tasarlamaya başlamaktadır. Tasarım sürecinde uygulama mimarisi ve yazılım geliştirme için belirlenmiş kalıplar kullanılmaktadır. Yazılım Mimarı, mevcut bileşenlerden bir uygulama oluşturmayı, yeniden kullanımı ve standardizasyonu teşvik etmek için TOGAF gibi bir mimari çerçeveyi kullanabilmektedir.

Tasarım aşamasının iki adımı bulunmaktadır.

- **Üst Düzey Tasarım :** Yazılım mimarları ve tecrübeli yazılım mühendisleri tarafından geliştirilecek yazılım ürününün mimarisi paylaşılmaktadır (Software Testing Material, 2019).

- **Düşük Seviyeli Tasarım :** Tecrübeli yazılım mühendisleri tarafından gerçekleştirilir. Üründeki her bir özelliğin ve her bileşenin nasıl çalışması gerektiği açıklanmaktadır (Software Testing Material, 2019).

En iyi veya en uygun tasarım seçildikten sonra, yazılım geliştirme aşaması hemen başlamaktadır. Programcılar, yazılımı tasarım aşamasında dokümante edilen şekilde geliştirmeli ve aynı zamanda şirketin tanımladığı kodlama standartlarını yakından takip etmelidir. Programlama araçları, tüm programcıların çalışmalarını düzenleyebilmeleri için belirli standartlara sahip olmalıdır. Geliştirme ile alakalı Teknik Analiz yazılacaksa teknik fonksiyonları içeren bir doküman olmalı ve kıdemli programcılar tarafından denetlenmelidir (Tiky, 2016: 8).

Bu fazda yapılan geliştirme sonucunda hedeflenen yazılım üretilmekte, kullanılan metodolojiye bağlı olarak bu aşamada farklılıklar gözlenebilmektedir. Bu aşamadan çıkan sonuç, Kaynak Kod Dokümanı (SCD) ve geliştirilen üründür (Software Testing Material, 2019).

Test süreci bu döngünün en önemli aşamalarından birisidir. Bu aşamada geliştirilen yazılımın beklentileri ve gereksinimleri karşılama durumu test edilmektedir (Barjtya vd., 2017: 2). Yazılım geliştirme sürecinde yapılan testin kalitesi geliştirilen ürünün kalitesini göstermektedir. Bu süreçte testlerin düzenli olarak yapılması, geliştirilen yazılımın kontrol edilmesi, bu kontrollerin asla atlanmaması sürecin devamlılığı açısından büyük önem arz etmektedir. Geliştirmenin üretim ortamına atılmadan detaylı bir şekilde test edilmesi ileride yaşanacak olası problemleri gerçekleşmeden engellemiş olacaktır. Kaliteyi ölçmek için çeşitli testler yapılmaktadır (Raygun, 2019).

- **Kod Testi:** Yazılımın geliştirilmesi aşamasında ilgili geliştirici tarafından yapılmaktadır. Bu test geliştirmesi aşamasında yaşanan problemlerin erken aşamada fark edilmesi açısından önemlidir.
- **Birim (Fonksiyonel) Testi:** Geliştirilen süreç ile ilgili iş bilgisine sahip analist veya test uzmanı rolüne sahip kişiler tarafından yapılmaktadır.
- **Entegrasyon Testi: U**ygulamaların birbiriyle bütünleşmiş şekilde çalıştığı durumda yapılan değişikliğin başka uygulamalara etkisini ölçmek amacıyla yapılmaktadır.
- **Performans Testi:** Yapılan geliştirmenin sistem performansına etkisini ölçmektedir.
- **Güvenlik Testleri:** Yapılan geliştirmenin sonrasında güvenlik açıklarını tespit etmek amacıyla yapılmaktadır.

Test aşamasının çıktısı, üretim ortamına iletilmeyi bekleyen işlevsel bir yazılımdır. Yapılan geliştirmeye göre yukarıda ifade edilen testlerin hangilerinin yapılacağı ilgili analist ve yazılım mimarı tarafından belirlenmektedir.

Bakım aşaması yeni bir sürecin başlaması için gerekli son aşamadır. Süreç burada sonlanmamakta, bu döngü yaşayan bir süreci ifade etmektedir. Bu sebepten yazılımın düzgün bir şekilde çalışabilmesi için sürekli takip edilmesi gerekmektedir. Üretim ortamında tespit edilen hatalar ve bulgular rapor edilmeli ve ilgili rapor doğrultusunda gerekli düzenlemeler periyodik olarak yapılmalıdır. Yapılan bakım işlemi çoğu zaman süreci yeniden beslemektedir.

Hata düzeltmelerinde süreç tüm döngüyü kapsamayacak şekilde yürütülebilir. Yapılacak düzeltmenin başka sorunlara neden olmaması için kısaltılmış bir işlem gereklidir. Bu işlem "regresyon" olarak bilinmektedir. Regresyon testi; "daha önce geliştirilen ve test edilen yazılımın bir değişiklikten sonra da çalışabilir kalmasını sağlamak için fonksiyonel ve fonksiyonel olmayan testlerin gerçekleştirilmesi" anlamına gelmektedir (Raygun, 2019).

2.1.3 Popüler Modeller ve Uygulamaları

Günümüz yazılım geliştirme dünyasında en çok bilinen ve en çok kullanılan iki uygulama "Şelale" ve "Çevik" yöntem olarak ifade edilmektedir (Awad, 2005: 35). Bu yöntemler aynı felsefe üzerinden geliştirilmiş olsa da, bu felsefeyi ele alma yöntemlerinde farklılıklar gözlenmektedir.

Şelale Yöntemi

Şelale ya da literatürde bilinen adıyla "Waterfall" yazılım geliştirme yöntemi, önceden belirlenmiş bir dizi işlem boyunca ilerleyen aşamalardan oluşmaktadır. Bu yöntem geleneksel mühendislikten uyarlanmıştır. İronik olarak, yöntemin kaynağı olarak gösterilen dokümantasyon, uygulamanın temelinde kusurlu olduğunu belirtmektedir. Bugün "Waterfall" olarak adlandırılan bu yöntem orjinal çalışmanın yanlış anlaşılmasından yanlışlıkla türetilmiştir. Buna rağmen, dünya çapında büyük projeler için yaygın, hatta standart bir yöntem haline gelmiştir (Benington, 1983: 37).

Şelale yöntemi uzun planlama ve tasarım aşamasıyla başlamaktadır. Geliştirme tamamlandıktan sonra, yazılım gerekli test aşamalarından geçmekte son olarak üretim ortamına aktarılmaya hazırlanmaktadır. Bu yöntem günümüzde birçok kişi tarafından değişen gereksinimlere uyum sağlayamayacak kadar katı olduğu düşünülmektedir (Fair, 2012: 3). Süreç boyunca yapılan geri bildirimler desteklenmemekte, geliştirme sırasında değişebilecek ihtiyaçların uygulanmasında sorunlar yaşanmaktadır. Bu zayıflık, Çevik ya da literatürde bilinen adıyla "Agile" gibi daha esnek yöntemlerin geliştirilmesine de yol açmıştır.

Çevik Yöntem

"Agile Manifesto", 2001 yılında bir grup yazılım geliştirici tarafından hazırlanmış ve imzalanmıştır (Fowler ve Highsmith, 2001). Manifestonun bir kısmında Şelale yöntemiyle alakalı yazılım tesliminde zorluklara yol açan temel sorunlar ele alınmaktadır. Şelale yöntemi "tek yönlü" olma eğilimindeyken, Çevik yöntemin buna karşın belirsizlik içeren daha esnek bir süreci ifade ettiği belirtilmektedir. Çevik yöntem, değişen gereksinimlere yanıt olarak ekip çalışması, prototip oluşturma ve geri bildirim döngüleri gibi kavramları vurgulamaktadır. Manifesto detaylı bir şekilde incelendiğinde, Şelale ve Çevik yöntemlerin arasındaki fark görülmektedir.

Manifesto'nun imzalanmasından bu yana geçen zaman içinde birden fazla yöntem ortaya çıkmıştır. Bu yöntemler arasında en çok tercih edilen Scrum, kendine özgü belirli rolleri ve etkinlikleri uygulamanın bir parçası olarak tanımlamaktadır (Schwaber ve Sutherland, 2017: 3). Yazılım ekipleri genellikle kendilerine en uygun süreci uyarlamak için yöntemleri incelemekte ve seçimlerini bu sonuçlara göre yapmaktadır.

Yazılım Geliştirme Yaşam Döngüsü Uygulamaları

Şelale yöntemi dünyanın dört bir yanında birçok şirket tarafından kullanılmakta, Çevik yöntem ise bilinirliğinin artmasıyla yazılım geliştirme sektöründe payını artırmaktadır. En çok tercih edilen bu iki yöntem uygulama açısından birbirine taban tabana zıt gözükse de uygulama noktasında yazılım geliştirme yaşam döngüsünün felsefesine uyumlu şekilde hareket etmektedir.

Yazılım geliştirme süreçleri temelinde pragmatik bir yaklaşıma bağlıdır. Geliştirilen uygulama yaşayan bir varlık olarak düşünüldüğünde, bu varlığın yaşantısını devam ettirmesi için sürekli yenilenmesi, hataların düzeltilmesi ve geliştirilmesi gerekecektir. Bu kapsamda yapılan uygulamaların bazı temel özelliklere sahip olması beklenmektedir (Raygun, 2019). Bu özellikler ve detayları aşağıda maddeler halinde paylaşılmıştır.

- Kaynak Kontrolü
- Sürekli Entegrasyon
- Sistem Yönetimi

Kaynak Kontrolü

Yapılan geliştirmelerin sürekli olduğu düşünülürse kaynaklar en verimli olacak şekilde kullanılmalıdır.

Sürekli Entegrasyon

Amacı yazılımı işlevsel bir durumda tutmaktır. Yapılan geliştirmelerin mevcut sistemi en az etkileyecek şekilde tasarlanması gerekmektedir.

Sistem Yönetimi

Büyük ve karmaşık sistemlerde uygulama geliştirmekten çok bunların yönetimi ayrı bir önem arz etmektedir. Yazılım geliştirme süreçlerinde bu etken de göz ardı edilmemelidir.

2.2. Geleneksel Yazılım Geliştirme Yöntemleri

Tezin bu bölümünde pek çok kurum ve projede kullanılan geleneksel yazılım geliştirme yöntemleri incelenecektir.

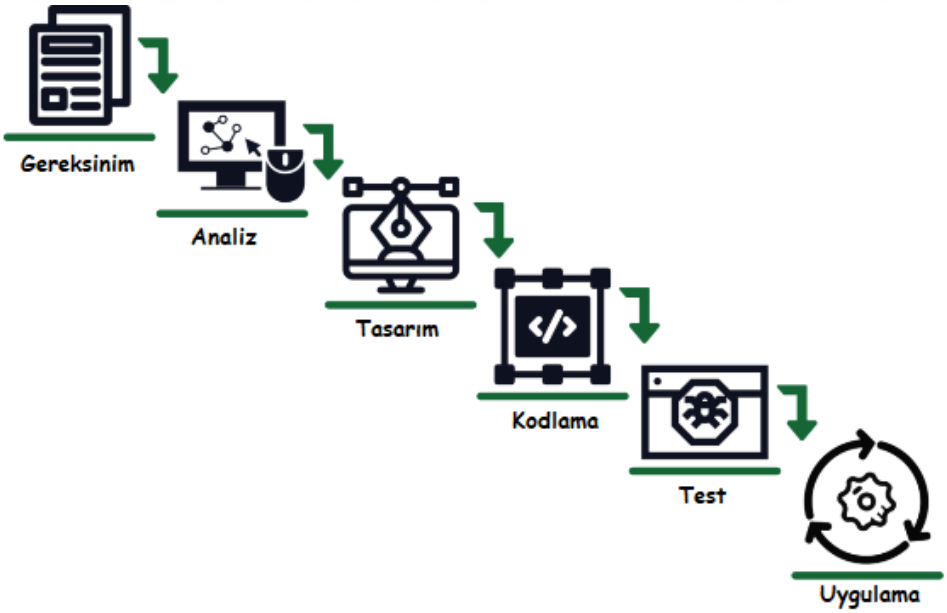
Belirli işlevi olan bir ürün veya bir sistem geliştirildiğinde, işlemlerin kontrollü bir şekilde gerçekleştirileceği bir dizi öngörülebilir adım olması önemlidir. Bu adımlar kimin, neyi, ne zaman ve nasıl yapacağı dâhil olmak üzere gerçekleştirilecek çeşitli görevleri belirtmektedir. Yazılım geliştirme faaliyetleri için bu özellikler "geliştirme sürecini" tanımlamaktadır. Geleneksel yöntemler, yazılım geliştirme tarihinde kullanılan ilk yöntem olarak kabul edilmektedir. Bu yöntemler gereksinimlerin tanımlanması, çözüm geliştirilmesi, test ve uygulama gibi sıralı bir dizi adıma dayanmaktadır. Bir projenin başlangıcında sabit bir

gereksinim listesi tanımlanması ve belgelenmesi gerekmektedir (Awad, 2005: 35). Geleneksel yöntemler, tüm yöntemler gibi YGYD sürecini temel almakta, bu süreç sırasıyla planlama, tasarım, yazılım geliştirme, test ve bakım aşamalarından oluşmaktadır.

Gerçek zamanlı yazılım geliştirme projeleri bu ideal aktivite sırasını nadiren takip etmektedir. Yazılım uzmanları tarafından insan çabasıyla gerçekleştirilen yukarıdaki faaliyetlerin her biri, daha sonraki bir zamanda tespit edilen insan hatalarından etkilenebilir. Ayrıca müşterilerin gereksinimleri, proje şekillendikçe değişmeye devam etmektedir. Tüm bunlar, yazılım geliştirme projesindeki ilerlemeyi, çok sayıda geri bildirim ve hata düzeltmesiyle birlikte tek yönlü olmaktan uzak tutmaktadır (Schach, 2007: 53).

2.2.1. Şelale Yöntemi

Yazılım endüstrisinin 1950-1960 yılları arasında ortaya çıkması ve zamanla gelişmesiyle birlikte büyük ölçekli yazılım projelerini daha iyi tahmin edip kontrol edebilme ihtiyacı, geleneksel proje yönetimi yönteminin yazılım geliştirmeye adapte olmasına neden olmuştur (Leffingwell, 2011: 45). Yazılım geliştirme sürecinde safhaların sıralı ve aşamalı bir şekilde kullanıldığı yöntem "Şelale" (Waterfall) olarak adlandırılmaktadır.



Şekil 7. Şelale Yönteminin Temel Adımları

Kaynak: Rezaid (22 Mart 2019) tarihinde https://rezaid.co.uk/sdlc-waterfall-model/ 'den alındı.

Şelale yönteminin icat edilmesiyle tanınan Winston W. Royce, aslında bunu büyük ölçekli yazılım geliştirme projeleri için işe yaramayacak bir model olarak tanımlamıştır. Royce, yöntemin uygulanmasının, anlatıldığı gibi gerçekleştiğini bu sebepten dolayı projeleri "riskli ve başarısız olmaya davet ettiğini" işaret etmektedir (Royce, 1987: 60).

1970 yılında Winston W. Royce tarafından resmi tanımı yapılmış olmasına rağmen 1976' da Bell ve Thayer, bu yöntemi "Şelale" yöntemi olarak adlandırmıştır (Bell ve Thayer, 1976: 62). Şelale yöntemi bir olaylar dizisidir; uzmanların bazıları özel bir tarifle onları aşamalar olarak adlandırmaktadır. Bu yöntemde bir aşama bittikten sonra bir diğeri başlayabilir. Royce'a göre, yöntemin aşamaları: gereksinim tespiti, analiz, tasarım, kodlama, testler ve kurulum, destek ve bakım gibi işlemlerdir (Royce, 1987: 60).

Şelale modeli, herhangi bir yazılım kuruluşu tarafından anlaşılması, benimsenmesi ve uygulanması kolay, yapılandırılmış bir yaklaşım sunmaktadır. Ayrıca, yazılım geliştirme sürecinde kolayca tanımlanabilen "kilometre taşları" sağlamaktadır. Bu nedenle şelale modeli birçok kitap ve derste yazılım geliştirme yönteminin başlangıç örneği olarak kullanılmıştır. Şelale modeli, önceki aşamaların gözden geçirilmesini engelleyen geleneksel ve esnek olmayan bir mühendislik yaklaşımıdır (Rezaid, 2019).

Şelale yöntemi, bazı zamanlarda büyük ölçekli yazılım projelerindeki başarısızlıklardan (bütçe ve zaman aşımı) dolayı suçlanmaktadır. Suçlamanın nedeni müşterilerin çalışan yazılımı görmeden önce gereksinimlerinin tam olarak ne olduğunu bilmemeleri olabilmektedir. Böyle bir durumda müşteri ihtiyaçlarına göre yazılımın değişmesi ve testlerin yeniden yapılması gerektiğinden süreç başarısızlıkla sonuçlanmaktadır (Parnas, 1987: 25). Dahası, tasarımcılar projeyi tasarlarken geliştirme aşamasındaki zorlukları göremeyebilirler. Bu durum gereksinimlerin gözden geçirilmesine ve bazı sorunlara yol açabilmektedir. (McConnel, 2004: 412)

Şelale Yönteminin Sınırlamaları

Şelale yöntemi süreç geliştiriciler tarafından ortaya atılan ilk model olsa ve onlarca yıl boyunca geliştiriciler tarafından bir referans model olarak takip edilse de, aşağıdaki sorunlardan kaynaklanan eksiklikleri bulunmaktadır.

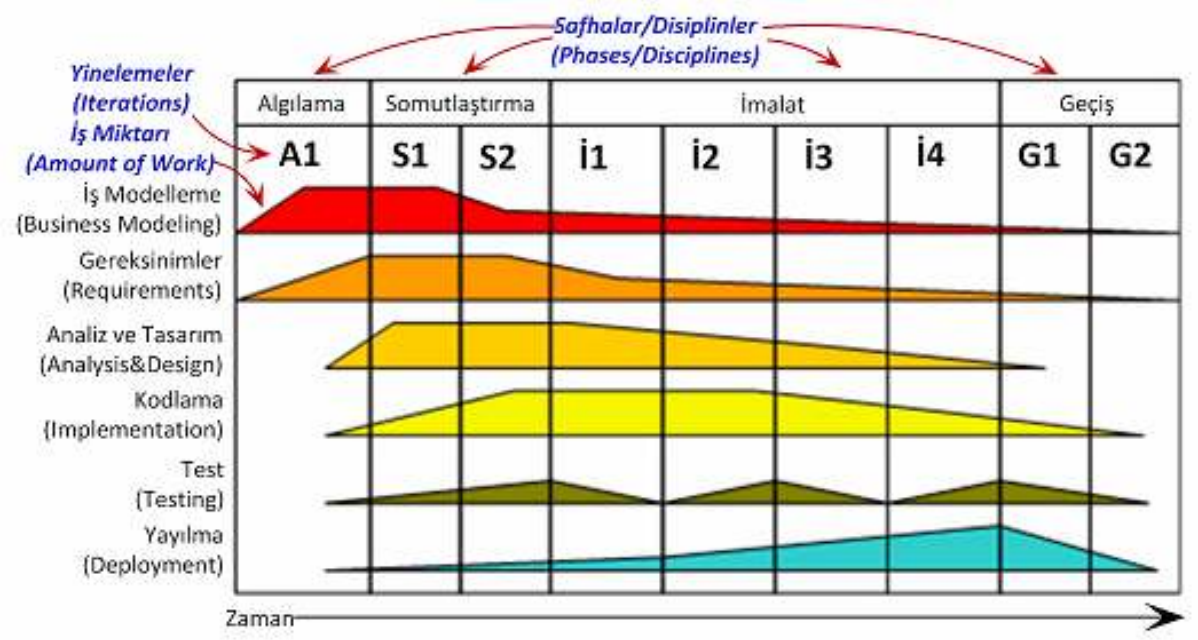
- Teorinin aksine projelerde geliştirme faaliyetine başlamadan önce tüm gereksinimleri eksiksiz bir şekilde elde etmeye çalışmak zor ve meşakkatli bir süreçtir. Müşterilerin/son kullanıcıların kafasında birçok gereksinimin belirsizliği ve esneklik seviyesinin düşük tutulması, proje ilerledikçe ve yazılım geliştiricileriyle etkileşime geçildiğinde ortadan kalkmaktadır (Boehm ve Jain, 2006: 3-4). Bu sebepten dolayı modelin değişen gereksinimleri karşılayacak bir donanıma sahip olmadığı düşünülmektedir.
- Geliştirilen yazılımın çalışan sürümü yalnızca proje sonunda müşteri temsilcileri tarafından değerlendirilebildiğinden, müşterilerin gelişim sürecinde sabırla beklemeleri gerekmektedir. Bu durum özellikle geliştiriciler ve müşteriler arasındaki iletişim sırasında, projenin erken aşamalarında gözden kaçan yanlış anlamaları veya hataları düzeltmek için erken müşteri geri bildirimleri ve müdahalenin olanaklarını engellemektedir (Boehm ve Jain, 2006: 6).

2.2.2. Birleşik Rasyonal İşlem

1980-1990 yılları arasında geliştirilen yazılım ürünlerinin kapsamı ve büyüklüğü hızlı bir şekilde artmış, bunun sonucunda nesne yönelimli gerçek sistemleri temsil etme ve yazılımı nesne yönelimli dilleri kullanarak geliştirme kavramı ön plana çıkmıştır. Gerçek hayatta ilgilenilen nesneler ve davranışlarıyla ilişkili veriler bağımsız varlıklar oluşturmak üzere gruplandırılmıştır. Aynı yapıya ve davranışa sahip bu nesnelerin sınıflanması uygulamadan uygulamaya çok az değişim gösterdiği ve dolayısıyla kararlı olduğu saptanmıştır. Nesnelerin ve bunların davranışlarının bu şekilde temsil edilmesiyle sağlanan yazılım yeniden kullanımı, büyük yazılım geliştirme projeleri yürütürken yazılım geliştiricilerin verimliliğini artırmıştır. Pek çok nesne yönelimli yazılım geliştirme metodolojisi önerilmiş olmasına rağmen, bunlar arasında en popüler üç yöntem bulunmaktadır (Loomis vd., 2000: 85).

Bunlar; Obje Modelleme Tekniği (OMT) (Loomis vd., 2000: 85), Grady Booch Metodu (Booch, 2006: 45) ve Objectory Metodu (Jacobson, 1992: 65) olarak ifade edilmektedir.

