T.C.

Radyo ve Televizyon Üst Kurulu

YAZILIM YAŞAM DÖNGÜSÜ, SÜREÇ MODELLERİ VE RTÜK'E UYARLANABİLİRLİĞİ

UZMANLIK TEZİ

Melike Gizem DAVUTOĞULLARI

ANKARA

HAZİRAN / 2021

T.C.

Radyo ve Televizyon Üst Kurulu

YAZILIM YAŞAM DÖNGÜSÜ, SÜREÇ MODELLERİ VE RTÜK'E UYARLANABİLİRLİĞİ

UZMANLIK TEZİ

Melike Gizem DAVUTOĞULLARI

DANIŞMAN

Üst Kurul Uzmanı Cem ULUOĞLAKCI

ANKARA

HAZİRAN / 2021

Radyo ve Televizyon Üst Kurulu Başkanlığına

Bu çalışma, Tez Değerlendirme Komisyonu tarafından oy birliği/oy çokluğu ile Uzmanlık Tezi olarak kabul edilmiştir.

	Adı Soyadı	İmza
Başkan :		
Üye :		
Üye :		
Üye :		
Üye :		

ONAY

…./…./2021

Ebubekir ŞAHİN

Üst Kurul Başkanı

TEZ TESLİM TUTANAĞI VE DOĞRULUK BEYANI

Radyo ve Televizyon Üst Kurulunda görev yapan Üst Kurul Uzman Yardımcısı olarak, Radyo ve Televizyon Üst Kurulu Tez Hazırlama Yönergesi'ne uygun şekilde hazırlamış olduğum uzmanlık tezi ilişikte sunulmuştur.

Bu uzmanlık tezindeki bütün bilgilerin akademik kurallara ve etik davranış ilkelerine uygun olarak toplayıp sunduğumu; ayrıca, bu kural ve ilkelerin gereği olarak, çalışmada bana ait olmayan tüm veri, düşünce ve sonuçları andığımı ve kaynağını gösterdiğimi beyan ederim.

Bilgilerinizi ve gereğini arz ederim.

…./…./2021

Melike Gizem DAVUTOĞULLARI

Üst Kurul Uzman Yardımcısı

Uzmanlık Tezinin Adı:

Yazılım Yaşam Döngüsü, Süreç Modelleri ve RTÜK'e Uyarlanabilirliği

EK:

1- TEZ (6 adet)

2- CD (1 adet)

Tezi Teslim Alan

Daire Başkanı

…./…./ 2021

TEŞEKKÜR

Bu tez çalışmasının hazırlığı sırasında bana destek veren başta Üst Kurul Başkanı Sayın Ebubekir ŞAHİN'e, Sayın Üst Kurul Üyelerine, Daire Başkanım Sayın Olcay ÇİÇEKLİTAŞ'a, Daire Başkan Yardımcım Sayın Murat İNCE'ye, Tez Danışmanım Sayın Cem ULUOĞLAKCI'ya ve Tez Komisyon Üyelerine teşekkürlerimi sunarım.

Bu uzun ve yoğun süreçte en büyük destekçilerim olan Sevgili Kardeşim Vildan, Annem Hacer ve Babam Nurullah'a bilhassa teşekkür ederim.