1997 yılında yukarıda ifade edilen yöntemlerin üç yazarı Rational Corporation'ın yöneticileri olarak bir araya gelmesiyle nesne yönelimli sistemlerin modellemesi ve geliştirilmesi için sağlam bir gösterim sağlayan "Birleşik Modelleme Dili" (UML) geliştirilmiş ve bu şekilde yöntemler arasındaki farklar çözülmüştür. Nesne yönelimli yazılım üretiminde dünyanın önde gelen şirketlerinin bir derneği olan Nesne Yönetim Grubu (OMG), UML'yi, bir kuruluştaki geliştiriciler arasında ve dünya çapındaki şirketler ile etkin iletişimi teşvik eden nesne yönelimli yazılım ürünlerini temsil etmek için uluslararası bir standart haline getirmek için girişimlerde bulunmuştur (Booch, 1999: 78).



Şekil 8. Birleşik Rasyonal İşlem

Kaynak: Modern Yazılım Geliştirme (22 Mart 2019) tarihinde https://sites.google.com/site/modernyazilimgelistirme/yazilim-gelistirme-sureci/yazilim-gelistirme-sureci-modelleri/usdp-modeli 'den alındı.

Üç nesne yönelimli yöntemin uyumlu ve arzu edilen özellikleri, "Birleşik Yazılım Geliştirme Süreci" veya "Birleşik Rasyonal İşlem (RUP)" oluşturmak için yaratıcı bir şekilde birleştirilmiştir (Bugünlerde yalnızca "Birleşik İşlem" olarak adlandırılmaktadır.) (Jacobson,

1999: 65). Birleşik İşlem (UP) yazılım geliştirme sürecinde beş ayrı aktivite tanımlamakta ve bunları temel iş akışları olarak adlandırmaktadır.

- Gereksinim
- Analiz
- Tasarım
- Uygulama
- Test

Her iş akışı, büyük projelerde birkaç adım veya artışla gerçekleştirilmektedir. İhtiyaç analizi ve yazılım tasarımı sırasında, geliştirme ekibi elde edilen soyut modeli UML şemaları biçiminde ortaya koymaktadır. Her iş akışının artımlı doğası ve model formülasyonuna katkıda bulunan çoklu ekipler nedeniyle, bileşen parçaları arasındaki bağlantıyı gösteren genel model, her iş akış süreci ve ekip üyeleri arasında yapılan tartışmaların ardından gelişmeye devam edecektir (Booch, 1999: 79-80).

Deneyimli yazılım uzmanlarının çoğunda, ekibin tüm üyeleri modelin doğruluğu konusunda hemfikir olana kadar geliştirme adımları birkaç kez tekrarlanacaktır. Başka bir deyişle, geliştirme süreci doğada yinelemelidir; her yinelemenin ürünü öncekine göre bir gelişmedir. Uygulamada bu artış ve yinelemenin büyük yazılım projelerinde kaçınılmaz olduğu anlaşılmıştır. Modelin hassaslığı ve doğruluğu her iterasyonda geliştirilirken model kapsamı da her artış ile uzatılmaktadır. Böyle bir süreç, geliştiricilerin tek seferde birden fazla bilgi birimini anlama noktasındaki başarısız girişimlerden kaçınarak bilgiyi bir anda kullanmalarını sağlamaktadır (Booch, 1999: 79-80).

Farklı iş akışı ana hatları sabit bir şekilde ilerlememektedir. Analiz iş akışının başlaması için gereksinimler iş akışının tamamlanması beklenmemektedir. Son kullanıcılar ile iletişimde daha fazla gereksinim ortaya çıksa bile, hali hazırda anlaşılan gereksinimler için analiz modeli oluşturma faaliyeti başlatılmaktadır. Benzer şekilde, tasarım ve uygulama faaliyetleri de analiz iş akışıyla önemli ölçüde örtüşecektir. Aynı şekilde, diğer kısımların tasarım ve uygulama aşamalarında olmasına rağmen, uygulanmış yazılım parçalarının birim testi devam edecektir (Booch, 1999: 79-80).

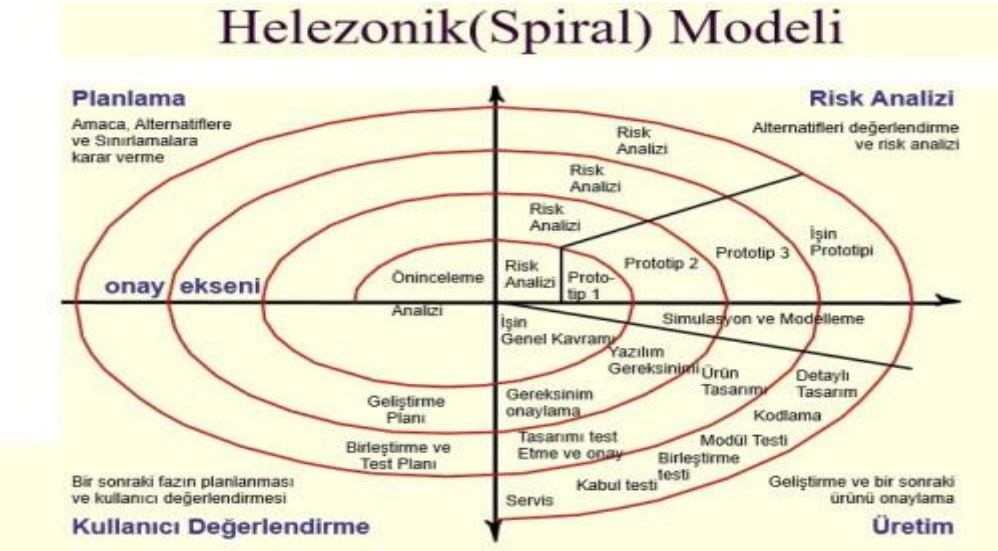
Tüm süreç modeli incelendiğinde, RUP ya da UP için; somut olarak tanımlanan tek bir süreç olmadığı, daha çok sürecin ihtiyaçları için uygun olan unsurları seçecek olan

geliştirme organizasyonu ve yazılım proje ekipleri tarafından tasarlanması amaçlanan uyarlanabilir bir süreç olduğu ifade edilebilir.

2.2.3. Spiral Model

Spiral yaşam döngüsü modeli ilk olarak Barry W. Boehm tarafından tüm müşteri gereksinimlerini karşılayan yazılıma ulaşmak için bir dizi evrim prototipi kullanılarak önerilmiştir. Büyük yazılım projelerinde, ilk aşamalarda yazılım kapsamı, çözüm yöntemi ve önerilen yazılım ürününün beklenen performansı konusunda belirsizlikler bulunmaktadır. Bu belirsizliklerden dolayı proje riskleri, kapsam belirleme, risk analizi, prototip yapımı, geliştirme ve geri bildirimi içeren her bir faaliyet turunda döngüsel bir yaklaşımla yönetilmektedir.

Her döngü, Şekil 9 'da gösterildiği gibi riskte önemli bir azalma ile ürünün daha iyi tanımlanmasına yol açmaktadır. Her turdan sonra, paydaşlar durumu gözden geçirip bir sonraki faaliyetler için taahhütte bulundukları bir bağlantı noktasına ulaşmaktadır. Başlangıçta yer alan proje riski çok yüksektir ve temkinli bir yaklaşım önerilen yöntemin uygulanabilirliğini gösterecek olan riskli kısımlar için bir "kavram kanıt prototipi" oluşturulmasını içermektedir. Hatalı problem çözme veya yazılım geliştirme yöntemlerinden kaynaklanan risklere ek olarak, riskler gereksinimlerdeki büyük değişikliklerden de kaynaklanabilmektedir (Boehm, 1988: 7).



Şekil 9. Spiral Model

Kaynak: System Analize (25 Mart 2019) tarihinde https://systemanalize.wordpress.com/gelistirme-metotolari/ 'dan alındı.

İlk örnek, performans yeterliliği ve çözüm yönteminin doğruluğunu onaylamak için değerlendirilmektedir. Kapsamın değiştirilmesi ve/veya kapsamının arttırılması için öneriler kullanıcı temsilcileri tarafından istendiğinden, geliştiriciler bu döngüde kullanılan yaklaşımın etkinliğini veya eksikliklerini fark etmekte, iyileştirilmiş planla ve yeniden değerlendirilmiş bir riskle, bir sonraki turda iyileştirilmiş prototip üretilmektedir (Boehm, 1988: 10).

İkinci örnek, yazılım gereksinimlerinin çoğunun doğrulanmasına yardımcı olacak, gereklilikleri ortadan kaldıracak ve geliştiricilerin bir sonraki faaliyet döngüsünü, düşük belirsizlik veya düşük riskle ve mevcut alternatifler arasında daha iyi yetenekler ile başlatabilmelerini sağlayacaktır (Boehm, 1988: 11-12).

Üçüncü turda yapılan geliştirilmiş prototip, yazılımın mimari tasarımını, ilgili doğrulama ve onaylama faaliyetlerini birlikte yapacak kadar olgunlaşmıştır. Ürünün müşteri tarafından kabul edilmeden önceki son testleri de bu aşamada planlanmaktadır. Şimdi operasyonel prototip tarafından öğrenilen tüm derslerle, nihai ürün tasarımı gerçekleştirilir, kodlanır ve test edilir. Bu ürünün müşteriye teslim edilmesinden sonra bile, hata düzeltme ve kapsam geliştirme gibi bakım faaliyetleri, spiral modelde bir başka geliştirme etkinliği olarak gerçekleştirilmektedir (Boehm, 1988: 13-14).

Spiral modeli, çeşitli yazılım ürünleri durumunda proje verimliliğini arttırdığı ifade edilmiştir. Yazılım geliştirme sürecinde mevcut esnekliklerle ilgili riskleri en aza indirmek için kullanılmaktadır. Bu model, tüm paydaşların tek bir kuruma ait olduğu ve nihai ürün kalitesine eşit şekilde bağlı kaldığı büyük şirket içi gelişim projeleri için uygun olduğu tespit edilmiştir. Spiral modelin önemli bir zayıflığı, gelecek döngü riskini ve tasarımcıların hatalarla sonuçlanabilecek gerçek uygulama olmadan yaptıkları varsayımları belirlemektir (Boehm, 1988: 15-16).

Spiral modeli incelendiğinde, her tur sonunda bir prototip üretildiği, bu prototip üzerinden geliştirilen ürünle ilgili faaliyetlerin gerçekleştirildiği, bir sonraki turda edinilmiş kazanımlarla birlikte olası kapsam değişikliklerinin eklenmesiyle birlikte yeni prototiplerin üretilmesi ve bunun sonucunda nihai ürüne başarılı bir şekilde ulaşıldığı görülmektedir. Ayrıca önemli bir nokta olarak her turda "Planlama, Risk Analizi, Geliştirme ve Kullanıcı Değerlendirme" işlemleri aksatılmadan yapılmaktadır.

2.3. Çevik Yazılım Geliştirme Yöntemleri

1970 - 1980 ‘ li yıllarda kalite bilincine sahip yazılım geliştirme çabaları, özellikle büyük, uzun ömürlü uygulamalar için geniş bir dokümantasyon hacminin yanı sıra titizlikle yapılan geliştirme süreçleriyle sonuçlanmıştır. 10 yıldan fazla bir süredir gelişen ve kapsamlı testlere tabi tutulan bir uçak kontrol sistemi, ürünün kurulum ve işletmeye alma sonrası birkaç on yıl boyunca istenen hedeflere ulaşması gerektiği için böyle bir yaklaşımı haklı gösterebilir. İş süreçlerindeki değişiklikler ve bunun sonucunda ortaya çıkan yazılım gereksinimleri önceden tahminlere karşı çıktığında veya kullanılan geliştirme kaynakları ve teknolojideki değişiklikler oldukça hızlı olduğunda yazılım geliştirme için hızlı uyarlanabilir süreçlere ihtiyaç bulunmaktadır. Mevcut literatür, plan odaklı yaklaşım kullanılarak başarılı bir şekilde geliştirilen sistemler olduğunu, hem plan odaklı hem de çevik yaklaşımın esnek ve hafif bir yaklaşım olduğu ve tüm görev süresi boyunca gereksinimlerdeki değişikliklere yer sağlayan durumlar olduğunu göstermektedir. (Boehm ve Turner, 2003: 16).

1990’ların sonunda, birkaç geliştirici grubu birkaç basit model için lider yaklaşımları olan hızlı anlamındaki “çevik” yazılım geliştirme modelini önermiştir (Larman ve Basili 2003, 54). "Çevik" terimi esneklik, harekete hazır olma, etkinlik, hareket halindeki el becerisi ve ayarlanabilirlik kavramlarıyla tanımlanabilir (Oxford, 2016: 21).

Bu önerilerdeki ortak özellikler, Fowler ve diğer 16 geliştirici tarafından “Agile Manifesto” yayınladığı zaman daha iyi tanınır hale gemiştir (Fowler ve Highsmith, 2001). Agile Manifesto incelendiğinde, Çevik yazılım geliştirme yöntemlerinin, geleneksel yazılım geliştirme yöntemlerinde belirlenmiş zayıflıkların üstesinden gelmesi için geliştirildiği görülmektedir. Geleneksel yaklaşımda tespit edilen zayıflıklar çevik yaklaşımda proje faaliyetlerinde değişikliklerin yinelemeli ve artan gelişim yoluyla değişmesine izin verilerek, küçük artımlı teslimatlara odaklanarak, müşterinin katılımını teşvik ederek ve müşteriden sürekli geri bildirim almak suretiyle düzeltilmiştir. Çevik yazılım geliştirme, sade işlemler süreci olarak tanımlanmaktadır. Sade bir süreçle birlikte gerekli belgeleri en aza indirildiği ve titiz bir standardizasyonun gerektiği söylenebilmektedir.

Bu çevik yaklaşım, etkin kalite güvencesini sağlamak için geleneksel geliştirme modelleri gibi titiz dokümantasyon ile tanımlayıcı değildir. Öte yandan bir dizi artışla sunulan, bir yazılım ürünüyle sonuçlanan, son kullanıcılarla yakın işbirliği yaparak ihtiyaç duyulan

yazılımı hızlı bir şekilde gerçekleştirmek için kurallar koyulmuştur. Bu yöntemler, yazılım geliştiricilerin kırılganlıklarını gözlemler, önceki disiplinli yaklaşımdan gelen tolerans veya sapmalara olan ihtiyacı tespit etmektedir. Birçok kişi tarafından çevik ve plan odaklı yöntemlerin karıştırılabildiği hissedilirken, uzmanlar yöntemlerin karıştırıldığında tehlikeye yatkın olduğunu söylemektedir (Awad, 2005: 16-18).

Bu bölümde çevik yazılım geliştirme yöntemlerinin arasında en çok tercih edilen aşağıdaki yöntemler incelenecektir.

- Ekstrem Programlama (XP)
- SCRUM
- Özellik Odakları Geliştirme (FDD)
- Dinamik Sistem Geliştirme Metodolojisi (DSDM)
- Yalın Sistem Geliştirme (LSD)

2.3.1. Çevik Metodolojilerin Oluşumu

2001 yılında Agile Alliance tarafından, insanlar ve proje süreçleri hakkında temel değerleri vurgulayan, daha iyi yazılım geliştirmenin yolunu belirten ilke ve uygulamaları içeren bir manifesto yayınlanmıştır. Konunun uzmanları çevikliğin müşteri ve olay odaklı olduğunu düşünmektedir. Bu değerler çevik yöntemin özünü oluşturmaktadır (Agile Alliance, 2001). Çevik geliştirme modelini takip ederken üzerinde durulan noktalardaki değişim Tablo 2'de özetlenmiştir.

Tablo 2 : Çevik ve Geleneksel Geliştirme Modellerinde Önemli Noktalar

Çevik Geliştirme Modeli	Geleneksel Geliştirme Modeli
Bireyler ve Etkileşimler	Süreçler ve Araçlar
Çalışan Yazılım Teslimi	Kapsamlı Belgeler
Müşteri İşbirliği	Sözleşme Müzakeresi
Değişime Cevap Verme	Kararlaştırılan Planı Takip Etme

Kaynak: PMI (27 Mart 2019) tarihinde https://www.pmi.org/learning/library/agile-versus-waterfall-approach-erp-project-6300 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

Çevik geliştirme modelininin başarılı olması için, ekip üyeleri ve müşteri temsilcilerinin çevik yaklaşımla uyumlu bir şekilde çalışması, aşağıda detayları verilen "insan faktörleri" nin sürece uygun olması gerekmektedir.

- **Yetkinlik**: Ekip üyelerinin her biri kendi fonksiyonel rollerinde yetkin olmalıdır.
- **Ortak Odak**: Tüm üyeler, geliştirilen ürün ve özellikleriyle benzer bağlılığa sahip olmalıdır.
- **İşbirliği**: Ekip içinde karşılıklı güven ve diğer üyelere saygı duyulmalı, ekip bir "jöle" takımı gibi "kendi kendini örgütleme" yeteneğine sahip olması beklenmektedir.
- **Karar Verme**: Üyeler, ortaya çıkan ve uzlaşma kararlarını alan farklılıkları ortadan kaldırmak için rasyonel mantık kullanmalıdır. Takımın karar alma süreci, önceden bilinen proje kısıtlamaları dışında dış baskılardan arındırılmış olmalıdır.
- **Belirsiz Problem Çözme Yeteneği**: Sağlam sağduyuya ve üyeler arasındaki farklı bakış açılarını göz önünde bulundurabilecek açık zihniyete dayalı bu yetenek, ekip tarafından gereksiz bir gecikme olmadan iyi karar alınmasını sağlayacaktır.

2.3.2. Çevik Manifesto'nun Arkasındaki İlkeler

2001 yılında Fowler ve on altı diğer yazılım geliştirme uzmanı bir araya gelmiş ve on iki çevik prensibi aşağıdaki şekilde tanımlamıştır (Fowler ve Highsmith, 2001).

- En yüksek önceliğimiz, değerli yazılımların erken ve sürekli teslimatı yoluyla müşteriyi memnun etmektir.
- Gereksinimler yazılımın sonunda değişse bile hoş karşılayın. Çevik süreçler, müşterinin rekabet avantajı için koşum değişikliklerini gerçekleştirir.
- Çalışan bir yazılımı, daha kısa zaman çizelgesini tercih ederek birkaç haftadan birkaç aya kadar süregelen bir zaman diliminde, sık sık müşteriye teslim edin.
- Paydaşlar ve geliştiriciler, proje boyunca günlük olarak birlikte çalışmalıdır.
- Motive olmuş bireyler etrafında projeler oluşturun. Onlara ihtiyaç duydukları ortamı ve desteği sağlayın, işi yapmaları için onlara güvenin.

- Bir geliştirme ekibine bilgi aktarmanın en etkili yöntemi yüz yüze konuşmadır.
- Çalışan bir yazılım, ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir kalkınmayı teşvik eder. Sponsorlar, geliştiriciler ve kullanıcılar, süresiz olarak sabit bir hızda kalabilmelidir.
- Teknik mükemmellik ve iyi tasarıma sürekli ilgi, çevikliği arttırır.
- Sadelik "yapılmayan iş miktarını en aza indirme sanatı" esastır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize eden ekiplerden doğar.
- Düzenli aralıklarla, ekibin nasıl daha etkili olacağına karar verilir, sonra davranış buna göre ayarlanır.

Çevik yöntemlerin farklı sayısız önerisi olsa da, Agile Alliance aşağıda ortak ilkeleri özetlemiştir (Agile Alliance, 2001).

- Paydaşlar ve yazılım geliştiriciler, geliştirme aşamalarında iletişim boşluklarını ortadan kaldırarak veya en aza indirerek birlikte çalışırlar. Bu hareketin geleneksel geliştirme modellerinde yaygın olan "bizler ve onlar" tutumunu kaldırması beklenir. Yüz yüze iletişim tercih edilir.
- Ürünün mevcut algılanan ihtiyaçları karşılaması için yazılım artışlarının mümkün olan en kısa sürede teslimi ve müşteri geri bildirimlerinin etkin kullanımı. Proje sırasında gereksinimlerdeki geç değişikliklere tepki gösterilmez.
- Geliştirme ekibi düzenli aralıklarla en son performanslarını kayıt altına alarak yansıtmaktadır. Bu, artan teslimat stratejilerinin geliştirilmesine yardımcı olur. Ekiplerin dışardan bürokratik kontroller olmadan kendi kendilerini organize etmeleri bekleniyor.

Yazılımın basitliği yalnızca şu anda üzerinde anlaşılan işlevleri yerine getirmek için çevik yaklaşımda vurgulanmaktadır. Bu durum "gelecekteki uzantılar için tasarım hükümleri oluşturan geleneksel gelişim stratejilerine bir tepkidir" denilmektedir (Agile Alliance, 2001).

Tablo 3 : Çevik Yöntemlerde Kullanılan Terminolojiler

Terminoloji	Tanım
Tekrarlı	Tekrarlanan. Tasarımın ve gereksinimlerin değiştirilmesine yardımcı olur.
Artırımlı	Sistem işlevsellik açısından küçük alt sistemlere ayrılır ve her yeni sürümde yeni bir işlevsellik eklenir.
Kendi Kendine Organize	Kendini örgütleme özerkliğine sahip olan ekip iş öğelerini en iyi şekilde tamamlamaktadır.
Acil	Gereksinim değişiklikleri ve ilgili yeni teknoloji, ürün geliştirme döngüsü boyunca ortaya çıkar.

Kaynak: Wikipedia (26 Mart 2019) tarihinde https://en.wikipedia.org/wiki/Agile_software_development 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

Çevik geliştirme projelerinde, geleneksel geliştirme projelerinde yapılan titiz planlamaya göre daha kesin bir planlama yapılmaktadır. Çevik planlamada en önemli prensip geri bildirim toplamaktır. Yazılımın çalışan sürümü veya minimum pazarlanabilir özelliği düzenli aralıklarla teslim edildikten sonra geri bildirim toplanmaktadır (Collins–Cope, 2002).

Geliştirme için Çevik Modeller

Bu bölümde, dünya genelinde kullanılan çevik yöntemlerin bazıları açıklanmaktadır. Tüm yöntemler uygulama şekilleri açısından farklılık gösterse de temelinde bazı özelliklere dayanmaktadır. Çevik yöntemler sekiz temel özellikle sınıflandırılmıştır (Abrahamsson vd., 2002: 13-19), (Cockburn, 2000: 35).

Tablo 4 : Çevik Yöntemlerin 8 Temel Özelliği

Fikir	Çıktı
Modeller	Benimseme
Teknikler ve Araçlar	Ürün
Kapsam	Roller

Kaynak: Arxiv (28 Mart 2019) tarihinde https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

Fikir: Bu kavram uygulama kümesini ve bu yöntemle ilişkili hedefleri belirtmektedir.

Modeller: Model tasarımı ve yönteme dâhil çalışma şeklini göstermektedir.

Teknikler ve Araçlar: Metodolojiyi takip ederken kullanılan araçların listesini açıklamaktadır. Bazı yöntemler araçları listelememektedir.

Kapsam: Belirli bir yöntemle ilgili sınırlamaları listelemektedir. Metodolojinin yaşam döngüsündeki kapsamını açıklamaktadır.

Çıktı: Bir yöntemin belirtilen aşamasından beklenen beklenen çıktıları listelemektedir.

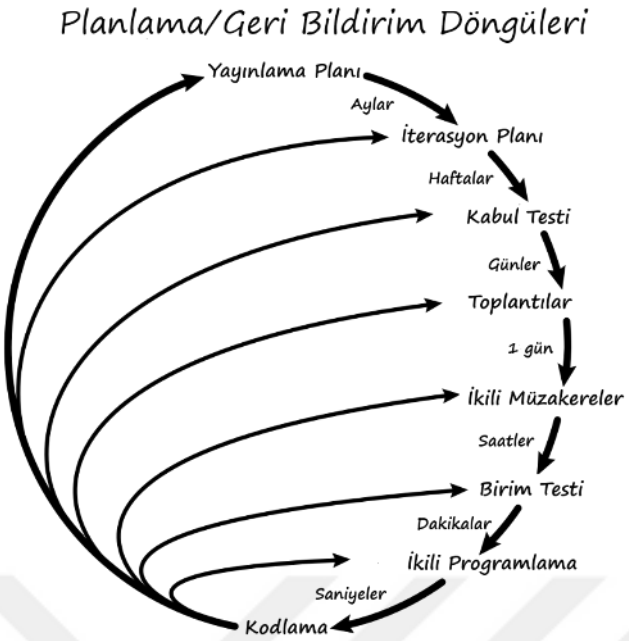
Benimseme: Belirli bir yöntemin geçmiş uygulamalardan geliştiğini açıklamaktadır. Metodolojiyi kullanan kuruluşları listelemektedir. Ayrıca, yöntemde belirtilen yetenek seviyelerini de belirtildiği gibi listelemektedir.

Ürün: Kılavuzları, eğitim için çevrimiçi araçları ve yöntemi tanımlayan kitapları listelemektedir.

Roller: Proje ekip üyelerinin faaliyetlerini açıklamaktadır.

2.3.3. Ekstrem Programlama

Ekstrem programlama (XP), yüksek kalitede yazılım ve geliştirme ekibi için daha yüksek yaşam kalitesi üretmeyi amaçlayan çevik bir yazılım geliştirme yöntemidir. XP, yazılım geliştirmeye uygun mühendislik uygulamalarıyla ilgili çevik yöntemlerin en belirgini şeklinde ifade edilmektedir (Agile Alliance, 2001).



Şekil 10. Ekstrem Programlama

Kaynak: Aleturo Teaching (01 Nisan 2019) tarihinde http://teaching.aleturo.com/Wild/slides/XP_Presentation.pdf 'den alındı.

XP, Kent Beck tarafından 1999 yılında Fowler ve diğer uzmanların desteğiyle geliştirilmiştir. Beck 'a göre, XP programlamadaki teknik ustalık, kod estetiği, geliştirici ekibi ve müşteriler arasındaki karşılıklı bağımlılığa dayanarak oluşturulmuştur. XP, projenin başarısının anahtarı olarak sorun ortamı hakkında karşılıklı bilgi alışverişinin yapıldığı iletişimi önemsemektedir (Beck, 1999: 1-3).

Brewer, ekibin çalışabilecek en basit şey olan minimum özelliği oluşturmaya başlaması gerektiğini netleştirmiştir. Geri bildirim, müşterinin değişiklikleri önerdiği ve ekibin teslim edilen yinelemeyi inceledikten sonra değişiklikleri kabul ettiği durumdur. XP takım düzeyinde gelişimsel faaliyetlere odaklanmaktadır (Brewer, 2001).

Sharp ve Robinson tarafından olgunlaşmış ekip kültürünün beş özelliği ekte listelenmektedir. Çalışmaları hikâye kartlarının önemini, bir kullanıcı hikâyesinin açıklamasını içeren kartlarını, XP ekipleri arasında işbirliği ve koordinasyon içinde verilen bir yinelemede uygulamak için kullandığı kullanıcı hikâyelerinden oluşan panolarını içermektedir (Sharp ve Robinson, 2008: 4-9).

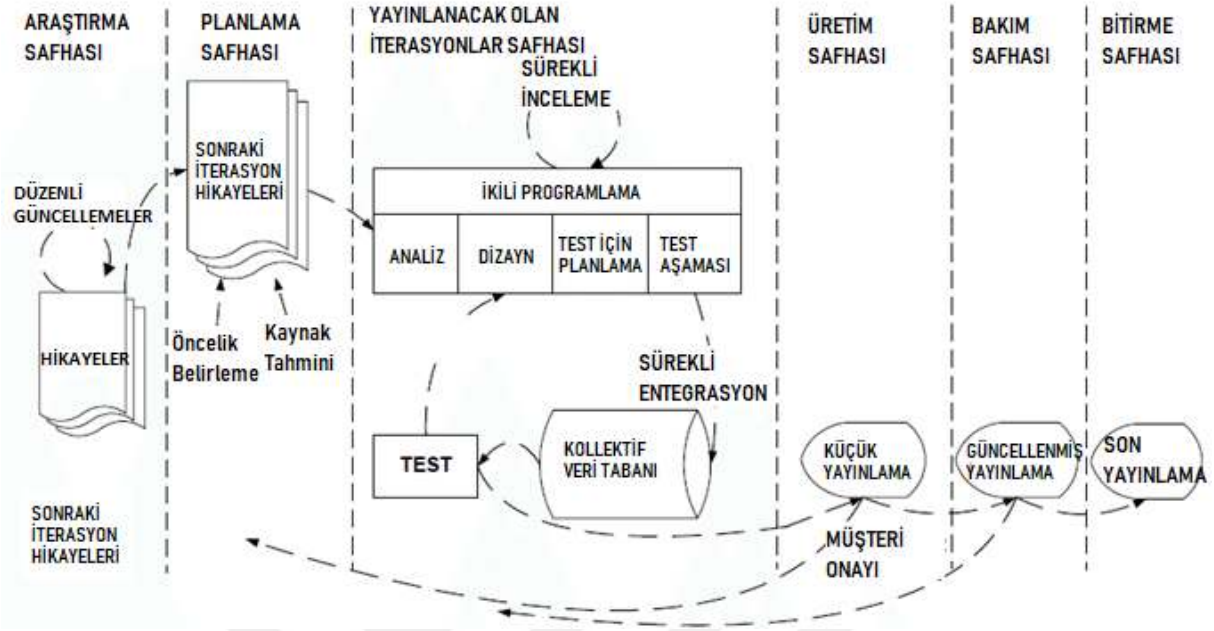
- Bireysel ve ekip düzeyinde saygı
- Bireysel ve ekip düzeyinde sorumluluk
- Çalışma hayatı kalitesini koruma
- Tekrar doğrulama ile birlikte kendisine olan güven artışı
- Güven

XP çalışma modelinde kullanıcı, görevleri kartlar üzerine yazmakta, daha sonra geliştirici hikâye başına gereken kodlama süresini tahmin etmekte, kullanıcıya ilk iterasyon için hikaye seçme ayrıcalığı verilmekte, her hikaye için fonksiyonel testler yazılmakta, kod geliştirilmekte, tek bir iterasyonda tasarımı mükemmelleştirmek için düzenleme yapılmakta ve geliştirme tamamlanana kadar döngü tekrarlanmaktadır (Beck, 1999: 18).

İkili programlama, XP'nin itibarına katkıda bulunan bir tekniktir. İki programcının yan yana bir bilgisayarda çalıştığı ve böylece aynı tasarım ve kod mantığı üzerinde sürekli işbirliği yaptığı bir programlama tarzıdır. Geliştirici tüm geliştirmeyi iş ortağıyla birlikte yapmaktadır. İlk – test programlaması, testin bitimine kadar beklemek yerine sistemin geliştirildiği gibi test edilmesi anlamına gelmektedir. Yeniden düzenleme, tasarımdaki değişimi sağlayan bir sonraki tekniktir. Diğer tekniklerde ise müşteri ve kod üzerinde ortak sahiplik bulunmaktadır. XP kendi kendine programlama için herhangi bir araç belirtmemekte, araçlar üzerindeki kararlar, proje gerekliliğine göre paydaşlar tarafından belirlenmektedir (Williams ve Kessler, 2000: 1-2).

XP' de kapsam olarak genellikle 3-10 kişi arasındaki küçük ve orta ölçekli ekipler gerektiren projeler için uygun görülmektedir. Williams ve Kessler (2000: 3) tarafından kararsız ve öngörülemeyen gereksinimlere sahip projeler için daha ideal olduğu ifade edilmektedir. Her küçük sürüm veya minimum pazarlanabilir özellik artışı için çalışır kod ve çift programlama yoluyla öğrenme sürecine sürekli katılımın bir sonucu olarak yüksek kaliteli programcılar elde edilmektedir.

XP yayınlama döngüsü Şekil 11'de gösterilmiştir.



Şekil 11. Ekstrem Programlama Yayınlama Döngüsü

Kaynak: Research Gate (02 Nisan 2019) tarihinde https://www.researchgate.net/profile/Shabib_Aftab/publication/316845761/figure/fig1/AS:492531558424576@1494440082066/Life-Cycle-of-Extreme-Programming-6.png 'den alındı.

XP, Kent ve Ward Cunningham tarafından pratikten geliştirilmiştir. Bu yöntem birden fazla çok uluslu şirkette yaygın şekilde kullanılan başarılı çevik yöntemlerden birisidir. Ayrıca akademik kurumlarda ders olarakta öğretilmiştir (Williams ve Kessler, 2000: 11).

Bu sonuçlar XP yönteminin şirketler ve akademik kurumlar tarafından benimsendiğini göstermektedir. XP rollerine bakıldığında geliştiricilerin kodu, müşterinin ise hikâyeyi yazdığı görülmektedir. Ayrıca müşteri fonksiyonel testleri yapmakta ve uygulama önceliğini de belirlemektedir. "Test uzmanı", müşterinin fonksiyonel testler yazmasına yardımcı olmakta, uygulama testini yapmakta ve test araçlarını korumaktadır. "İzleyici", tahminlerin doğruluğu hakkında geri bildirim vermekte ve iterasyonların ilerlemesini takip etmektedir. "Koç", XP sürecini takip etmek için takımı yönlendirmekte, "Danışman", takımı sorunları çözmesi için yönlendirmekte, "Yönetici" ise karar verilmesine yardımcı olmakta ve pratikte süreci takip etmektedir (Abrahamsson vd., 2002: 23).

Her rolle ilgili sorumluluklar ve gerekli yeterlilik Tablo 5' te özetlenmiştir.

Tablo 5 : Ekstrem Programlama Rolleri

Roller	**Açıklamalar**
Koç	Süreci bir bütün olarak yöneten, ekibi yönlendiren ve süreci diğer XP ekiplerinden öğrenmeyi anlayan kişidir.
Geliştirici	Çalışma ve kodlamada basitliği koruyan iyi iletişim becerilerine sahip kişidir.
Müşteri	XP ekibinin ayrılmaz bir parçası olan müşteri kuruluşundan bir kişidir.
Test Uzmanı	Müşteriye fonksiyonel testler yazmada yardımcı olan, düzenli olarak çalışan ve herkesin bilgisi için sonuçlar gönderen kişidir.
Proje Sponsoru	Takımın ilerlemesini düzenli olarak kontrol etmekten ve sorunlarını dinlemek için ekiple iletişim kurmasından sorumlu kişidir.

Kaynak: Extreme Programming (04 Nisan 2019) tarihinde http://www.extremeprogramming.org/ 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

XP uygulaması sırasında belirli değerlere özel vurgu Tablo 6' da özetlenmiştir.

Tablo 6 : Ekstrem Programlama Değerleri

Değerler	**Açıklamalar**
İletişim ve Saygı	XP, ekibin tüm üyeleri arasında dürüst iletişim ve yakın işbirliği gerektirir.
Basitlik	Takımın, daha sonraki aşamalarda asla kullanılamayacak olan karmaşık şeyler için çok fazla zaman ve çaba harcamak yerine basit şeyler üzerinde yoğunlaştığı anlamına gelmektedir.
Geri Bildirim	Programcılar ünite testleri ile kodlarından geri bildirim almakta, müşteriler, kullanıcı hikâyelerinin tamamlanmasıyla programcılardan geri bildirim almaktadır. Müşteriler incelemeler yoluyla ekibe geri bildirim sağlamaktadır.
Cesaret	Sorunları adresleme, problemleri çözme ve alternatif daha iyi tasarıma sahip uygulamalar lehine kodu eksiltme cesareti gerektirir.

Kaynak: Extreme Programming (04 Nisan 2019) tarihinde http://www.extremeprogramming.org/ 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

2.3.4. SCRUM

Scrum rehberine göre Scrum tanımı, “Takımlar için birlikte çalışarak ürün geliştirebilecekleri bir yöntemdir. Scrum kullanılarak ürün geliştirme, küçük parçalar halinde gerçekleşir ve her parça önceden oluşturulmuş parçalar üzerine inşa edilir. Ürünleri tek seferde küçük bir parça halinde oluşturmak, yaratıcılığı teşvik eder ve ekiplerin geri bildirime ve değişime yanıt vermesini, tam olarak ve sadece ihtiyaç duyulanı oluşturmasını sağlar.” şeklinde ifade edilmektedir (Schwaber ve Sutherland, 2017: 3). Bu tanım Scrum'un bir geliştirme yöntemi olduğunu ve karmaşık projelerin başarısı için kullanıldığını göstermektedir. Öte yandan, bu rehber, Scrum'un sadece “geliştirme yöntemi” olmadığını, insanın yaratıcı, etkili ve daha etkileşimli olmasını desteklediğini de ifade etmektedir.

Başka bir tanımda Scrum uzmanlarına göre “Scrum, karmaşık projeleri tamamlamak için kullanılan çevik bir yöntemdir. Başlangıçta yazılım geliştirme projeleri için şekillendirilmiştir, ancak karmaşık, yenilikçi iş kapsamı için iyi şekilde çalışmaktadır. İmkânları sınırsızdır. Scrum yöntemi aldatıcı bir şekilde basittir.” denilmektedir (Scrum Alliance, 2009).

Scrum için yapılan bir başka tanımda, “Scrum, yazılım projelerini ve ürün veya uygulama geliştirmeyi yönetmek için yinelemeli ve artımlı çevik bir yazılım geliştirme yöntemidir. "Bir geliştirme ekibinin ortak bir hedefe ulaşmak için bir birim olarak çalıştığı esnek, bütünsel bir ürün geliştirme stratejisi" olarak tanımlamaktadır. Ürün geliştirmeye yönelik "geleneksel, sıralı yaklaşım" varsayımlarına meydan okur.” denilmektedir (Scrum Alliance, 2009).

Krishnamurthy, “İnsanların bir kısmı tarafından Scrum bir süreç olarak görülmektedir. Ancak bu tanımlar dikkate alındığında Scrum için “özel bir yöntem” ifadesini kullanmak daha doğru olacaktır.” şeklinde bir tanımlama yapmıştır (Krishnamurthy, 2012).

Tüm tanımlamalar incelendiğinde Scrum’un bir geliştirme yönteminden fazlasını içerdiğinden emin olmakla birlikte özünde sadelik ve basitlik barındırdığı ifade edilebilir. Bütünsel bir ürün stratejisi ile esneklik ve hız sağlamakta, odaklanma noktasındaki etkileşimleriyle yaratıcılığı ve kaliteyi arttırmaktadır.

Scrum Tarihçesi

Scrum; ilk önce yeni, hızlı, esnek ve kendini organize eden bir ürün geliştirme süreci olarak tanımlanmıştır. Stratejisinin temelini "oyun dışı topla tekrar oyuna girmek" kavramıyla rugby oyunundan almaktadır (Schwaber ve Beedle, 2004: 130).

OOPSLA konferansında (1995) Schwaber ve Sutherland tarafından scrum deneyimleri paylaşılmıştır. Schwaber ve Beedle, Scrum kullanarak "Agile Software Development with Scrum" adıyla ilk Scrum kitabını yazmıştır (Krishnamurthy, 2012). 2001 yılında Scrum toplulukları tarafından "Sertifikalı Scrum Uzmanı" yetiştirilmek amacıyla Scrum Alliance adıyla bir platform oluşturulmuştur (Scrum Alliance, 2009).

Scrum Teorisi

Scrum kendi özelliği olan şeffalık, denetim ve adaptasyon ile desteklenen deneysel bir süreçtir. Sürecin şeffaflığı tüm paydaşlar tarafından görülebilir olmalı, süreç yetenekli denetçiler tarafından belirli dönemlerde denetlenmelidir. Adaptasyona katkı sağlayan dört scrum etkinliği bulunmaktadır (Scrum Alliance, 2009).

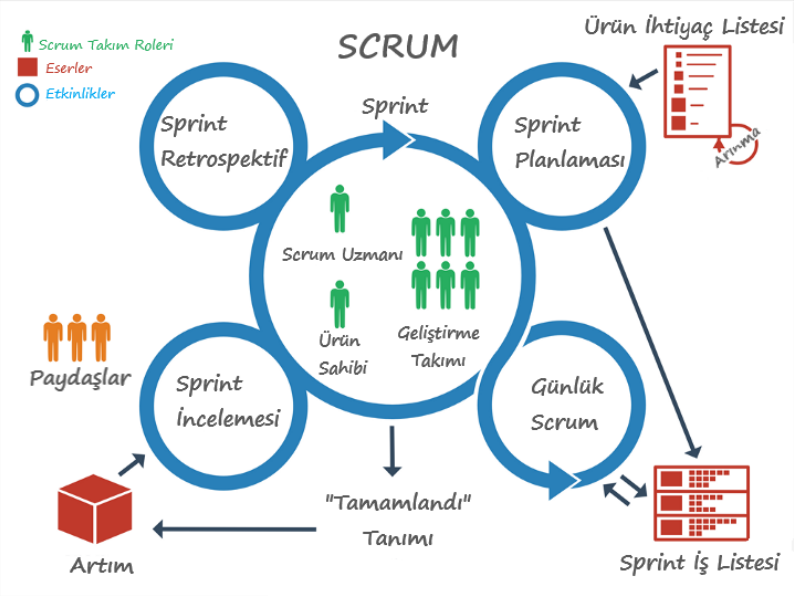
Bu etkinlikler aşağıda paylaşılmış olup detayları ilerleyen bölümlerde anlatılacaktır.

- Günlük Scrum Toplantısı
- Sprint Planlama Toplantısı
- Sprint İnceleme Toplantısı
- Geçmişe Yönelik Sprint Toplantısı

Scrum Takımları

Önemli bir hedefe ulaşmak için, her organizasyon bir takım olarak hareket etmeli ve organizasyon üyeleri birbirlerinin hatalarını düzeltmelidir. Scrum takımı aynı zamanda geliştirilen ürünün takımı yansıtacağını "aynası olacağını" bilmektedir. Bu gerçeği göz önünde bulundurmak, ekip büyüklüğü, ekip üyeleri içindeki iletişim, üyelerin birbirlerine güvenmesi, scrum uyumu için çok önemlidir. Sistemin teknik tasarımı, ekip uzmanlarının alan bilgileri,

yeni teknolojilerin adaptasyonu, proje teslim tarihi ile ilgili beklentiler ekibi ve ürün başarısını etkilemektedir (Cohn, 2009: 175-177).



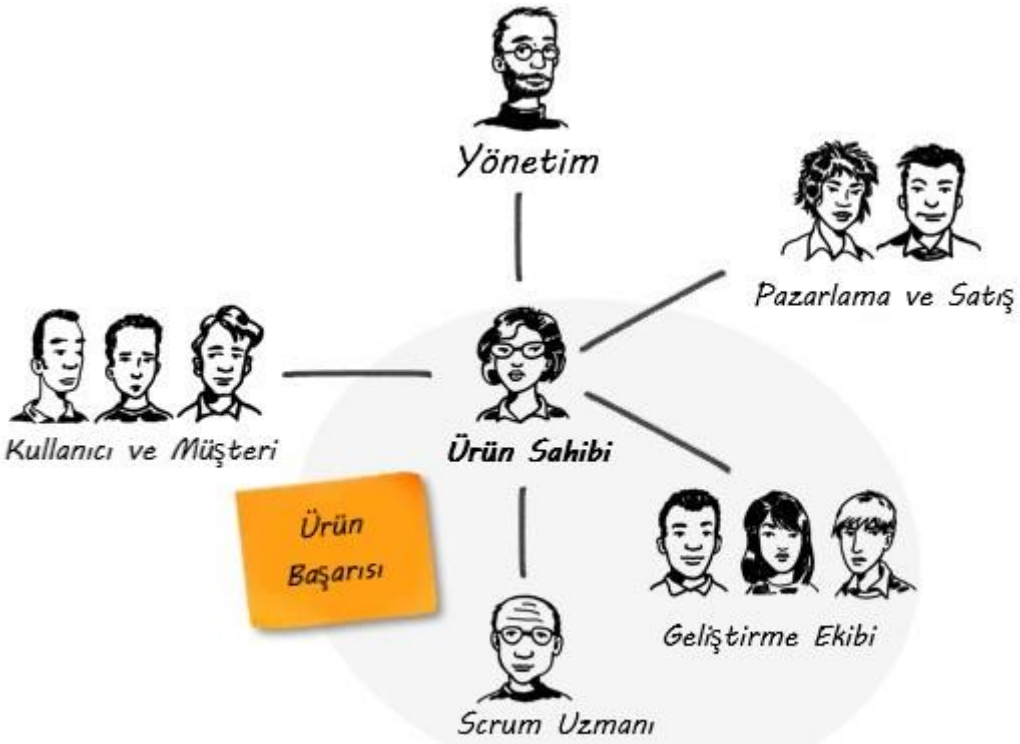
Şekil 12. Scrum Takımı ve Geliştirme Döngüsü

Kaynak: Scrum Guide (07 Nisan 2019) tarihinde https://www.scrum.org/resources/scrum-guide 'den alındı.

Scrum takımı fonksiyonel olmalı ve kendi kendini organize etmelidir. Bu her takım üyesinin birbirlerinin becerilerine bağlı olmadığı ve takım hakkında kendi kararlarını verdikleri anlamına gelmektedir. Ürünler, iş fırsatları elde etmek için scrum ekipleri tarafından tekrarlı ve artımlı bir şekilde teslim edilir. Scrum ekibi tarafından "Tamamlanan" her ürün potansiyel olarak faydalı ve çalışır kabul edilir. "İhtiyaç listesini tamamladınız mı?" sorusu scrum ekibinin her üyesi için yeterli değildir, aynı zamanda Scrum çalışmalarında müşteri memnuniyeti ve takım hedefleri de ciddi şekilde düşünülmektedir. Scrum ekibi, "Ürün Sahibi, Geliştirme Takımı ve Scrum Uzmanından" oluşmaktadır (Schwaber ve Sutherland, 2017: 6).

Scrum Rolleri

Bu alt bölümde Scrum için en önemli parçalardan birisi olan roller incelenecektir. Scrum yönteminde roller diğer yöntemlere göre farklılık göstermektedir. Yöntemin kendisi ve işleyişi de geleneksel metotlara kıyasla farklılık göstermekte, ekip üyeleri ve ekibin çalışma şekli de tamamen farklıdır.



Şekil 13. Scrum Rolleri

Kaynak: romanpichler (10 Nisan 2019) tarihinde https://www.romanpichler.com/blog/one-page-product-owner/ 'den alındı.

Ürün Sahibi (Product Owner)

Ürün sahibi, ürünün değerini en üst düzeye çıkarmaya çalışmaktadır. Bunu yapabilmek için pazar trendlerini araştırır, ürün gereksinimlerinin yönetiminden sorumlu tek kişidir. Ürün gereksinimlerinin yönetimi, ürün kalemlerinin açık bir şekilde tanımlanmasını, ürün biriktirme listesindeki öğelerin sıralanmasını, öğelerin her ekip üyesi için şeffaflığının, görünürlüğünün ve netliğinin sağlanmasını içermektedir (Schwaber ve Sutherland, 2017: 6-7).

Ürün sahibi işletmeyi çok iyi tanıyor olmalıdır. Şayet böyle bir kişi yoksa üye olmayan üyelerden biri (Örneğin, scrum uzmanı da ürün sahibi olabilir) ürün sahibinin rolünü üstlenebilir ancak normal durumlarda herkesin yalnızca bir rolü olmaktadır (Levison, 2008).

Ürün Sahibi aşağıdaki görevlerden sorumludur.

- Ürünü tanımlamak.
- Piyasa değerlerini dikkate almak suretiyle biriken kalemleri sipariş etmek.
- Ürün teslim tarihine karar vermek.
- Gerekli görüldüğünde, her sprint döngüsünde içerik ve önceliklerde değişiklik yapmak.
- Üründen sorumlu olmak.
- Ürün özellikleri kabul ya red etmek.

Ürün sahibi, ürünün sahibidir, ancak geliştirme ekibiyle birlikte tüm ürünler üzerinde ortak sahiplik hakkı bulunmaktadır. Bu durum Scrum içindeki mükemmelliğin sürdürülebilirliğini ve ürün yeniliğine yönelik arzuyu güçlendirmektedir (Scrum, 2017).

Scrum Uzmanı (Scrum Master)

Scrum uzmanının temel görevi sürecin çok iyi çalışmasını sağlamaktır. Scrum Uzmanı, scrum takımı için "hizmetkâr – lider" rolü üstlenmektedir. Yönetici değildir, ancak lider özelliklere sahip olması ve sorunların üstesinden gelmek zorunda olduğu içinde etki alanını iyi bilmesi gerekmektedir (Schwaber ve Sutherland, 2017: 7-8).

Scrum uzmanı;

- Ekip işlevselliğini, üretkenliğini ve motivasyonu artırmaya çalışır.
- Takım üzerindeki ilişkileri organize eder.
- Yüksek performanslı bir ekip kurmaya çalışır.
- Ekip işlevselliğini ve süreci etkileyen problemleri ortadan kaldırır.
- Takımı dikkat dağılmasından korur.
- Her üyenin günlük görüşmelere, sprint planlamasına ve sprint inceleme toplantılarına katılımını takip eder.
- Yanma (burn-down) çizelgelerini izler.
- Scrum'un çalıştığından emin olur.

Mountain Goat Software kuruluşundan Mike Cohn, iyi bir scrum uzmanına ait altı özelliği aşağıdaki şekilde ifade etmektedir (Cohn, 2007).

- Sorumluluk sahibi
- Mütevazı
- İşbirlikçi
- İşlenen
- Etkili
- Bilgili

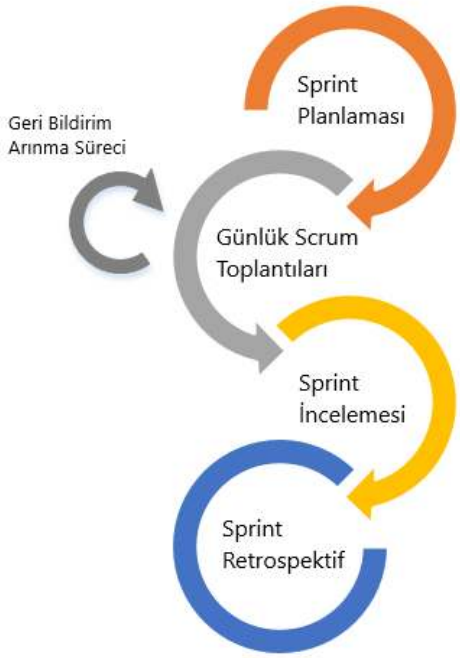
Geliştirme Ekibi (Development Team)

Geliştirme Ekibi, her bir Sprint hedefine ulaşmak için gerekli eylemlere karar verme ve kendisini organize etme yetkisine sahip proje ekibidir (Abrahamsson vd., 2002: 33).

Geliştirme ekibi dengeli becerilere sahiptir. Tüm üyeler "geliştirici" ünvanına sahiptir. Başarı ve başarısızlıktan bir kişi değil tüm ekip sorumludur. İş süreçlerinin sağlıklı bir şekilde yürütülmesi ve takım içi iletişim zorlukları nedeniyle ekipler az kişiden oluşmaktadır (Schwaber ve Sutherland, 2017: 7).

Scrum Etkinlikleri

Scrum'da düzenlilik oluşturmak ve tanımlanmayan toplantılara duyulan ihtiyacı en aza indirmek için önceden belirlenmiş etkinlikler kullanılmaktadır. Tüm etkinlikler zamana bağımlıdır. Sprint başladığında süresi sabittir, kısaltılamaz veya uzatılamaz. Kalan etkinlikler, etkinliğin amacına ulaşıldığı zaman sona erebilir, bu da süreç içerinde israfa izin vermeden zamanın uygun bir şekilde kullanılmasını sağlamaktadır (Schwaber ve Sutherland, 2017: 9).



Şekil 14. Scrum Etkinlikleri

Kaynak: Medium (11 Nisan 2019) tarihinde https://medium.com/@magdamiu/events-in-scrum-a7cc0dfe69ea 'den alındı.

Scrum etkinlikleri aşağıda paylaşılmış olup detayları ilerleyen bölümlerde anlatılacaktır.

- Sprint
- Sprint Planlaması
- Günlük Scrum Toplantısı
- Sprint İncelemesi
- Geçmişe Yönelik Sprint

Sprint

Sprint, değişen çevresel değişkenlere (gereksinimler, zaman, kaynaklar, bilgi, teknoloji vb.) uyum sağlama prosedürüdür. Scrum Takımı, Sprint' te yaklaşık otuz takvim günü süren yeni bir çalıştırılabilir ürün artışı üretmek için kendisini organize etmektedir (Abrahamsson vd., 2002: 34).

Sprint sırasında:

- Sprint hedefini tehlikeye atacak hiçbir değişiklik yapılmaz.
- Kalite hedefleri azalmaz.
- Kapsam, gerekli durumlarda Ürün Sahibi ile Geliştirme Ekibi arasında netleştirilebilir ve yeniden müzakere edilebilir.

Sprint için yapılan tanımı incelendiğinde, bir aydan fazla ufku bulunmayan bir proje olarak ifade edilebilir. Projeler gibi sprintler de bir şeyi başarmak için kullanılmaktadır. Her Sprint içinde neyin geliştirileceğine, geliştirme sürecine rehber olacak esnek bir tasarıma, geliştirme sonunda "işleyen" ve ortaya çıkan ürünü artırmak gibi bir amacı bulunmaktadır.

Sprint için verilen süre çok uzun olduğunda, geliştirilen ürünün tanımı değişebilir, karmaşıklık ve risk artabilir. Sprint'ler, en azından her takvim ayında bir Sprint Hedefine yönelik ilerlemenin denetlenmesini ve uyarlanmasını sağlayarak öngörülebilirliği sağlamaktadır. Ayrıca süresi bir takvim ayı ile kısıtlandığından, göze alanın risk bir takvim ayı maliyetiyle sınırlanmaktadır (Schwaber ve Sutherland, 2017).

Sprint Planlaması

Scrum ekibinin tamamı, yapılacakların gündemini belirlemek için bir toplantı yapmaktadır. Her Sprint başında düzenlenen "Ürün İş Listesi" toplantısı yapıldıktan sonra her bir özelliğin amacı ve misyonu "Ürün Sahibi" tarafından belirlenen şekilde tartışmaya açılır. Sprint Planlama bölümünde ekip üyeleri, her bir özelliğin çalışması için her bir üyenin kaç saat harcayacağını belirlemektedir. Bu biriktirmeyi seçenlerin, neden bu görevlerin seçileceğini açıklamaları ve ne yapılması gerektiğini anlamaları gerekir (Permana, 2015: 199).

Sprint Planlaması aşağıdaki soruları yanıtlamaktadır:

- Yaklaşan Sprint sonucundaki artışla ne teslim edilebilir?
- Artış elde edilecek teslimat için nasıl bir çalışmaya ihtiyaç duyuluyor?

Sprint Hedefi

Sprint Hedefi, Sprint için belirlenen Ürün İhtiyaç Listesinin uygulanması yoluyla karşılanabilir bir amaçtır. Geliştirme Ekibine, artışın neden oluştuğu konusunda rehberlik etmektedir. Sprint Planlama toplantısı sırasında oluşturulur. Sprint Hedefi, Geliştirme Ekibine Sprint içerisinde uygulanan fonksiyonellik konusunda biraz esneklik sağlamaktadır. Geliştirme Ekibi çalıştıkça, Sprint Hedefi her zaman akılda tutulmaktadır (Schwaber ve Sutherland, 2017).

Günlük Scrum

Günlük Scrum toplantıları, Scrum Takımı'nın ilerlemesini sürekli takip etmek için düzenlenir. Ayrıca yapılan bu kısa (yaklaşık 15 dakika) günlük toplantıda sorunlar ve diğer değişik konular görüşülerek kontrol edilir. Bu toplantılar "Scrum Uzmanı" tarafından düzenlenir. Toplantıya scrum takımının yanı sıra yöneticiler de katılabilir (Abrahamsson vd., 2002: 35).

Sprint İncelemesi

Sprint İncelemesi, ilerlemeyi incelemek ve gerekli durumlarda Ürün İş listesini uyarlamak için Sprint sonunda yapılmaktadır. Sprint İncelemesi sırasında, Scrum Takımı ve paydaşlar Sprint'te ne yapıldığı konusunda işbirliği halindedir. Sprint sırasındaki Ürün İş Listesinde yapılan değişikliklere dayanarak, katılımcılar değeri optimize etmek için yapılabilecek sonraki şeyler üzerinde işbirliği yaparlar. Bu bir durum toplantısı değil, gayri resmi bir toplantıdır ve ilerlemenin sunumu geri bildirimi ve işbirliğini teşvik etmeyi amaçlamaktadır (Schwaber ve Sutherland, 2017).

Sprint İncelemesinin sonucunda bir sonraki Sprint için Ürün İş Listesi öğelerini tanımlayan gözden geçirilmiş bir liste elde edilmektedir.

Şekil 15. Sprint İncelemesi

Kaynak: Scrum.org (12 Nisan 2019) tarihinde https://www.scrum.org/resources/what-is-a-sprint-review 'den alındı.

Sprint İncelemesi aşağıdaki unsurları içermektedir:

- Ürün Sahibi tarafından davet edilen Scrum Ekibi ve kilit paydaşlar
- Ürün İş Listesi öğelerinin "Tamamlandı" ve "Yapılmadı" durumunda olduğunun bilgisi;
- Sprint sırasında neyin iyi gittiği, hangi problemlerle karşılaşıldığı ve nasıl çözüldüğü;
- Geliştirme Ekibi "Tamamlandı" çalışmasını gösterir ve artışla ilgili soruları cevaplar.
- Ekibin tamamı daha sonra yapılacak işler hakkında işbirliği halindedir, Sprint İncelemesi sonra yapılacak Sprint Planlamaları için değerli girdiler üretmektedir.
- Pazarın veya ürünün potansiyel kullanımının nasıl değişebileceğinin gözden geçirilmesi, daha sonra yapılacak en değerli şeyin ne olduğu;
- Ürünün bir sonraki beklenen işlevsellik ve yetenek sürümleri için zaman çizelgesi, bütçe, potansiyel yetenekleri ve pazarın gözden geçirilmesi.

Sprint Retrospektif

Sprint Retrospektif (SR), "Scrum ekibinin kendisini denetlemesi ve bir sonraki Sprint boyunca yürürlüğe girecek iyileştirmeler için bir plan yaratması adına bir fırsat" olarak tanımlanmaktadır. SR verilerine Sprint İncelemesinden sonra ve bir sonraki Sprint

planlamasından önce ulaşılmaktadır. Bir aylık Sprint döngüsünde en çok üç saatlik bir toplantıdır. Daha kısa sprintler için, etkinlik genellikle daha kısadır. Scrum Uzmanı, etkinliğin gerçekleşmesini ve katılımcıların amacı anlamalarını sağlamaktadır. Bu etkinlik için "tüm üyelerin katılımıyla Scrum ekibinin gelişmesi için bir fırsat" denilmektedir (Scrum, 2017).

Şekil 16. Sprint Retrospektif

Kaynak: Scrum.org (12 Nisan 2019) tarihinde https://www.scrum.org/resources/what-is-a-sprint-retrospective 'den alındı.

Sprint Retrospektifinde ekip aşağıdaki konuları tartışır:

- Ne işe yaradı?
- Neler iyileştirilebilir?
- Bir sonraki Sprint için ne yapacağız?

Scrum Uzmanı, bir sonraki Sprint'i daha etkili ve eğlenceli hale getirmek için Scrum ekibini geliştirme sürecini ve uygulamalarını geliştirmeye teşvik etmektedir. Her SR etkinliğinde, Scrum ekibi ürün veya organizasyon standartlarıyla çelişmemesi halinde, "Tamamlandı" tanımının uyarlanması için iş süreçlerini iyileştirerek ürün kalitesini arttırmanın yollarını planlamaktadır (Schwaber ve Sutherland, 2017).

Scrum Eserleri

Scrum eserleri, denetim ve adaptasyon amacıyla şeffaflık ve fırsatlar sağlamak için işi veya değeri temsil etmektedir. Scrum tarafından tanımlanan eserler, temel bilgilerin şeffaflığını en üst düzeye çıkarmak için tasarlanmıştır, böylece herkes aynı yapıyı anlayabilmektedir (Schwaber ve Sutherland, 2017).

Sprint Eserleri aşağıdaki unsurları içermektedir.

- Ürün İhtiyaç Listesi
- Sprint İş Listesi
- Artım

Bu unsurlar ayrı başlıklar altında detaylı şekilde anlatılmıştır.

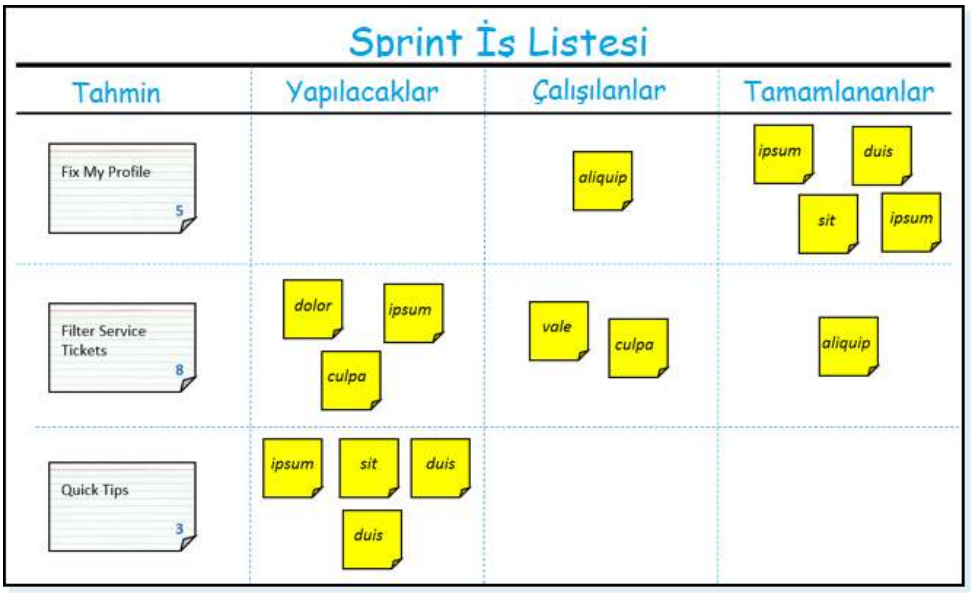
Ürün İş/İhtiyaç Listesi

Ürün İş Listesi, üründe gerekli olduğu bilinen her şeyin sıralı listesidir. Ürün İş Listesi, gelecekteki sürümlerde üründe yapılacak değişiklikleri oluşturan tüm özellikleri, işlevleri, gereksinimleri, geliştirmeleri ve düzeltmeleri listeler. Ürün Sahibi, içeriği, bulunabilirliği ve siparişi dahil olmak üzere Ürün İş Listesinden sorumludur (Scrum, 2017).

Ürün Sahibi, ürün biriktirme listesi olarak çıkacak biriktirmeleri hazırlamaktadır. Scrum'ın ilk adımında Scrum uzmanının önceliklerine göre özelliklerin belirlenmektedir. Ürün İhtiyaç listesinin belirlenmesi durumunda, "Scrum Uzmanı" rolünü "Proje Yöneticisi" üstlenecektir (Permana, 2015: 200).

Sprint İş Listesi

Sprint İş Listesi için "Sprint için seçilen Ürün İş Listesi öğelerinin kümesidir, ayrıca ürünü geliştirmek ve Sprint Hedefini gerçekleştirmek için bir plandır. Bir sonraki artışta hangi işlevselliğin olacağını ve bu işlevin bir "Tamamlandı" haline getirilmesi için gereken çalışma hakkında Geliştirme Ekibi tarafından yapılan bir tahmindir." şeklinde bir tanım yapılmaktadır (Scrum, 2017).



Şekil 17. Sprint İş Listesi

Kaynak: Scrum.org (13 Nisan 2019) tarihinde https://www.scrum.org/resources/what-is-a-sprint-backlog 'den alındı.

Sprint İş Listesi, Geliştirme Ekibinin Sprint Hedefine ulaşmak için gerekli gördüğü tüm çalışmaları görünür kılmaktadır. Sürekli iyileştirme sağlamak için, önceki Retrospektif toplantısında belirlenen en az bir yüksek öncelikli süreç iyileştirme içermektedir. Devam eden değişikliklerin Günlük Scrum'da anlaşılabileceği yeterince ayrıntılı bir plandır. Geliştirme ekibi, Sprint süresince Sprint İş Listesini değiştirir. Bu değişikliği sadece bu ekip yapabilmektedir. Sprint İş Listesi, ekibin Sprint sırasında gerçekleştirmeyi planladığı işin son derece görünür, gerçek zamanlı bir resmidir ve yalnızca geliştirme ekibine aittir (Schwaber ve Sutherland, 2017: 15).

Artım

Artım, bir Sprint sırasında tamamlanan tüm öğelerin toplamı ve önceki tüm Sprint'lerin artış değeridir. Bir Sprint sonunda, yeni artımın "Tamamlandı" durumuna gelmesi beklenmektedir. Bu da kullanılabilir durumda olması ve Scrum ekibinin "Tamamlandı" tanımına uygun olması gerektiği anlamına gelmektedir. Her artım, Sprint sonunda deneyimlemeyi destekleyen kontrol edilebilen, tamamlanmış bir çalışmadır. Artım, bir vizyon ya da hedefe doğru atılan bir adımdır. Ürün sahibinin yayınlamaya karar verip vermemesine bakılmaksızın kullanılabilir durumda olmalıdır. Scrum'ın tüm amacı bir "Tamamlandı" artımı sağlamaktır (Schwaber ve Sutherland, 2017).

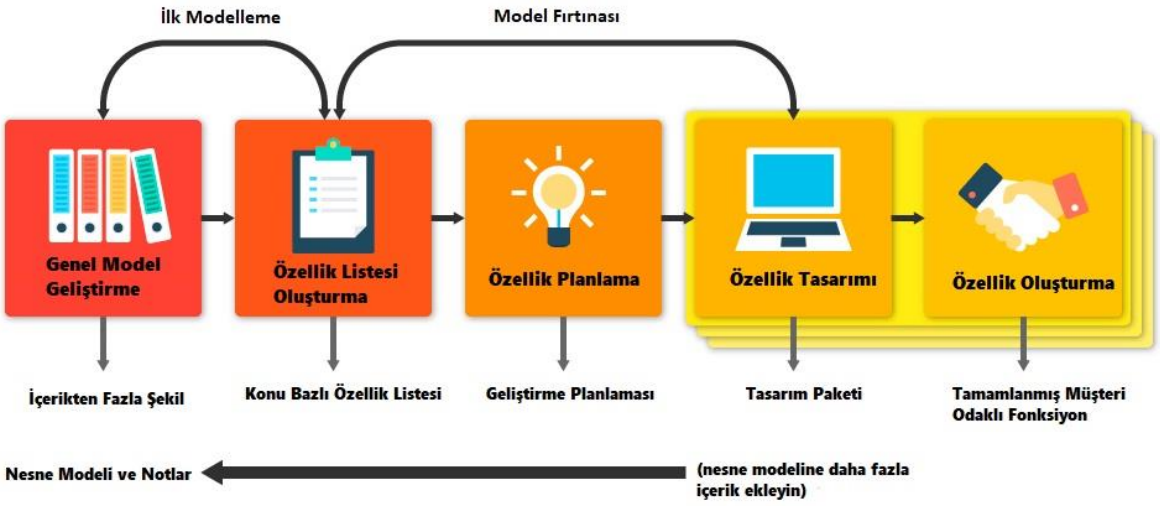
"Tamamlandı" 'nın Tanımı

Bir Ürün İş Listesi öğesi veya bir Artım "Tamamlandı" olarak ifade edildiğinde, herkes "Tamamlandı" olarak anlamalıdır. Bu tanım, Scrum ekiplerine göre önemli ölçüde değişebilmesine rağmen, üyelerin şeffaflığın sağlanması için çalışmanın tamamlanmasının ne demek olduğu konusunda ortak bir anlayışı olmalıdır. Bu ifade, "Tamamlandı" nın tanımıdır ve ürün Artımı üzerinde çalışmanın ne zaman tamamlandığını değerlendirmek için kullanılmaktadır (Scrum, 2017).

Aynı tanım, Sprint Planlaması sırasında kaç Ürün İş Listesi'nin seçebileceğini bilmek konusunda Geliştirme Ekibine rehberlik etmektedir. Her Sprint'in amacı, Scrum ekibinin mevcut "Tamamlandı" tanımına uyan potansiyel olarak serbest bırakılabilir işlevsellik artımı sağlamaktır. Geliştirme Ekipleri, her Sprint'te ürün işlevselliği noktasında artım sağlamakta, bu Artım kullanılabilir olmaktadır. Bu nedenle Ürün Sahibi derhal serbest bırakmayı seçebilir. Bir artım için "Tamamlandı" tanımı, geliştirme organizasyonunun sözleşmelerinin, standartlarının veya yönergelerinin bir parçasıysa, tüm Scrum ekipleri asgari olarak süreci takip etmektedir (Schwaber ve Sutherland, 2017: 16).

2.3.5. Özellik Odaklı Geliştirme

Özellik Odaklı Geliştirme (FDD), müşteri odaklı, mimari merkezli ve faydacı bir yazılım sürecidir. FDD yöntemindeki "müşteri" terimi, çevik modellemenin (AM) proje paydaşları veya XP müşterileri olarak adlandırdıklarını ifade etmek için kullanılmaktadır. FDD ilk olarak 1999 yılında "Java Modeling In Color with UML" kitabında, Jeff DeLuca'nın şirketi ve Peter Coad tarafından tanımlanan özelliklerin konseptini takip eden yazılım sürecinin bir kombinasyonu olarak dünyaya tanıtılmıştır. İlk FDD uygulaması 1997 yılında büyük bir Singapur bankası için 15 aylık, 50 kişilik bir projede denenmiş ve bunu hemen ardından 18 ay süren ikinci bir 250 kişilik proje izlemiştir (Agile Modeling, 2019).



Şekil 18. Özellik Odaklı Geliştirme (FDD)

Kaynak: New Line Technologies (18 Nisan 2019) tarihinde https://newline.tech/blog/feature-driven-development-methodology/ 'den alındı.

FDD, Nesneye Yönelik Analiz ve Tasarım (OOAD) ve projenin gerçeklerini grafiksel olarak yakalayan bir projenin modellenmesi temelinde çalışmaktadır. Geliştirici kısa yinelemeli özellik tabanlı planlama ve tasarım sürecinden, yönetici ise projenin kolay bir şekilde ortaya çıkmasını sağlayan proje ilerlemesi ve durumu tablosundan faydalanmaktadır (Palmer ve Felsing, 2002: 42).

Bu yöntemde kullanılan etki alanı nesne modeli, her etki alanının özellikler listesine göre oluşturulur. Planlama süreci bu özelliklere dayanmaktadır. Yinelemeli tasarımın son aşaması, ana geliştirme ile sonuçlanan özelliktir. Bir özelliğin yinelenmesi normalde bir ila üç hafta sürmektedir (Palmer ve Felsing, 2002: 42).

FDD, dört ila onbeş arasında değişen küçük ve orta ölçekli ekiplerle sınırlıdır. İlk çıktı, nesne modellerinin analizinden üretilen kapsamlı bir özellikler listesi olmaktadır. Özellik listesinin elde edilmesinden sonra genel bir model üretilmekte ve sırayla sınıf sahibi olarak adlandırılan bireysel geliştiricilere iletilmektedir. Tasarım ve geliştirme son kaynak koduyla nihai halini almaktadır. Nesne odaklı projelerde FDD yönteminin kullanımı önerilmektedir (Palmer ve Felsing, 2002: 42).

Roller

Tablo 7 : FDD Rolleri

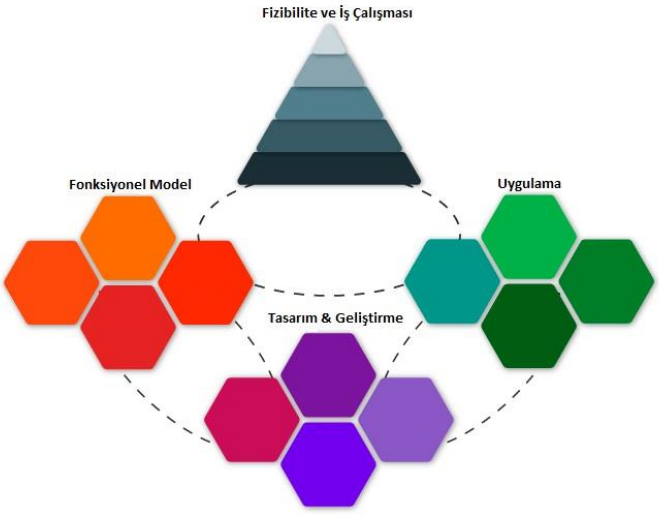
Rol	Tanımı
Proje Yöneticisi (Project Manager)	Projenin idarecisi olarak görev yapacak ve kaynak yönetiminden sorumlu olacaktır.
Yazılım Mimarı (Chief Architect)	Baş tasarımcı olarak görev yapacak ve çalışmanın tasarımını ilgili oturumlarda yürütecektir.
Kıdemli Yazılım Mühenhisi (Chief Programmer)	Gereksinimlere, tasarıma ve analizlere katılan deneyimli bir geliştiricidir.
Etki Alanı Uzmanı (Domain Expert)	Çeşitli gereksinimlerin nasıl uygulanması gerektiği konusunda bilgi sahibidir.
Etki Alanı Yöneticisi (Domain Manager)	Gereksinimler hakkında farklı görüşler konusunda etki alanı uzmanlarına liderlik eder.
Aktarım Yöneticisi (Release Manager)	İlerlemeyi kontrol eder ve ilgili kişilere raporlamasını sağlar.
Geliştirme Mühendisi (Build Engineer)	Derleme işlemini kurar, sürdürür ve çalıştırır.
Sistem Yöneticisi (System Administrator)	Sunucuları, iş istasyonlarını, geliştirme ve test ortamlarını yapılandırır, yönetir, olası sorunları giderir.
Test Uzmanı (Tester)	Sistemin gereksinimleri karşılayıp karşılamadığını kontrol eder. Fonksiyonel testleri yapar. Test sonuçlarını Proje Yöneticisine raporlar.
Teknik Analist (Technical Writer)	Kullanıcı dokümanlarını ve teknik analizleri yazar.

Kaynak: Agile Modeling (13 Nisan 2019) tarihinde http://agilemodeling.com/essays/fdd.htm 'den alındı. (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

2.3.6. Dinamik Sistem Geliştirme Metodolojisi

Dinamik Sistem Geliştirme Yöntemi (DSDM), Çevik proje yönetimi ve ürünlerin teslimi açısından kendini kanıtlamış bir yöntemdir. Hızlı ve etkili sonuçların alınmasına yardımcı olmaktadır. Yıllar içinde küçük yazılım geliştirmelerinden tam ölçekli iş süreçlerinin değişmesine kadar çok çeşitli projelere uygulanmıştır. Her ne kadar DSDM küçük ve basit projeler üzerinde kolay ve etkili bir şekilde çalışsa da, karmaşık iş dünyasında "yetişkin" bir yaklaşım sağlamak için her zaman kurumsal proje tabanlı çevreye odaklanmayı sürdürmüştür (Agile Business Consortium, 2014).

DSDM ilk olarak 1994 yılında, öncelikle iş odaklı bilgisayar çözümlerini geliştirdikleri gibi Hızlı Uygulama Geliştirme (RAD) süreçlerinde kaliteyi artırmak isteyen birçok şirkette çok sayıda proje uygulayıcısının işbirliği ile oluşturulmuştur (Agile Business Consortium, 2014).



Şekil 19. Dinamik Sistem Geliştirme Metodolojisi

Kaynak: Agile Business Consortium (16 Nisan 2019) tarihinde https://www.agilebusiness.org/what-is-dsdm 'den alındı.

DSDM felsefesi, geliştirmeyle alakalı "Müşterilerin iş gereksinimleri hakkındaki bilgilerini geliştirme topluluğunun teknik becerileri ile birleştiren bir ekip çalışmasıdır." şeklinde yorum yapmaktadır. Artımlı bir teslimat olmalı, müşteriye pazalanabilir bir ürün teslim edilmelidir. İleri tarihlerde asgari pazarlanabilir özelliğin oluşturulmasıyla teslimata devam edilmektedir. Yöntem model olarak fonksiyonel ve tasarım prototiplerine odaklanmıştır (Agile Business Consortium, 2014).

Jennifer Stapleton'a (2003) göre DSDM dokuz temel prensibe dayanmaktadır. Aşağıda detayları paylaşılan dokuz ilke, herhangi bir DSDM uygulaması için çok önemlidir; bunlardan birinin görmezden gelinmesi, yöntemi felsefesinden koparacak ve proje risklerini önemli ölçüde artıracaktır (METU, 2009: 2).

- Aktif kullanıcı katılımı zorunludur.
- Takımlar karar verme yetkisine sahip olmalıdır.
- Teslimatların sıklığına önem verilmelidir.
- Bir görevi kabul etmek için gerekli olan kriter onun iş amacına olan uygunluğudur.
- DSDM kullanıcıya geri bildirim veren, tekrarlanan ve artan gelişimler üzerine çalışır.
- DSDM yaklaşımında gelişim sırasındaki bütün zorluklar çözüme kavuşturulur.
- Yüksek seviyede karar almanın başlangıç aşamasında gerekli açıklamalar kabul edilir.
- Değerlendirci geliştirme yazılım üretim döngüsünün önemli bir parçasıdır.
- DSGM bütün ilgili taraflar arasında iş birlikçi ve paylaşımcı yaklaşımlar planlamaktadır.

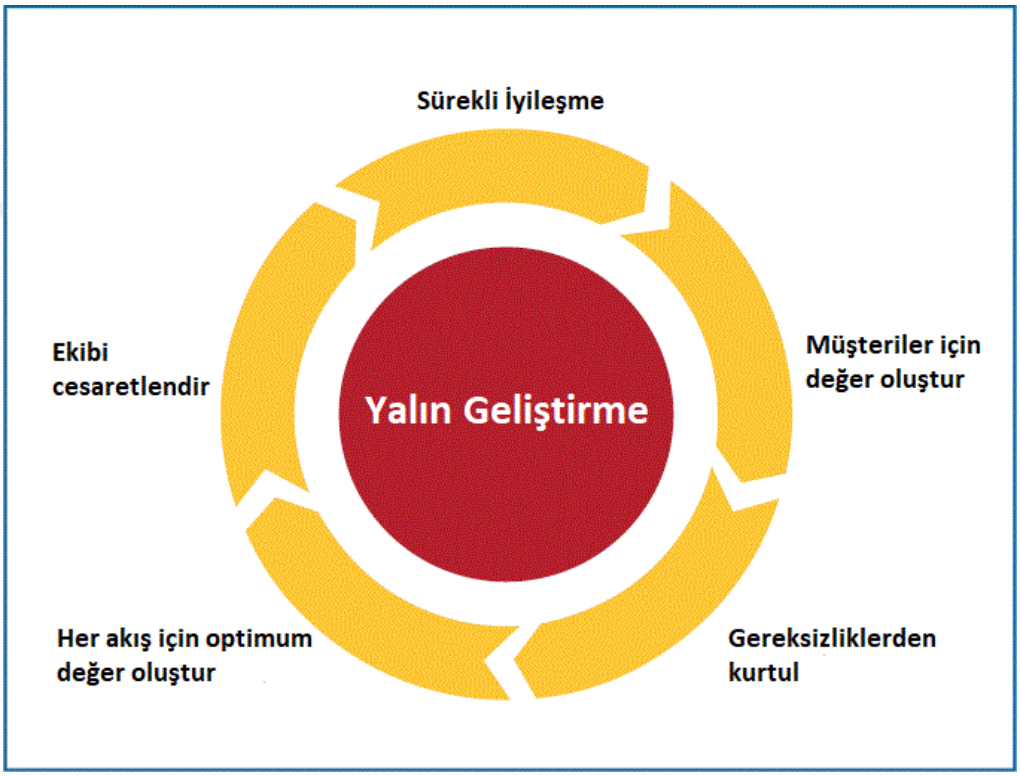
Bu modelin ana amacı: zamanı bütün yazılım gelişimi sürecinde tutumlu bir şekilde kullanılabilir olması, Prototip ve HUG modellerinde olduğu gibi ve yazılım ürünleri hızlı bir şekilde geliştirilebilir olmasıdır (METU, 2009: 2).

Bir başka tanımda DSDM konsorsiyumu tarafından bu yöntemin anahtar 10 başarı faktörü üye deneyimlerinden alınan geri bildirimlerle derlenmiştir (Voigt, 2014: 13).

- Çalışmaya başlamadan önce DSDM felsefesinin kabul edilmesi,
- Geliştirici ve kullanıcıların karar verme gücü,
- Son kullanıcı katılımı sağlama noktasında üst düzey kullanıcı yönetimi taahhüdü
- Artımlı teslimat
- Geliştiriciler tarafından son kullanıcılara kolay erişim
- İstikrarlı takım
- Geliştirme ekibi becerileri
- Geliştirme ekibinin büyüklüğü
- Destekleyici bir ticari ilişki
- Geliştirme teknolojisi

2.3.7. Yalın Sistem Geliştirme

Yalın Geliştirme felsefesi, yazılım geliştirmede risk liderliğine radikal bir değişime toleranslı yaklaşımın benimsenmesine dayanmaktadır. Yöntemin temel ilkesi insani çaba, geliştirme saatleri ve yeni bir pazar ortamına uyum sağlamak için araçlara ve yöntemlere yatırım yapılmasını içeren değişime dayanıklı yazılım oluşturmaktır (Poppendieck ve Poppendieck, 2003: 12).



Şekil 20. Yalın Yazılım Geliştirme

Kaynak: Agile Business Consortium (17 Nisan 2019) tarihinde https://www.agilebusiness.org/what-is-dsdm 'den alındı.

LD, yalın üretim yaklaşımı yalın düşünme kavramına dayanmaktadır. Bu yöntemin gerçeğe bakışı: "görünür müşteri değerinin hızla yaratılması, değişime dayanıklı yazılım oluşturulması, yalnızca gerekli işlevlerin oluşturulması, artık olmaması ve toplantılarda saldırganlık ve inatçılık yöntemin genel hedeflerine ulaşma inancıdır" şeklinde tanımlanmaktadır (Poppendieck ve Poppendieck, 2003: 13).

Yöntemde farklı teknikler ve araçlar tercih edilmektedir. Değer analizi süreci, her eserin değerini kanıtlamak için kullanılmaktadır. Ürünün hızlı teslimi, değişen gereksinimlerle rekabet eden başka bir tekniktir, teslimat ne kadar hızlı olursa gereksinimlerdeki değişim de o denli az yaşanmaktadır. Bu yöntemde takım boyutunda herhangi bir tanım bulunmamaktadır. LD, bir yöntemden çok bir yazılım geliştirme yönetim felsefesidir. Geliştirme son halini almadan önce müşteri talepleri artımlı olarak yayınlanabilmekte ve değişiklikler uygulanabilmektedir (Poppendieck ve Poppendieck, 2003: 14).

Bu yöntemde rol ve sorumluluklar için özel bir tanımlama bulunmamaktadır. Her projede olduğu gibi müşteri ve proje yöneticisi kavramlarını içinde barındırmaktadır (Highsmith, 2009: 62).

2.4. Geleneksel ve Çevik Yöntemlerin Karşılaştırılması

Tezin bu aşamasında detayları önceki bölümlerde anlatılan Geleneksel ve Çevik yöntemler farklı açılardan karşılaştırılacaktır.

Son yıllarda çevik yöntemler proje yönetimi ve yazılım geliştirme dünyasında adeta fırtına etkisi yaratmıştır (Forbes, 2019). Sektördeki her kuruluş bu proje yönetimi metodolojisinden bahsetmekte, kendi süreçlerine adapte etmeye uğraşmaktadır (CMS Wire, 2019). İş dünyası teknolojinin gelişme trendine bağlı bir şekilde hızla değişmekte, işletmeler bir işi sorunsuz bir şekilde yürütmelerine yardımcı olabilecek süreçleri, yaklaşımları ve metodolojileri aramaktadır. Onlarca farklı proje yönetimi yaklaşımı olsa da, işletmenin doğasını ve gereksinimlerini göz önünde bulundurarak nihai seçim yapılmalıdır. Tüm yöntemler gözden geçirildiğinde Geleneksel ve Çevik yöntemlerin genellikle birbirlerine karşı uygulandığı da görülebilmektedir.

Geleneksel proje yönetimi bilindiği üzere projelerin ardışık bir döngüde yürütüldüğü yerleşik bir metodolojidir. Başlangıç, planlama, yürütme, izleme ve kapanış şeklinde sabit bir sırayı takip etmektedir. Geleneksel proje yönetimi yaklaşımı, doğrusal süreçlere, belgelere, açık planlamaya ve önceliklendirmeye özel önem vermektedir. Geleneksel yönteme göre zaman ve bütçe değişkendir ve gereksinimler genellikle bütçe ve zaman çizelgesi sorunlarıyla karşı karşıya kalması nedeniyle belirlenmektedir (Rezaid, 2019).

Geleneksel yöntemlerin genel faydaları aşağıdaki şekilde ifade edilmektedir.

- Açıkça tanımlanmış hedefler
- Kontrol edilebilir süreçler
- Temiz dokümantasyon
- Daha fazla sorumluluk

Çevik yöntemler, yazılım geliştirme için kullanılan genel bir yaklaşım olsa da, ekip çalışmasına, işbirliğine, zaman kısıtlı görevlere ve değişime olabildiğince çabuk yanıt verebilme esnekliğine dayanmaktadır (Abrahamsson vd., 2002: 91).

Çevik manifesto dört önemli değere sahiptir. Bu değerler aşağıdaki şekilde ifade edilmektedir.

- Süreçlerden ve araçlardan ziyade bireylere ve etkileşimlere odaklanmanın önemini,
- Çalışan bir yazılımın kapsamlı dokümantasyondan daha önemli olduğu,
- Müşteriyle yapılan işbirliğinin pazarlıktan daha önemli olduğu,
- Bir planı körü körüne bir şekilde takip etmek yerine değişime cevap verilmesi gerektiği,

Çevik yöntemin genel faydaları aşağıdaki şekilde ifade edilmektedir.

- Önceliklendirme noktasında esneklik
- Erken ve öngörülebilir teslimat
- Tahmin edilebilir maliyetler ve programlar
- Arttırılabilir kalite
- Daha fazla şeffaflık

Çevik yaklaşımın en önemli uygulanan örneği Scrum'da projeler kısa süreli sprintlere bölünen yinelemeli bir süreci izlemektedir. Geleneksel yaklaşımın aksine, önceden planlamaya daha az zaman harcanır ve çeviklik, özelliklerdeki değişiklikler ve geliştirme açısından daha esnek olduğundan önceliklendirme yapılabilmektedir (Scrum, 2017).

2.4.1. Geleneksel ve Çevik Metodoloji Arasındaki Farklar

Aşağıdaki tabloda geleneksel ve çevik yöntemlerin arasındaki temel farklılıklar gösterilmektedir.

Tablo 8 : Metodolojiler Arasındaki Farklar

Özellikler	**Çevik Yaklaşım**	**Geleneksel Yaklaşım**
Örgütsel Yapı	Tekrarlı	Doğrusal
Proje Ölçeği	Küçük ve Orta Ölçekli	Büyük Ölçekli
Kullanıcı Gereksinimleri	İnteraktif	Uygulamadan Önce Açıkca Tanımlanmış
Müşterilerin Katılım Oranı	Yüksek	Düşük
Geliştirme Modeli	Evrimsel Teslimat	Yaşam Döngüsü
Müşteri Etkisi	Müşteriler işin yapıldığı süre boyunda etkilidirler	Müşteriler geliştirme başlamadan projenin başında devreye girer
Eskelasyon Yönetimi	Problem ortaya çıktığında, tüm ekip sorunu çözmek için birlikte çalışır	Problem ortaya çıktığında yöneticilere eskale edilir
Model Tercihi	Adaptasyon	Beklenti
Ürün veya İşlem	Resmi ve yönlendirici süreçlere daha az odaklanır	Süreçlere üründen daha fazla önem verilir
Test Belgeleri	Her sprint için ayrı test planlanır	Kapsamlı test planı yapılır
Tahmini Çaba	Scrum Uzmanı ekibin tahmin yapmasına olanak sağlar	Proje yönetici tahmin yapar ve PO tarafından onay alınır
Yorumlar ve Onaylar	İncelemeler her yinelemeden sonra yapılır	Liderler tarafından detaylı inceleme yapılır ve onay verilir

Kaynak: (Hirsch, 2005) (Tablonun Türkçeleştirilmesi Tarafımca Yapılmıştır.)

Çevik metodoloji, birçok geliştirici ve proje yöneticisi tarafından farklı nedenlerle tercih edilmektedir. Bu nedenlerden bazıları aşağıda başlıklar halinde detaylandırılmıştır. Tablo 8 de bazı özellikler açısından yöntemler birbiriyle kıyaslanmıştır. Bu karşılaştırmada yöntemlerin birbirinden ayrılan özellikleri tercih edilmiştir.

Daha Fazla Esneklik

Üründe veya süreçte değişiklik yapılması söz konusu olduğunda çevik metodoloji, şelale metodolojisinden çok daha esnektir. Çalışırken, ekip üyeleri bir şeyi deneyimlemeyi veya planlanandan farklı bir şeyi deneme ihtiyacı hissediyorsa, çevik metodoloji bunu kolayca yapmalarını sağlamaktadır. Bu metodolojinin en iyi özelliği, katı bir yöntem izlemekten ziyade ürüne daha fazla odaklanmasıdır. Geleneksel yaklaşımın aksine, çevik metodoloji doğrusal değildir, yukarıdan aşağıya bir yaklaşımı izler. Bu nedenle, sonucu etkilemeden ve proje programını aksatmadan son dakika değişiklikleri sürece dâhil edebilir.

Şeffaflık

Çevik metodolojide her şey ortada ve şeffaftır. Müşteriler ve karar vericiler, bir ürünün başlatılmasına, planlanmasına, gözden geçirilmesine ve test bölümüne aktif olarak katılırlar. Geleneksel yaklaşımda ise, projenin dizginlerini proje yöneticisi elinde bulunur, bu nedenle diğer ekip üyeleri önemli kararları verememektedir. Çevik metodoloji, ekip üyelerine ilerlemeyi başından sonuna kadar görme konusunda yardımcı olur. Bu şeffaflık düzeyi sağlıklı bir çalışma ortamı oluşturmak için önemli bir rol oynamaktadır.

Sahiplik ve Hesap Verebilirlik

Her iki proje yönetimi yaklaşımındaki çarpıcı farklılıklardan biri de, her birinin ekip üyelerine sağladığı sahiplik ve hesap verebilirlik düzeyidir. Geleneksel proje yönetiminde, bir proje yöneticisi geminin kaptanıdır, bütün sorumluluk kendisine aittir. Müşteriler ise planlama aşamasına da katılır, ancak katılımları uygulama başladığı anda sonlanmaktadır.

Çevik metodolojide, her ekip üyesi projenin sahipliğini paylaşır. Her biri, sprinti tahmini süre içinde tamamlamak için aktif bir rol oynamaktadır. Geleneksel proje yönetiminden farklı olarak, projeye dâhil olan herkes, başından sonuna kadar olan gelişmeleri kolayca görebilmektedir.

Geri Bildirim İçin Kapsam

Geleneksel yaklaşımda, her bir süreç açıkça projenin başlangıcından itibaren tanımlanmakta ve planlanmaktadır. Projenin tahmini süre ve bütçe dâhilinde tamamlanması beklenir. Bu nedenle, son teslim tarihini etkileyebilecek herhangi bir değişiklik veya geri bildirim dikkate alınmaz. Çevik yönetim ise daha iyi çıktı sağlamada yardımcı olan sürekli geri bildirime izin verir.

Çevik metodolojide geri bildirimlerin kabul edilme olanağının yüksek olması nedeniyle, birçok proje yöneticisi ve yazılım geliştiricisi için ilk tercih haline gelmiştir. Teslimat süresinde yüksek kaliteli bir ürün veya hizmet sunmalarını sağlayan her yinelemeyi doğrularken müşterilerinin isteklerine yanıt verebilirler.

Proje Karmaşıklığı

Doğrusal yaklaşımı nedeniyle, geleneksel proje yönetimi metodolojisi büyük ölçüde küçük veya daha az karmaşık projeler için kullanılır. Daha önce de tartışıldığı gibi, bu metodoloji ani değişikliklerin hayranı değildir ve takımı ilk kareye geri götüreceği için kesinlikle bunlardan kaçınır.

Çevik proje yönetim metolojisi, büyük ve karmaşık projeleri yönetmek için en iyi seçenek olabilir. Projenin birbirine bağlı birden fazla aşaması varsa ya da bir aşaması diğerlerine bağlıysa bu karmaşıklığı çözmek için seçilecek en doğru yöntemdir.

2.4.2. Doğru Yaklaşımın Seçim Yöntemi

Teorinin aksine gerçek hayatta her proje veya organizasyon için uygun "tek bedene uyan" bir metodoloji yoktur. Bir metodoloji uygulama seçimi büyük ölçüde projenin doğası, büyüklüğü, diğerleri arasında yer alan kaynaklar gibi faktörlere bağlıdır. Çoğu zaman, proje yöneticileri proje fikir aşamasında veya başlangıcında hangi metodolojinin uygulanacağına karar vermektedir. Proje yöneticisi, projeye dâhil olan kişiler ve proje sponsorlarıyla uyumlu bir şekilde kesin kararı verme hakkına sahiptir. Aşağıda, proje için doğru yöntemi seçerken dikkate alınabilecek bazı faktörler yer almaktadır.

- Proje gereksinimlerine göz atılır. Gereksinimler açık bir şekilde tanımlanmış mıdır? Proje gereklilikleri belirsizse veya değişme eğilimindeyse, çevik metodolojinin seçilmesi uygun olacaktır. Geleneksel metodoloji gereksinimlerin açıkça tanımlandığı ve ilk bakışta iyi anlaşıldığı bir durumda seçilebilir.
- Projede yer alan teknoloji incelenir. Yeni bir teknoloji veya araç yoksa geleneksel proje yönetimi metodolojisi daha uygundur. Çevik metodoloji, yeni teknolojilerle daha fazla deneme yapabilmek için daha esnektir.
- Proje istenmeyen risk ve tehditlere açık mı kontrol edilir. Geleneksel metodolojinin katı doğası göz önüne alındığında, tercih edilmesi önerilmez. Bununla birlikte, çevik yaklaşımda riskler daha erken ele alınabilir, risk yönetimi açısından daha iyi bir seçenek gibi görünmektedir.
- Bir diğer önemli faktör, kaynakların ulaşılabilir olmasıdır. Geleneksel yaklaşım, büyük, karmaşık ekipler ve projelerle en iyi şekilde çalışır. Çevik bir takım genellikle sınırlı sayıda deneyimli ekip üyesinden oluşur.
- Bir nihai ürünün kritikliği, seçilen proje yönetimi metodolojisinin doğasına bağlıdır. Geleneksel yöntem dokümantasyon içerdiğinden, çevik proje yönetimi metodolojisine kıyasla kritik ürünler için daha uygundur.

Yazılım geliştirme süreçlerinde kullanılan metodoloji zamana, uygulama alanına ve uygulanan projelere göre farklılık göstermektedir. Bu farklılığın sebebi anlık ihtiyaçları karşılama amaçlı olduğu gibi, uzun süreli geliştirme süreçlerine yol gösterecek nitelikte de olabilmektedir. Basit ya da karmaşık bir proje ele alındığında, projenin başlaması ve sonlanması (çoğu zaman başarılı olmamakla birlikte) arasında geçen zaman içinde proje ekipleri tarafından belirli bir gayret harcanmaktadır. Süreçleri basit, izlenebilir ve ölçülebilir şekilde yönetmek için farklı teknikler kullanılsa da tüm süreçler istenildiği şekilde başarılı olarak sonuçlanmamakta, hatta projelerin büyük bir çoğunluğu da karşılaşılan başarısızlık sonucunda iptal olmaktadır.

Yöntemlerin çeşitlenmesi ve ölçüm araçlarının da gelişmesiyle birlikte proje yönetim metodolojileri her açıdan karşılaştırılır hale gelmiştir. Konuyla ilgili "The Standish Group" tarafından belirli zaman aralıklarıyla hazırlanan KAOS Raporu (CHAOS Report) bu karşılaştırmayı farklı açılardan yapmaktadır.

2.4.3. KAOS Raporu (2015)

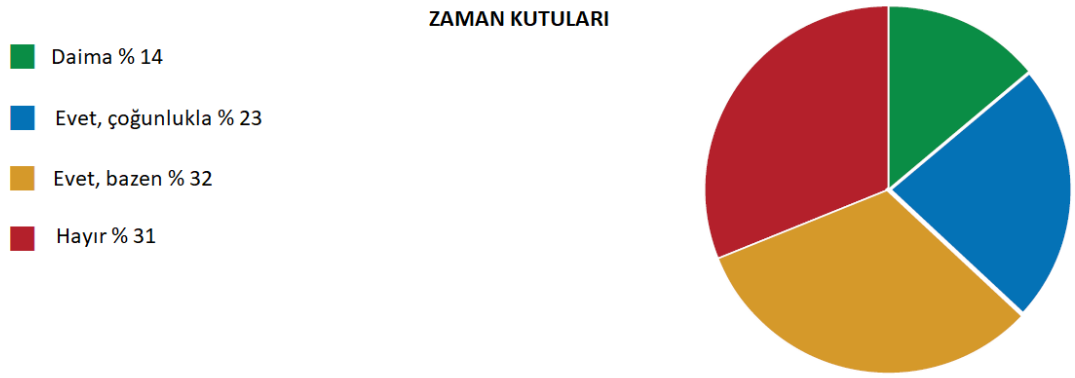
KAOS Raporu, Standish Group tarafından belirli dönemlerde, projelerin başarı/başarısızlık faktörlerini ele alan kapsamlı bir araştırmadır. Bu araştırma projelerin büyüklüğüne göre yöntemlerin başarılarını ölçümlemeyi amaçlamaktadır.

ÖLÇEK	METOT	BAŞARILI	PROBLEMLİ	İPTAL
Tüm Projeler	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Büyük Ölçekli Projeler	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Orta Ölçekli Projeler	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Küçük Ölçekli Projeler	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Şekil 21. KAOS Raporu (2015)

Kaynak: Standish Group (25 Nisan 2019) tarihinde https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf 'den alındı.

Tabloda (Şekil 17), çevik ve şelale yöntemine göre bölümlenmiş yeni KAOS veritabanında 2011-2015 mali yıllarına ait tüm yazılım projelerinin çözümlerini karşılaştırmaktadır. Rapora konu olan toplam yazılım projesi sayısı 10.000'den fazladır. Tüm projelerin sonuçları, çevik yöntemlerin şelale yöntemine kıyasla başarı oranının neredeyse dört katı olduğunu ve şelale yönteminin de çevik yöntemine kıyasla başarısızlık oranının üç katı olduğunu göstermektedir. Sonuçlar ayrıca proje büyüklüğüne (büyük, orta ve küçük) göre de ayrılabilir. Genel sonuçlar, şelale yönteminin iyi ölçeklenmediğini, çevik yöntemlerin ise daha iyi ölçeklendiğini açıkça göstermektedir. Bununla birlikte, projenin kapsamı ne kadar küçükse, çevik ve şelale yöntemi arasındaki farkın da o denli küçük olduğuna işaret etmektedir (Standish Group, 2015).



Şekil 22. "Zaman Kutuları" Anketi

Kaynak: Standish Group (25 Nisan 2019) tarihinde https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf 'den alındı.

Yine aynı raporda BT yöneticilerine uygulanan bir ankette projelerde zaman yönetimi ile alakalı soru yöneltilmiştir. Ankete konu soru "Projelerinizi optimize etmek için genel olarak "time-box" kullanıyor musunuz?" şeklindedir (Standish Group, 2015).

Ankete verilen yanıtlar incelendiğinde her üç katılımcıdan bir tanesi temsil ettiği kuruluşta bu tekniğin kullanılmadığını ifade etmiş, yine yaklaşık her üç katılımcıdan bir tanesi de zaman zaman bu tekniğin kullanıldığını ifade etmiştir. Tüm sonuçlara geniş bir perspektiften bakıldığında iki kuruluştan bir tanesinde zaman yönetimi noktasında geleneksel tekniklerin kullanılmaya devam edildiği görülebilmektedir. Çevik yöntemlerin kullanımı zamanla artsa da geleneksel yöntemle süreçlerini yürüten kuruluşlar göz ardı edilemeyecek ölçüde fazladır.

ÜÇÜNCÜ BÖLÜM
METODOLOJİ

Bu bölüm, araştırmanın yürütülmesinde kullanılan adımları ve araştırma yöntemini tanımlamaya ve tartışmaya yöneliktir.

3.1.Araştırmanın Amacı ve Önemi

Bu tez çalışmasının temel amacı çevik yazılım geliştirme yönteminin, yazılım geliştirme süreçlerinde firma performansına etkisini tespit etmektir. Teknolojide yaşanan gelişim ve değişim bu teknolojilerin geliştirilmesi aşamasında kullanılan teknik ve yöntemleri de etkilemektedir. Gelişen teknolojinin ve yazılım sistemlerinin etkisiyle elde edilen "hız" zaman içinde yazılım sektöründe çalışanların ve bu sektörden hizmet alan insanların da süreçlere bakış açısını değiştirmiştir. Süreçlerin ve uygulamaların geliştirilmesi aşamasında kullanılan yöntemler yazılım geliştirme kavramının ortaya çıktığı günden bu yana değişmekte ve sürekli kendini geliştirmektedir. Bu sebepten dolayı aradan geçen yıllar boyunca ele alınan projelerde farklı yöntemler ve teknikler tercih edilmiştir. Bu tercihin sebebi bazı durumlarda proje kapsamı olurken bazı durumlarda ise zaman kısıtından kaynaklanmaktadır. Yazılım geliştirme süreçlerinde tercih edilen yöntemlerin farklılaşması sonucunda bu süreçlerde görev alan çalışanlar birinci dereceden etkilenmektedir. Yazılım geliştirme süreçleri konusunda tüm dünyada olduğu gibi ülkemizde de birçok bilimsel çalışma olmasına rağmen, tercih edilen yönteme ilişkin çalışanların bakış açılarını ölçümleyen ve değerlendiren kapsamlı araştırmalar yapılmamıştır. Bu yüksek lisans tezinde yazılım geliştirme sürecinde tercih edilen farklı yöntemlere çalışanların bakış açısının gözlemlenmesi ve değerlendirilmesi bununla birlikte gelecek çalışmalara da katkı sağlaması beklenmektedir.

3.2.Araştırmanın Kapsamı ve Sınırları

Bu araştırma kapsamı ve sınırları İstanbul ilinde faaliyet gösteren teknoloji firmaları ve bankaların bilgi işlem departmanında görev yapan çalışanlara uygulanan anket ölçeğinin sonuçlarından oluşmaktadır. Araştırmada ölçülmek istenilen tez konusu çevik yazılım geliştirme yönteminin tercih edildiği işletmelerde bu yöntemin firma performansı üzerindeki etkisini ölçümlemektir. Uygulanan metodoloji şekil itibariyle bazı kısıtlamaları da beraberinde getirse de yapılan örneklem altı farklı teknoloji şirketinde uygulanmıştır. Bu şirketlerin iş

yapma şekilleri, organizasyon şemaları ve kurum kültürlerinin birbirlerinden farklı olduğu düşünüldüğünde geniş bir kitleye ulaşmak hedeflenmiştir. Ayrıca çalışmanın hedefinde bankacılık sektöründe faaliyet gösteren şirketlerin ve çalışanların olduğu, bu şirketlerin yazılım geliştirme bölümlerinin büyük bir bölümünün İstanbul ilinde faaliyet gösterdiği düşünülürse bu durum anket sonuçlarının etkisini arttıracaktır.

3.3.Araştırma Yöntemi

Araştırma kapsamında veri toplama metodu olarak anket yöntemi uygulanmıştır. Araştırmanın hedef kitlesi Türkiye'de faaliyet gösteren 6 adet banka ve teknoloji şirketi çalışanlarıdır. Bu kapsamda yazılım geliştirme sektöründe aktif olarak çalışmakta olan ilgili şirket çalışanlarına 01.01.2019 – 31.05.2019 tarihleri arasında toplamda 350 adet anket dağıtılmış, 260 adet geri dönüş alınmıştır. Geri dönüş alınan anketler incelendiğinde 203 adet anketin istatistiki analiz için yeterli olduğu tespit edilmiştir. Verilerin toplanmasında kullanılan anket formu, 5'li Likert şeklinde hazırlanmış olup; EK-1'de verilmektedir. Anketteki cevapların değerlendirilmesine ilişkin seçenekler şu şekilde sıralanmaktadır: (1) Kesinlikle katılmıyorum, (2) Katılmıyorum, (3) Kararsızım, (4) Katılıyorum, (5) Kesinlikle katılıyorum seçeneğini ifade etmektedir.

Toplanan veriler SPSS 21.0 istatistiksel veri analizi paket programı kullanılarak frekans dökümü, geçerlilik, güvenilirlik, korelasyon ve regresyon analizlerine tabi tutulmaktadır.

Ölçüm aracımız olan anket formunun bilimsel bir temele uygun olarak hazırlanmasında ilgili genel kabul gören bir takım kriterler ve hususlar dikkate alınmaktadır (Günsel 2017, 45-46-47):

- İlk olarak anket formunun giriş yazısında net ve anlaşılır bir dil ile araştırmanın içeriği, bilimsel ve sosyal faydaları üzerinde durulmaktadır.
- İkinci olarak katılımcılara anketin isimsiz olarak toplanacağı ve içeriğinin gizli tutulacağı bilgisi verilmektedir. Bu sayede katılımcıların çekinmeden ve rahat bir şekilde anketi doldurabilmeleri amaçlanmaktadır.
- Giriş yazısıyla birlikte araştırmayı yürüten kişilerin isimleri, unvanları ve iletişim bilgileri de anket formunda yer almaktadır.

- 5'li Likert ölçeğin doğru bir şekilde kullanılması için açıklayıcı ifadelere anket formunda yer verilmektedir.
- Ankette yer alan soru ifadelerinin basit, net ve yalın olmasına dikkat edilmektedir.
- Anket formunda yer alan sorular, 5'li Likert tipinde kapalı uçlu hazırlanmıştır.

Anket formu üç bölümden oluşmakta olup; ilk bölümde katılımcıların işletmelerine ait bir kısım özellikler (faaliyet düzeyi ve faaliyet alanı) ile sosyodemografik özelliklerini (departman, ünvan, yaş ve eğitim) ölçmekte olan beş adet soru yer almaktadır. İkinci bölümde Şahin (2016)'den uyarlanarak Çevik yöntemlerin verimliliğini ve etkinliğini ölçmek için 12 soru, katılımcıların sorulara verdiği cevapları kontrol etmek için 12 soru, katılımcıların Çevik yöntemleri kullanmak suretiyle herhangi bir görev alıp almadıklarını öğrenmek için 1 soru ve katılımcıların Çevik yöntemlerin avantajları ve dezavantajları hakkında yorumlarını almak için son olarak 2 adet açık soru yöneltilmiştir. Böylece araştırma kapsamında anket formundaki toplam soru sayısı 32 olarak karşımıza çıkmaktadır.

3.4.Değişkenler, Araştırma Modeli ve Hipotezler

Bu tez çalışmasında firma performansı bağımlı değişken, kapsam yönetimi, zaman yönetimi ve maliyet yönetimi bağımsız değişken olarak alınarak, çevik yazılım geliştirme yönteminin firma performansı üzerindeki etkileri irdelenmektedir. Araştırma kapsamında herhangi bir aracı değişkene yer verilmemektedir.

Araştırmanın Hipotezleri:

$\boldsymbol{H_1}$**:** Çevik proje geliştirme yönteminin firma performansına doğrudan ve pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1A}}$**:** Kapsam yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1B}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1C}}$**:** Maliyet yönetimi açısından çevik yöntemin tercih edilmesinin firma performansına pozitif yönde bir etkisi bulunmaktadır.

3.5.Bulgular ve Değerlendirme

Bu bölümde tez çalışmasının bulgularına ve değerlendirmesine yer verilmektedir. Bu bağlamda demografik verilere dair tanımlayıcı istatistikler, değişkenlerimize dair geçerlilik ve güvenilirlik testleri ve hipotez testleri detaylı bir şekilde sunulmaktadır.

Araştırma kapsamında öncelikle ankete katılan kişiler hakkında genel bilgiler yer almaktadır. Katılımcılara dair unvanı/statüsü, cinsiyeti, yaşı, mevcutta çalıştığı işyerindeki çalışma süresi ve yazılım geliştirme süreçlerindeki çalışma süresini gösteren toplam 5 adet demografik soru ölçüm formumuzda yer almaktadır. Araştırmaya dâhil olan 203 adet katılımcının demografik verilerinin değerlendirilmesine dair bulgular, aşağıdaki tablo ve açıklamalarda sunulmaktadır.

Tablo 9: Ankete Cevap Veren Çalışanların Ünvan/Görev Tanımları

Katılımcı Grubu	**# Katılımcılar**
Yönetici	28
İş Analisti	70
Test Uzmanı	6
Yazılım Mühendisi / Mimarı	68
Proje Yöneticisi	10
Scrum Uzmanı	5
Agile Koçu	4
Diğer	14
Toplam	203

Ankete katılan çalışanların sahip oldukları ünvan/görev tanımları dağılımı tablo 8'de gösterilmektedir. Anket katılan 28 kişi (%13) Yönetici, 70 kişi (%34) Analist, 6 kişi (%3) Test Uzmanı, 68 kişi (%33) Yazılım Mühendisi / Mimarı, 10 kişi (%34) Proje Yöneticisi, 5 kişi (%3) Scrum Uzmanı, 4 kişi (%2) Agile Koçu pozisyonunda görev almaktaktadır. 14 kişi de (%7) görev aldığı pozisyonu diğer seçeneği ile belirtmiştir. Ankete katılan çalışanların çoğunluğunun (%67)'ile analist ve yazılımcılardan oluşuyor olması, yazılım geliştirme süreçlerinde en çok zaman geçiren kişiler olan bu çalışanların yaklaşımlarının tespit edilmesinde faydalı bir çoğunluğu temsil etmektedir.

Tablo 10: Ankete Cevap Veren Katılımcıların Cinsiyet Oranları

Katılımcı Grubu	**# Katılımcılar**
Erkek	143
Kadın	62
Toplam	203

Ankete katılanların cinsiyet dağılımı tablo 9.'da gösterilmektedir. Ankete katılanlardan 143 kişi (%70) erkek, 62 kişi (%30) ise kadın çalışanları oluşturmaktadır. Dolayısıyla örneklemimiz erkek yoğun bir yapı sergilemektedir. Katılımcıların yaş ortalaması ise 25-30 aralığında hesaplanmıştır. Dolayısıyla karşımıza erkek egemen ve 25-30 yaş aralığında çalışma hayatında belirli bir deneyim kazanmış genç bir örneklem profili çıkmaktadır.

3.5.1 Geçerlilik ve Güvenilirlik

Bu çalışmada ölçümlerimizin geçerlilik ve güvenilirliklerini test etmek üzere faktör analizi ve "Cronbach Alfa" analizlerinden yararlanılmıştır. Geçerlilik, ölçümlerin gerçekten ölçmek istediği şeyi ölçüp ölçmediğini, güvenilirlik ise ölçümlerin ölçmek istediği şeyi doğru ölçüp ölçmediğini ortaya koyan kriterler olarak ifade edilebilir.

Araştırma kapsamında toplam 203 katılımcıdan elde edilen anket sonuçları SPSS programı ile güvenilirlik analizine tabi tutulmuş ve 24 soru için genel güvenilirlik düzeyi 0,862 olarak belirlenmiş olup; bu değer eşik noktası 0,700'ün üzerindedir. Ölçeğin tümünün dışında her değişken teker teker güvenilirlik analizlerine tabi tutulmuştur. Genel olarak tüm anket ve her bir değişkene dair hesaplanan güvenilirlik katsayıları Tablo 11 ve Tablo 12'de gösterilmektedir.

Tablo 11: Anket Genelinin Güvenilirlik Analizi

Güvenilirlik	
Cronbach's Alpha	Madde Sayısı
0,862	24
Z"Anket Sayısı	N
Geçerli	203

Tablo 12: Faktör Bazlı Güvenilirlik Analizi

Güvenilirlik Analizi		
Faktör	**Madde Sayısı**	**Cronbach's Alpha**
Kapsam Yönetimi	8	0,854
Zaman Yönetimi	6	0,786
Maliyet Yönetimi	10	0,832

Tablo 12 'de görüleceği üzere tüm bağımsız değişkenlerin güvenilirlik değerleri eşik değerin üzerinde çıkmıştır. Dolayısıyla, ölçek hem değişkenleriyle hem de genel olarak anlamlı ve güvenilir olarak kabul edilmektedir.

Tablo 13: Faktör Bazlı Güvenilirlik Analizi

Cronbach's Alpha	**Güvenilirlik**
$\alpha \geq .9$	Mükemmel
$.9 > \alpha \geq .8$	İyi
$.8 > \alpha \geq .7$	Kabul edilebilir
$.7 > \alpha \geq .6$	Şüpheli
$.6 > \alpha \geq .5$	Kötü
$.5 > \alpha$	Güvenilmez

Kaynak: Crombach L. J. Coefficient Alpha and Internal Structure of Tests. Psychometrika 1951, 16, 311

Tablo 13'te gösterildiği üzere bu tez çalışmasında hesaplanan Cronbach's Alpha değeri 0.900 ile 0.800 arasında bulunduğu için "İyi" olarak derecelendirilmiş, güvenilir bulunmuştur.

3.5.2 Faktör Analizi

Geçerlilik kapsamında verileri değerlendirmek için kullanılan faktör analizi, anket formundaki soruların gerçekten kendi faktörleri (değişkenler) altında toplandığı bilgisini tespit etmektedir. Faktör analizinin geçerliliğinin sağlanması için her sorunun kendi faktörü altında toplanması gerekmektedir. Bu tezin bağımlı ve bağımsız değişkenlerine yapılan faktör analiz bulgularına göre "Kaiser-Meyer-Olkin (KMO)" ölçümü 0,748 olarak hesaplanmıştır. Bu değer, eşik değeri 0,600`ın üstünde olduğu için verilerin faktör analizine uygun olduğu sonucuna varılmıştır. Bu çerçevede anketi dolduran katılımcıların soruları bilinçli bir şekilde okuduğu ve anladığı, birbirine benzeyen sorulara aynı doğrultuda cevap verdiği gözlemlenmiştir. Böylece araştırma modelinin doğru bir şekilde kurulduğu ve bir sonraki adımda gerçekleştirilecek olan sebep-sonuç ilişkisinin araştırma modeline atıf da bulunacağı ortaya çıkmaktadır.

Bu anketin verileri, SPSS 21.0 istatistiksel veri analizi paket programı ile keşifsel faktör analizine (EFA) tabi tutulmuştur. Bağımsız değişkenleri oluşturan kapsam yönetimi, zaman yönetimi ve maliyet yönetimi bir EFA modeline; bağımlı değişken firma performansı ise bir diğer EFA modeli içine dâhil edilmiştir. Tablo 14 ve tablo 15`te ortaya çıkan faktör dağılımları gözükmektedir.

Tablo 14: Bağımsız Değişkenlere Dair Faktör Analiz Sonuçları

	Faktörler		
	1	**2**	**3**
Zaman Yönetimi			
Çevik yöntemler, yazılım gereksinimlerindeki hızlı değişikliklere cevap verme noktasında daha iyidir.	0,813		
Çevik yöntemler ile projeler daha uzun zamanda tamamlanmaktadır.	0,835		
Çevik yöntemler, proje süresinin azaltılmasına yardımcı olur.	0,897		
Diğer yazılım geliştirme metodolojileri, yazılım gereksinimlerindeki hızlı değişikliklere cevap verme noktasında Çevik yöntemlerden daha iyidir.	0,775		
Çevik yöntemleri uygulamak suretiyle çalışmaya devam etmek her zaman yararlıdır.	0,730		
Çevik yöntemlerin uygulandığı ekipler, zaman yönetimini daha başarılı uygulamaktadır.	0,825		

Kapsam Yönetimi			
Çevik yöntemler, bankacılık uygulamalarında COBIT ve ITIL standartlarıyla uyumludur.		0,761	
Genel olarak (avantajlar, eksiklikler) geleneksel yöntemler, Çevik yöntemlerden çok daha iyidir.		0,540	
Çevik yöntemler şeffaflığı sağlar. Ne yapıldığını ve projedeki sorunların ne olduğunu görebiliyorum.		0,650	
Çevik yöntemleri uygulayan ekipler, kendi kendine motive olamazlar ve takım dışından herhangi bir yardım almadan kendi başlarına bir şey öğrenemeyen kendi kendilerine organize olmayan ekiplerdir.		0,553	
Çevik yöntemleri bankacılık uygulamalarında COBIT ve ITIL standartlarıyla uyumlu değildir.		0,580	
Çevik yöntemler şeffaflığı sağlamaz. Ne yapıldığını, ne yapılacağını, projede sorunlarında ne olduğunu göremiyorum.		0,610	
Genel olarak (avantajlar, eksiklikler) Çevik yöntemler, geleneksel yöntemlerden çok daha iyidir.		0,830	
Çevik yöntemlerin uygulandığı ekiplerde, ekip üyeleri metodolojiyi çok iyi tanıyor, her şeye tam destek veriyor ve her türlü gereksinimi yapıyorlar.		0,860	
Maliyet Yönetimi			
Çevik yöntemleri uygulayan ekipler, ekip dışından herhangi bir yardım almadan kendi başlarına bir şey öğrenen, kendi kendilerini motive eden ve kendi kendilerini organize eden ekiplerdir.			0,697
Yönetim, BT projelerinde Çevik yöntemlerin kullanılmasını destekliyor.			0,776
Çevik yöntemlerin uygulandığı ekiplerde yardımlaşma, koordinasyon ve diğer ekiplerle işbirliğinde gözle görülür bir seviyede artış gözlenir.			0,820
Çevik yöntemlerin uygulandığı projelerde, projeyi daha nitelikli ve amaçlarına daha uygun hale getirir.			0,625
Çevik yöntemleri uygulayan ekipler, kendi kendine motive olamazlar ve takım dışından herhangi bir yardım almadan kendi başlarına bir şey öğrenemeyen kendi kendilerine organize olmayan ekiplerdir.			0,588
Yönetim, BT projelerinde Çevik yöntemlerin kullanılmasını desteklemez.			0,545
Çevik yöntemlerin uygulandığı ekiplerde yardımlaşma, koordinasyon ve diğer ekiplerle işbirliğinde azalış gözlenir.			0,530
Çevik yöntemler, projelerin kalitesini düşürür ve aynı zamanda gereksinimleri de karşılayamazlar.			0,615

Tablo 15: Bağımsız Değişkenlere Dair Faktör Analiz Sonuçları

Firma Performansı	Faktör
	1
Çevik yöntemlerin "Zaman Yönetimi" açısından performansa etkisi	0,890
Çevik yöntemlerin "Kapsam Yönetimi" açısından performansa etkisi	0,723
Çevik yöntemlerin "Maliyet Yönetimi" açısından performansa etkisi	0,630
Genel olarak firma performansına olan etkisi	0,747

3.5.3 Korelasyon Analizi

Araştırmada pazarlama stratejileri ve örgüt yapısının firma performansı üzerindeki etkilerine ait değişkenlerin güvenilirlik analizleri test edildikten sonra, her bir faktöre yüklenen değişkenlerin ortalamaları alınmıştır. Regresyon analizine geçmeden hemen önce, faktörler arasındaki bire bir ilişkileri inceleyen korelasyon analizleri uygulanmıştır.

Korelasyon analizi; iki değişken arasındaki doğrusal ilişkiyi veya bir değişkenin iki veya daha çok değişken ile olan ilişkisini test etmek, varsa bu ilişkinin derecesini ölçmek için kullanılan istatistiksel bir yöntemdir. Korelasyon analizinde amaç bağımsız değişken değiştiğinde, bağımlı değişkenin hangi yönde değişeceğini belirlemektir. Korelasyon analizi sonucunda, doğrusal ilişki olup olmadığı varsa bu ilişkinin derecesi korelasyon katsayısı ile hesaplanır. Korelasyon katsayısı (-1) ile (+1) arasında değerler alır. Korelasyon, neden-sonuç ilişkisi anlamına gelmez. Farklı durumlar için farklı korelasyon katsayıları geliştirilmiştir. Bunlardan en iyi bilineni ve sosyal bilimlerde en sık kullanılanı; "Pearson'un Korelasyon Katsayısı"dır. Pearson korelasyon katsayısı, iki sürekli değişkenin doğrusal ilişkisinin derecesinin ölçümünde kullanılır. Yani iki değişken arasında anlamlı bir ilişki var mıdır sorusunun cevabı alınır.

Pearson Korelasyon Katsayısı;

- r= -1 ise tam negatif doğrusal ilişki vardır. Yani bir değişken artarken diğeri tersine azalır, tersine bir değişken azalırken diğeri artar.
- r= 1 ise tam pozitif doğrusal ilişki vardır. Yani bir değişken arttığında diğeri de artar, bir değişken azaldığında diğeri de azalır.
- r= 0 ise iki değişken arasında ilişki yoktur.

Korelasyon analizi sonuçları, regresyon analizi hakkında da ön fikir beyan etmeleri açısından önemlidir. Diğer bir deyişle; regresyon analizleri için araştırma modelindeki ilişkilerin bir anlamda test edilmesi amacıyla da kullanılabilir. Diğer tüm faktörler sabitken, her bir korelasyon katsayısını paylaşan iki faktör arasında pozitif ya da negatif yönde bir ilişkinin varlığından söz edilebilir. İki değişken arasındaki Pearson korelasyon katsayısının düzeyleri aşağıdaki Tablo 16'da gösterilmektedir (Gürer 2017, 74-75):

Tablo 16: Korelasyon Katsayı Düzeyleri

r	**İlişki**
0.00 – 0.25	Çok Zayıf
0.26 – 0.49	Zayıf
0.50 – 0.69	Orta
0.70 – 0.89	Yüksek
0.90 – 1.00	Çok Yüksek

Kaynak: Gürer S., Türk Sağlık Hizmetlerinde Yalın Yönetim İncelemesi: Karadeniz Bölgesi'nde Bir Uygulama, Yayınlanmamış Yüksek Lisans Tezi, Beykent Üniversitesi SBE, 2017

Tablo 17: Korelasyon Katsayı Düzeyleri

FAKTÖRLER	**Zaman Yönetimi**	**Kapsam Yönetimi**	**Maliyet Yönetimi**	**Firma Performansı**
Zaman Yönetimi	1	.567**	.236**	.378**
Kapsam Yönetimi		1	.198**	.311**
Maliyet Yönetimi			1	.332**
Firma Performansı				1

Tablo 17'de yer alan (**) işaretli faktörler arasındaki birebir ilişkiler p<0,01 (%1) düzeyinde istatistiksel olarak anlamlıdır. Korelasyon tablosunun en önemli göstergesi, bağımsız değişkenler ile çakışmanın bağımlı değişkeni arasındaki ilişki katsayısıdır. Bu açıdan bakıldığında, tüm değişkenlerin performans ile ilişkili olduğu ortaya çıkmıştır. Ayrıca, herhangi bir otokorelasyon sorunu olmaması için tüm değişkenlerin kendi arasındaki ilişki yükleri kontrol edilmiştir. Yapılan analiz sonucu, her değişkeni ayrı bir faktör yapısı oluşturduğu için soru gruplarının birbirinden tamamen ayrıştığı tespit edilmiştir.

3.5.4 Regresyon Analizi

Regresyon analizi, iki ya da daha çok değişken arasındaki ilişkiyi ölçmek için kullanılan bir analiz çeşididir. Bu analiz ile değişkenler arasındaki ilişkinin varlığı ve ilişki varsa bu ilişkinin gücü hakkında bilgi edinilebilir. Burada kullanılacak olan regresyon analizi, firma performansı olarak adlandırılan bağımlı değişkenlerle zaman yönetimi, kapsam yönetimi ve maliyet yönetimi faktörlerinin bağımsız değişkenleri arasında gerçekleştirilecektir. Burada önemle ele alınması gereken hususlar şunlardır:

- "P" aralığı 0,05'ten küçük olmalıdır.
- "t" değeri yaklaşık olarak -2000 ya da +2000'den fazla olmalıdır.
- "β" katsayısı yaklaşık olarak (+) ya da (-) 150 ve üzeri olmalıdır.

Bu şartlar sağlanıyorsa R^2 değerine bakılarak analiz gerçekleştirilir. R^2, modeldeki bağımsız değişkenlerin bağımlı değişkenleri ne oranda ölçtüğünü gözlemlemeye yarayan bir değerdir (Coşar, 2013: 118).

Araştırmada kurulan modelde bağımlı değişken firma performansı üzerinde bağımsız değişkenler olan zaman yönetimi, kapsam yönetimi ve maliyet yönetimi faktörlerinin etkileri incelenmiştir. Modelde birden fazla bağımsız değişken olması nedeniyle çoklu regresyon kullanılmıştır.

Regresyon analizinde amaç; araştırma hipotezinde de belirtilen zaman yönetimi, kapsam yönetimi ve maliyet yönetimi konularının firma performansı üzerine etkilerini incelemek olmuştur.

Tablo 18: Regresyon Analizi İçin Model Özeti

	Standardize Edilmemiş Katsayılar	**Standardize Katsayılar**		**t**	**Sig.**
Model	**B**	**Std. Hata**	**Beta**		
Sabit	0.512	0.455		1.123	0.23
Zaman Yönetimi	0.223	0.03	0.205	2.236	0.035
Kapsam Yönetimi	0.146	0.068	0.154	2.725	0.008
Maliyet Yönetimi	0.05	0.055	0.174	2.499	0.016
Firma Performansı	0.21	0.072	0.205	3.12	0.005
Bağımlı Değişken: Firma Performansı					
Model Özeti	**R**	**R²**	**Düzeltilmiş R²**	**Tahmini Standart Hata**	**"Anova" Kareler Toplamı**
1	.489^a	0,213	0,213	1.14965	12.786

Korelasyon analizinden sonra gerçekleştirilen regresyon analizinde bağımlı değişken olarak firma performansı, bağımsız değişkenler olarak da zaman yönetimi, kapsam yönetimi ve maliyet yönetimi alt boyutları analize dahil edilmiştir. Regresyon analizi modeldeki sebep-sonuç ilişkisini inceleyen ve araştırma modelinin bağımlı değişken ne ölçüde açıklayabildiğini ortaya koyan ikinci düzey bir analizdir.

Korelasyon analizinden farklı olarak bire bir ilişkileri göstermez, tüm faktörleri aynı anda değerlendirmeye alır. Tablo 18'de görüleceği üzere tüm bağımsız değişkenlerin firma performansı üzerinden asgari düzeyde etkisi bulunmaktadır. Diğer bir değişle araştırmanın bağımlı değişkeni olan firma performansındaki değişimlerin %21,3'ü modeldeki bağımsız değişkenler tarafından açıklanmaktadır. Bu oran, sosyal bilimler için çok yüksek bir seviyedir. Çalışmanın performans üzerindeki yaklaşık %20'lik değişimi açıklaması, araştırma modelinin doğru tasarlandığının da bir kanıtıdır.

Tablo 19: Araştırma Hipotezlerinin Kabul Çizelgesi

	BAĞIMLI DEĞİŞKEN - Firma Performansı -	
BAĞIMSIZ DEĞİŞKENLER	Hipotez	Sonuç
Çevik Yöntem Etkisi	$\boldsymbol{H_1}$	Desteklenmiştir.
Zaman Yönetimi	$\boldsymbol{H_{1A}}$	Desteklenmiştir.
Kapsam Yönetimi	$\boldsymbol{H_{1B}}$	Desteklenmiştir.
Maliyet Yönetimi	$\boldsymbol{H_{1C}}$	Kısmen Desteklenmiştir.

$\boldsymbol{H_1}$**:** Çevik proje geliştirme yönteminin firma performansına doğrudan ve pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1A}}$**:** Kapsam yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve pozitif yönde etkilemektedir.

$\boldsymbol{H_{1B}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve pozitif yönde etkilemektedir.

$\boldsymbol{H_{1C}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve kısmen etkilemektedir.

SONUÇ

Değişim ve yeniliğin sürekli hale geldiği, "hız" kavramının iş ve özel hayatın merkezinde olduğu rekabetçi bir dönemde yaşanmaktadır. İş hayatında yeni değerler oluşturmak, mevcut uygulamaları geliştirmek ve çağın gereksinimlerini karşılamak için farklı özellikteki tüm firmalar yüksek gayret göstermektedir. Yapılan çalışmalarda kullanılan yöntem ve teknikler kimi zaman gereksinimleri karşılamak için yeterli olsa da değişen şartlarla kimi zaman eksik kalabilmektedir. Yıllar boyunca ele alınan projelerde, geleneksel yöntemler tercih edilmiş, bu durum beraberinde bazı sorunları ortaya çıkarmıştır. Günümüz dünyasında müşterilerin ihtiyaçları ve istekleri daha sık değişmekte, bu değişime yanıt veremeyen işletmelerde/kuruluşlarda bazı sorunlar yaşanabilmektedir. Geleneksel yöntemler, müşteri gereksinimlerindeki ani değişikliklere cevap verme noktasında eksik kalmakta, bu sebepten dolayı kapsam değişikliği veya bütçe sıkıntıları nedeniyle süreçler başarısızlıkla sonuçlanmaktadır. Bu sıkıntılara çözüm olması amacıyla konunun uzmanları tarafından yeni bir yaklaşım ortaya atılmış, bu ilkeler "Agile Manifesto" adıyla dünyayla paylaşılmış, bu manifestoyu temel alan birçok yöntem geliştirilmiştir. BT uzmanları tarafından Çevik yöntemlerin ani değişikliklere cevap verme noktasında sorunları çözebileceği, projelerin daha başarılı sonuçlanacağı ve paydaşların da bu sonuçtan memnun olacağı ifade edilmektedir.

Çalışmanın temel amacı çevik yazılım geliştirme yönteminin kullanıldığı kuruluşlarda bu yöntemin uygulanması sonucunda firma performansının nasıl etkilendiğini ölçümlemektir. Bu doğrultuda yapılan deneysel çalışmalar neticesinde çevik yöntemin kullanılması durumunda zaman ve kapsam yönetimi açısından yöntemin firma performansı üzerinde doğrudan ve pozitif yönde katkı sağladığı söylenebilir.

Proje yönetimi yaklaşımında birbiriyle doğrudan ilişkili üç ana kavram bulunmaktadır. Bu kavramlar "zaman", "kapsam" ve "maliyet" başlığı altında toplanmaktadır. Bu kavramların işlevselliği ancak bir diğeri ile doğru bir şekilde kurduğu bağ zaman mümkün olabilmektedir. Bugün birçok projenin başarılı bir şekilde sonuçlandırılamamasının nedeni bu bileşenlere ait çalışmaların doğru bir şekilde yapılmamasından kaynaklanmaktadır. Proje yöntecileri ve paydaşların temel amacı bu kaynakların en optimal şekilde kullanılmasını sağlamaktır.

Tezin ana konu başlığını oluşturan kavram; "çevik yöntem" yazılım geliştirme süreçlerinde yıllar içinde yaşanan değişiklik ve elde edilen birikimlerin sonucunda ortaya çıkmıştır. Yazılım geliştirme süreçlerinde kullanılan teknikler karşılaşılan zorluklara ve iş

yapma şekillerindeki değişikliklere adapte olmaya zorlanmakta, bu değişikliği kendisine yeni bir özellik olarak eklemeyi başaran yöntemler ise bu süreçten faydalı çıkmaktadır. Yazılım geliştirme süreçlerin elde edilmeye çalışılan edinim aslında bu süreçlerde tercih edilen yöntemler ve tekniklerden beslenmektedir. Yeni bir uygulama geliştirme aşamasında ihtiyaçlara, zamana ve mekana en uygun yöntemin seçilmiş olması sonucunda optimal dengenin sağlanması ve bunun sonucunda performansın da en üst düzeyde seyretmesi anlamına gelecektir. Bu süreçlere basit bir çerçeveden bakarsak; belirli bir kapsamda ortaya konan kaynakların (zaman ve maliyet) en optimal kullanımı sonucunda en büyük kazanım elde edilmeye çalışılmaktadır. Daha net bir ifadeyle, yazılım geliştirme süreçlerinde yenilikleri takip eden, kendi dinamiklerini, çalışanlarını tanıyan ve buna uygun şekilde yöntem belirleyen kuruluşlar işletme performansını arttırabilmektedir.

Bu kavramlar içerisinde "maliyet yönetimi" boyutunun bu tez çalışmasının sonucuna göre bağımlı değişken üzerindeki etki gücünün diğer etkenlere göre düşük olmasının en temel sebebi, teknolojik yatırımların ve insan gücünün verimli kullanılması uğraşından kaynaklanmaktadır. Bu dengenin sağlanması ve verimliliğin yüksek tutulmaya çalışıldığı zamanlarda diğer etkenlere olumsuz etki verebilme ihtimalinden dolayı "maliyet yönetimi" üzerinde durulması gereken hassas bir konu olarak düşünülmektedir.

Tezin ana konusuna ait yapılan analizlerden sonra ortaya çıkan hipotezlerimiz aşağıdaki gibidir;

$\boldsymbol{H_1}$**:** Çevik proje geliştirme yönteminin firma performansına doğrudan ve pozitif yönde bir etkisi bulunmaktadır.

$\boldsymbol{H_{1A}}$**:** Kapsam yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve pozitif yönde etkilemektedir.

$\boldsymbol{H_{1B}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve pozitif yönde etkilemektedir.

$\boldsymbol{H_{1C}}$**:** Zaman yönetimi açısından çevik yöntemin tercih edilmesi firma performansını doğrudan ve kısmen etkilemektedir.

Çalışmayı bilimsel olarak değerlendirmek isteyen proje yöneticileri ve yazılım geliştirme süreç çalışanlarına ithafen bu tez içerisinde yazılım geliştirme yöntemleri ve performansa etkileriyle ile ilgili önemli özelliklere yer verilmiş olup, kavramların ehemmiyetini vurgulayan literatürler detaylı bir şekilde incelenerek çalışmaya eklenmiştir. Yapılan bu tez çalışması daha

farklı iş sektörlerinde, örneğin hizmet, sağlık sektörü vb. işletmeler ve bu işletmelerin çalışanlarına yapılacak örneklemler ile daha kapsamlı hale getirilebilir. Örneklem sayısının yapılan anket sayısı bakımından arttırılması da sonuçların daha tutarlı olması bakımından da fayda sağlayacaktır.

Araştırma sonucu kapsamında benzer sektörlerde faaliyet gösteren ve satış ve pazarlama yaparak faaliyetlerini sürdüren işletmelerin yöneticilerine şu önerilerde bulunabilir;

- Yazılım geliştirme süreçlerinde farklı yöntemlerin denenmesiyle insanlardan çalışma alışkanlıklarını değiştirmesi beklenmektedir. Böyle bir durumla karşılaşıldığında bu değişim süreçlerini yaşayan kişilerden tecrübelerini aktarmaları için yardım alınması süreci kolaylaştıran bir etken olacaktır.
- Geleneksel yöntemler genellikle büyük ölçekli, çevik yöntemler ise orta ve küçük ölçekli projelerde tercih edilmektedir. Bu sebepten dolayı projeye başlamadan önce ölçeği doğru tespit edilmeli, projenin ihtiyaçlarına en uygun yöntem uygulanmalıdır.
- Metodolojiler teorik olarak detaylı bir şekilde anlatılmış olsa bile farklı iki kuruluşta aynı metodoloji aynı şekilde uygulanamayabilir. Yapılması gereken en önemli şey uygulanması düşünülen yöntemin temel özelliklerini kaybetmeden ilgili kuruluşa göre özelleştirilmesidir.
- Türkiye'de bankacılık sektörünün denetim standartları, COBIT ve ITIL olarak belirlenmiştir. Çevik yöntemler ise “Agile Manifesto” ile belirlenen standartları temel kabul etmektedir. Bu sebepten standartların farklılaştığı noktada süreç olumsuz etkilenmeyecek ise esneme ve istisna uygulanabilir. Aynı şekilde standartlarında gerektiğinde gözden geçirilmesi, değişime ve gelişime ayak uydurması daha iyi bir çalışma ortamının sağlanması açısından yararlı olacaktır.

KAYNAKÇA

Abrahamsson P., Salo O., Ronkainen J., Warsta J., Agile Software Development Methods, 2002, 13-91

Awad M. A., *A Comparison between Agile and Traditional Software Development Methodologies,* 2005

Barjtya S., Sharma A., Rani U., *A detailed study of Software Development Life Cycle (SDLC) Models*, 2017, 2

Beck K., *Extreme Programming Explained, Embrace Change,* 1999, 1-19

Bell T. E., Thayer T. A., *Software requirements: Are they really a problem?,* 1976, 62

Benington H. D., *Production of Large Computer Programs, IEEE Annals of the History of Computing. IEEE Educational Activities Department,* 1983

Boehm B., Turner R., *Balancing Agility and Discipline: A Guide for the Perplexed,* 2003, 16

Boehm B., *A Spiral Model of Software Development and Enhancement*, TRW Defense Systems Group, 1988, 7-20

Boehm B., Jain A., *A Value-Based Theory of Systems Engineering,* 2006,

Booch G., *The Unified Modeling Language User Guide 2nd Edition,* 1999, 78-80

Booch G., *Object-Oriented Analysis and Design,* ADDISON-WESLEY, Santa Clara, California, 2006, 45-50

Cleland D. I., Gareis R., *Global Project Management Handbook,* 2006, 4-5

Cockburn A., *Agile Software Development: The Agile Software Development Series,* 2000, 35

Cohn M., *Succeeding with Agile: Software Development Using Scrum*, Boston: Addison-Wesley Professional, 2009, 175-177

Cressler J. D., 2016. Silicon Earth: Introduction to Microelectronics and Nanotechnology, Second Edition, 35

Çulha D., *AN AGILE BUSINESS PROCESS SOFTWARE DEVELOPMENT METHODOLOGY*, Yayınlanmamış Doktora Tezi, METU, 2014

Doran G. T., *"There's a S.M.A.R.T. way to write management's goals and objectives".* Management Review, 1981

Derindere M., *PROJECT AND PROCESS MANAGEMENT METHODS INFLUENCING AGILE SOFTWARE DEVELOPMENT*, Yayınlanmamış Yüksek Lisans Tezi, Fatih Üniversitesi, 2013

Larson E. W., Gray C. F., *Project Management - The Managerial Process Fifth Edition*, 2011, 5-7

Fair J., *Agile versus Waterfall: approach is right for my ERP project*, 2012, 3

Goldratt E. M., *Project Management - The Managerial Process Fifth Edition*, 1997, 120

Günsel M., *Toksik ve Yıkıcı Liderliğin Çalışan Performansı Üzerindeki Etkileri*, Yayınlanmamış Yüksek Lisans Tezi, Beykent Üniversitesi SBE, 2017

Larson E. W., Gray C. F., *Project Management - The Managerial Process Fifth Edition*, 2011, 5-7

Highsmith J., *Agile Project Management: Creating Innovative Products,* 2009, 60

Hirsch M., *Moving from a plan driven culture to agile development (ICSE),* 2005, 38

Jacobson I., *Object-Oriented Software Engineering,* 1992, 65

Kanigel R., *The One Best Way: Frederick Winslow Taylor and the Enigma of Efficiency*, 1997, 85-86

Kerzner H., *Project Management. A systems approach to planning, scheduling, and controlling, 10th Edition*, John Wiley and Sons Inc, Hoboken, New Jersey, 2009, 35

Kır G., *Scrum Adoption in KuveytTürk IT,* Yayınlanmamış Yüksek Lisans Tezi, Doğuş Üniversitesi FBE, 2014

Kıraç M., *PROJECT MANAGEMENT METHODOLOGIES FOR NEW PRODUCT DEVELOPMENT,* Yayınlanmamış Yüksek Lisans Tezi, Özyeğin Üniversitesi, 2018

Kwak Y. H., *Brief History Of Project Management. In K. A. Carayannis, The Story of Managing Projects*, 2003, 1-9

Larman C., Basili V. R., *Iterative and Incremental Development: A Brief History,* 2003, 54

Larson E. W., Gray C. F., *Project Management: The Managerial Process*, 2008

Leffingwell D., *Agile Software Requirements. Boston: Pearson Education Inc*., 2011, 45

Lock D., *Project Management (9th ed.),* 2007, 1-5

Loomis M. E. S., Shah A.V., Rumbaugh J. E., *An Object Modeling Technique for Conceptual Design,* 20002, 85

Manning, S., *"Embedding projects in multiple contexts – a structuration perspective". International Journal of Project Management*, 2008

McConnell S., *Code Complete 2nd Edition,* 2004, 412

Moe N. M., Dingsøyr T., *Agile Processes in Software Engineering and Extreme Programming. Scrum and Team Effectiveness: Theory and Practice Volume 9: Limerick*, Ireland: Springer, 2008, 11-20

Olivier M., *Project feasibility – Tools for uncovering points of vulnerability,* 2017*, 145*

Oxford, *Oxford Dictionaries, 2016, 21-991*

Palmer S. R., Felsing J. M., *A Practical Guide to Feature-Driven Development, 2002, 42*

Parnas D. L., *SDI: Two Views of Professional Responsibility,* 1987, 25

Permana P. A. G., *Scrum Method Implementation in a Software Development Project Management,* 2015, 199

Philips J., *PMP Project Management Professional Study Guide,* 2003

PMBOK,*"Proje Yönetimi Bilgi Birikimi Klavuzu" 5.Basım, Proje Yönetimi Enstitüsü, 2013, 6*

Poppendieck M., Poppendieck T., *Lean Software Development an Agile Toolkit,* 2003, 12-14

PMI, *A guide to the project management body of knowledge PMBOK® Guide, 6 th edition*, Newtown Square, Pennsylvania, 2017

Royce W.W., *Managing the development of large software systems: concepts and techniques. In ICSE'87 Proceedings of the 9th international conference on Software Engineering*. Los Alamitos, 1987. IEEE Computer Society Press., 60

Schach S. R., *Object-Oriented Classical Software Engineering 7th Edition, 2007, 53*

Schwaber K., Sutherland J., *The Scrum Guide*, 2017

Schwaber K., Beedle M., *Agile Software Development with Scrum*, 2004, 130

Seymour T. J., *The History of Project Management,* 2014, 233-238

Sharp H., Robinson H., *A Distributed Cognition Account of Mature XP Teams,* 2008, 4-9

Stevens M. J., *Project Management Pathways, 2002, 15*

Şahin M., *AGILE (SCRUM) IMPLEMENTATIONS FOR BANKING APPLICATIONS IN TURKEY, IS SCRUM USEFUL OR NOT?,* Yayınlanmamış Yüksek Lisans Tezi*, 2016*

Williams L. A., Kessler R. R., *All I Really Need to Know About Pair Programming I Learned in Kindergarten,* 2000, 1-12

Witzel M., *Fifty Key Figures in Management,* Routledge 11 New Fetter Lane, London, *2003, 83-85*

İNTERNET KAYNAKLARI

Agile Alliance, 2001, Extreme Programming, (Erişim Tarihi: 15/04/2019).
https://www.agilealliance.org/glossary/xp/

Agile Alliance, 2001, Agile 101, (Erişim Tarihi: 15/05/2019).
https://www.agilealliance.org/agile101/

Agile Modeling, Tarih yok, Feature Driven Development (FDD) and Agile Modeling, (Erişim Tarihi: 20/04/2019).
http://agilemodeling.com/essays/fdd.htm

Agile Business Consortium, 2014, The DSDM Agile Project Framework, (Erişim Tarihi: 25/04/2019).
https://www.agilebusiness.org/content/introduction-0

Benefield G., 2008, Rolling out Agile in a Large Enterprise, (Erişim Tarihi: 25/04/2019).
http://www.controlchaos.com/storage/scrum-articles/YahooAgileRollout.pdf

Brewer J., *Jera design*, 2001, (Erişim Tarihi: 20/04/2019).
https://www.jera.com/techinfo/xpfaq

Business Dictionary, 2018, "What is project? Definition & Meaning", (Erişim Tarihi: 16/05/2019).
https://www.BusinessDictionary.com

CMS Wire, 2019, When to Use Agile vs. Waterfall for Your Next Project, (Erişim Tarihi: 14/05/2019)
https://www.cmswire.com/information-management/when-to-use-agile-vs-waterfall-for-your-next-project/

Cohn M., "*Six Attributes of a Good Scrum Master*", 2007, (Erişim Tarihi: 08/04/2019).
http://www.scrumalliance.org/community/articles/2007/february/leader-of-the-band

Collin – Cope, M., 2002, "*Planning to be agile? A discussion of how to plan agile, iterative and incremental developments*" , (Erişim Tarihi: 20/01/2019).
http://www.ration.co.uk/whitepaper_12.pdf

Columbia University Computing History, Programming the ENIAC, 2019 (Erişim Tarihi: 01/05/2019). http://www.columbia.edu/cu/computinghistory/eniac.html

Fabe. 2010. «*MODERN PROJE YÖNETİMİ KİMİN İCADI?*» (Erişim Tarihi: 24/03/2019). http://fabe.biz/?p=1577

Fowler M., Highsmith J., *Agile Manifesto*, 2001, (Erişim Tarihi: 30/03/2019). https://agilemanifesto.org/

Forbes, 2019, Make Agile Your Own: Adapt It To Succeed In A Client-Centric Environment, (Erişim Tarihi: 10/05/2019) https://www.forbes.com/sites/forbestechcouncil/2019/04/30/make-agile-your-own-adapt-it-to-succeed-in-a-client-centric-environment/#6c931f2764fc

Historic Projects, PROMPT, 2019, (Erişim Tarihi: 15/07/2019)

https://www.historicprojects.com/PROMPT.html

Jensen G., Top Project Management Approaches Explained, 2017 (Erişim Tarihi: 01/05/2019).

https://guthriejensen.com/blog/top-project-management-approaches-visual-guide/

IBM, What is software development?, 2017, (Erişim Tarihi: 10/07/2019). https://www.ibm.com/topics/software-development

Krishnamurthy V., *"A Brief History of Scrum",* 2012, (Erişim Tarihi: 17/03/2019). http://www.techwell.com/2012/10/brief-history-scrum

Levison M., *Can Product Owner and Scrum Master be Combined?*, 2008 (Erişim Tarihi: 06/04/2019).

http://www.infoq.com/news/2008/12/scrum-master-product-owner

METU, Software Development Methodologies, 2009 (Erişim Tarihi: 20/08/2019). http://ocw.metu.edu.tr/pluginfile.php/609/mod_resource/content/0/2dersnotu2.pdf

Mind Tools, 2019, SMART Goals, (Erişim Tarihi: 10/04/2019). https://www.mindtools.com/pages/article/smart-goals.htm

Raygun, *Software Development Life Cycle Practices*, 2019 (Erişim Tarihi: 20/04/2019).

https://raygun.com/blog/software-development-life-cycle/#practices

Rezaid, 2019, *Software Development Life Cycle - Waterfall Model*. (Erişim Tarihi: 28/04/2019).

https://rezaid.co.uk/sdlc-waterfall-model/

Software Testing Material, 2019, Software Development Life Cycle – SDLC | Software Testing Material, (Erişim Tarihi: 08/04/2019). https://www.softwaretestingmaterial.com/sdlc-software-development-life-cycle/

Scrum Alliance, *About Scrum*, 2009, (Erişim Tarihi: 05/04/2019). http://www.scrumalliance.org

Scrum, 2017, "*What is Scrum?"*, (Erişim Tarihi: 20/04/2019).

https://www.scrum.org/resources/what-is-scrum

Standish Group, 2015, CHAOS Report, (Erişim Tarihi: 05/03/2019). https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

Türk Dil Kurumu. 2019. (Erişim Tarihi: 09/02/2019). http://www.tdk.gov.tr

TIKY Y. T. W., (2016), *Software Development Life Cycle*, 16 Temmuz 2019 tarihinde https://www.cse.ust.hk/~rossiter/independent_studies_projects/software_development/software_development_report.pdf adresinden alındı.

Voigt B. J. J., *Dynamic System Development Method*, 2004, 13 (Erişim Tarihi: 05/07/2019). https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws03/14_Voigt_DSMD_Ausarbeitung.pdf

EKLER

EK 1. Anket Formu Örneği

Sayın Katılımcı,

Bu anket formu, Beykent Üniversitesi tarafından yürütülmekte olan *"Çevik Proje Geliştirme Yöntemlerinin Firma Performansına Etkileri"* isimli araştırmanın deneysel kısmı ile ilgilidir.

Bu araştırma çalışması tamamen akademik bir amaca yönelik olup, çalışmanın amacı; bankaların Bilgi Teknolojileri alanında faaliyet gösteren kişilerin yazılım/proje geliştirme yöntemlerine bakış açısı ve bu yöntemlerin kullanımıyla ilgili veriler elde ederek bu hususta anlamlı bilimsel sonuçlara ulaşabilmektir.

Anketteki hiçbir sorunun doğru ya da yanlış cevabı yoktur. Elde edilen sonuçlar kişi bilgileri belirtmeksizin genel ve ortalama özellikler şeklinde, bu araştırmaya katılan kişilere de arzu etmeleri durumunda gönderilecektir. Birbirine benzeyen ve tekrar gibi görünen sorular araştırma tekniği açısından sorulması zorunludur. Dolayısıyla bütün soruların cevaplandırılması, değerlendirmenin sağlıklı yapılabilmesi için büyük önem arz etmektedir.

Anket iki bölümden oluşmaktadır. Birinci bölümde demografik sorular, ikinci bölümde ise araştırma soruları bulunmaktadır. Birinci bölüm tamamlandığında, İLERİ butonu ile ikinci bölüme devam edebilirsiniz.

Katılımınız, ayırdığınız değerli vaktiniz ve bilimsel çalışmaya verdiğiniz destek için teşekkürlerimizi sunar, çalışmalarınızda başarılar dileriz.

Dr. Canan Urhan

cananurhan@beykent.edu.tr

Araştırma Danışmanı

Mustafa Samet Özkan (Yüksek Lisans Öğrencisi)

mustafa.sametozkan@gmail.com

Araştırma Sorumlusu

Anket Soruları

1. Lütfen görev tanımınızı belirtiniz.
2. Cinsiyet?
3. Yaşınız?
4. Çalıştığınız kurumda ne zamandan beri görev yapıyorsunuz?
5. Ne zamandan beri yazılım geliştirme ve/veya test süreçleriyle ilgileniyorsunuz?
6. Çevik yöntemleri kullanmak suretiyle iş süreçlerinde görev aldınız mı?
7. Çevik yöntemler, yazılım gereksinimlerindeki hızlı değişikliklere cevap verme noktasında daha iyidir.
8. Çevik yöntemler ile projeler daha uzun zamanda tamamlanmaktadır.
9. Çevik yöntemler, bankacılık uygulamalarında COBIT ve ITIL standartlarıyla uyumludur.
10. Çevik yöntemler, proje süresinin azaltılmasına yardımcı olur.
11. Çevik yöntemler ile takımlar daha az motive olur ve daha az performans gösterir.
12. Genel olarak (avantajlar, eksiklikler) geleneksel yöntemler, Çevik yöntemlerden çok daha iyidir.
13. Diğer yazılım geliştirme metodolojileri, yazılım gereksinimlerindeki hızlı değişikliklere cevap verme noktasında Çevik yöntemlerden daha iyidir.
14. Çevik yöntemler şeffaflığı sağlar. Ne yapıldığını ve projedeki sorunların ne olduğunu görebiliyorum.
15. Çevik yöntemleri uygulayan ekipler, ekip dışından herhangi bir yardım almadan kendi başlarına bir şey öğrenen, kendi kendilerini motive eden ve kendi kendilerini organize eden ekiplerdir.
16. Yazılım geliştirme ekipleri Çevik yöntemler ile daha motive olmuştur ve daha fazla performans gösterirler.
17. Çevik yöntemler şeffaflığı sağlamaz. Ne yapıldığını, ne yapılacağını, projede sorunlarında ne olduğunu göremiyorum.
18. Yönetim, BT projelerinde Çevik yöntemlerin kullanılmasını destekliyor.
19. Çevik yöntemleri uygulayan ekipler, kendi kendine motive olamazlar ve takım dışından herhangi bir yardım almadan kendi başlarına bir şey öğrenemeyen kendi kendilerine organize olmayan ekiplerdir.
20. Çevik yöntemleri uygulamak suretiyle çalışmaya devam etmek her zaman yararlıdır.
21. Çevik yöntemlerin uygulandığı ekiplerde yardımlaşma, koordinasyon ve diğer ekiplerle işbirliğinde gözle görülür bir seviyede artış gözlenir.

22. Çevik yöntemlerin uygulandığı projelerde, projeyi daha nitelikli ve amaçlarına daha uygun hale getirir.
23. Çevik yöntemlerin uygulandığı ekiplerde yardımlaşma, koordinasyon ve diğer ekiplerle işbirliğinde azalış gözlenir.
24. Çevik yöntemlerin uygulandığı ekipler, zaman yönetimini daha başarılı uygulamaktadır.
25. Çevik yöntemler, projelerin kalitesini düşürür ve aynı zamanda gereksinimleri de karşılayamazlar.
26. Çevik yöntemlerin uygulandığı ekiplerde, ekip üyeleri metodolojiyi çok iyi tanıyor, her şeye tam destek veriyor ve her türlü gereksinimi yapıyorlar.
27. Yönetim, BT projelerinde Çevik yöntemlerin kullanılmasını desteklemez.
28. Çevik yöntemleri bankacılık uygulamalarında COBIT ve ITIL standartlarıyla uyumlu değildir.
29. Çevik yöntemlerin uygulandığı ekiplerde, ekip üyeleri metodolojiyi çok iyi tanımıyorlar ve her türlü şartı yerine getirmeye istekli değiller.
30. Genel olarak (avantajlar, eksiklikler) Çevik yöntemler, geleneksel yöntemlerden çok daha iyidir.
31. Sizce Çevik yöntemlerin avantajları nedir? Kısaca açıklayınız.
32. Sizce Çevik yöntemlerin dezavantajları nedir? Kısaca açıklayınız.

(0 – Atlandı, 1 – Kesinlikle katılmıyorum, 2 – Katılmıyorum, 3 – Kararsızım, 4 – Katılıyorum, 5 – Kesinlikle katılıyorum)

ÖZGEÇMİŞ

31 Ağustos 1991 tarihi, Çorum ili doğumluyum. İlköğretim ve lise eğitimini Çorum'da tamamladıktan sonra, Marmara Üniversitesi, Fen ve Edebiyat Fakültesi, Fizik bölümüne kaydoldum. Bu bölümden 2013 yılında mezun oldum. 2014 yılından itibaren özel bir bankanın yazılım geliştirme (IT) departmanında İş Analisti olarak görevimi sürdürmekteyim. 2016 yılı içerisinde Beykent Üniversitesi, İşletme Anabilim dalında, İşletme Yönetimi bölümünde yüksek lisans eğitimine başladım. 2019 yılında başladığım askerlik görevimi, MEBS Okulu ve Eğitim Merkezi Komutanlığı'nda tamamladım.

Özel ilgi alanlarım, yazılım ve süreç geliştirme, bankacılık, proje yönetimi ve iş ilişkileri yönetimidir. Yabancı dilim İngilizce olup, Japonca öğrenmeye çalışmaktayım.

Mustafa Samet ÖZKAN