İÇİNDEKİLER

TEZ TESLİM TUTANAĞI VE DOĞRULUK BEYANI I

TEŞEKKÜR II

İÇİNDEKİLER III

ŞEKİLLER ve TABLOLAR VII

KISALTMALAR VIII

GİRİŞ 1

TEZİN KONUSU 3

TEZİN AMACI 3

TEZİN ÖNEMİ 3

TEZİN YÖNTEMİ 4

TEZİN KAPSAMI VE SINIRLILIKLARI 4

BİRİNCİ BÖLÜM 5

TEMEL KAVRAMLAR 5

1.1. YAZILIM NEDİR? 5

1.1.1. Sistem Yazılımları 6

1.1.2. Uygulama Yazılımları 7

1.2. METODOLOJİ (YÖNTEMBİLİM) VE SÜREÇ MODELLERİ 7

1.3. YAZILIM MÜHENDİSLİĞİ 8

1.4. YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ 10

1.4.1. Yazılım Geliştirme Yaşam Döngüsü Aşamaları 14

1.4.2. Yazılım Geliştirme Yaşam Döngüsüne Alternatif Bir Bakış Açısı .. 20

1.5. DOĞRUSAL, YİNELEMELİ, EVRİMSEL VE ARTIMLI GELİŞTİRME YAKLAŞIMLARI 21

1.6 ISO/IEC 12207 YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ SÜREÇLERİ STANDARDI 23

İKİNCİ BÖLÜM 26
YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ: GELENEKSEL YÖNTEMLER 26
2.1 ŞELALE MODELİ (WATERFALL MODEL) 27
2.1.1. Şelale Modelinin Avantajları 29
2.1.2. Şelale Modelinin Dezavantajları 30
2.2. V-ŞEKLİ MODELİ 31
2.2.1 V-Şekli Modeli Avantajları 33
2.2.2 V-Şekli Modeli Dezavantajları 34
2.3 HELEZONİK (SPİRAL) MODEL 34
2.3.1 Helezonik Model Avantajları 38
2.3.2 Helezonik Model Dezavantajları 39
2.4. PROTOTİP MODELİ 40
2.4.1. Yazılım Prototipleme Türleri 43
2.4.2. Prototip Modelinin Avantajları 44
2.4.3. Prototip Modelinin Dezavantajları 45
2.5. ARTIMLI MODEL (INCREMENTAL MODEL) 45
2.5.1. Artımlı Modelinin Avantajları 48
2.5.2. Artımlı Modelinin Dezavantajları 48
ÜÇÜNCÜ BÖLÜM 50
YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ: ÇEVİK (AGİLE) YAZILIM GELİŞTİRME YÖNTEMLERİ 50
3.1. SCRUM 53
3.1.1. Scrum İçerikleri 55
3.1.2. Scrum Avantajları 65
3.1.3. Scrum Dezavantajları 66

3.2. EKSTREM PROGRAMLAMA (EXTREME PROGRAMMING) 67
3.2.1. Ekstrem Programlama Yaşam Döngüsü 68
3.2.2. Ekstrem Programlama Değerleri 70
3.2.3. Ekstrem Programlama Prensipleri 71
3.2.4. Ekstrem Programlama Rolleri 73
3.2.5. Ekstrem Programlama Teknikleri 74
3.2.6. Ekstrem Programlama Avantajları 77
3.2.7. Ekstrem Programlama Dezavantajları 78
3.3. KANBAN 78
3.3.1. Kanban Değerleri 81
3.3.2. Kanban Prensipleri 82
3.3.3. Kanban Teknikleri 83
3.3.4. Kanban Avantajları 85
3.3.5. Kanban Dezavantajları 85
3.4. KRİSTAL GELİŞTİRME YÖNTEMLERİ 85
3.4.1. Kristal Yöntemlerin Avantajları 88
3.4.2. Kristal Yöntemlerin Dezavantajları 88
3.5. DİNAMİK SİSTEM GELİŞTİRME YÖNTEMİ (DYNAMIC SYSTEM DEVELOPMENT METHOD) 88
3.5.1. Dinamik Sistem Geliştirme Yöntemi Prensipleri 89
3.5.2. Dinamik Sistem Geliştirme Yöntemi Aşamaları 91
3.5.3. Dinamik Sistem Geliştirme Yöntemi Rolleri 93
3.5.4. Dinamik Sistem Geliştirme Yöntemi Temel Araçları ve Teknikleri 94
3.5.5. Dinamik Sistem Geliştirme Yöntemi Avantajları 95
3.5.6. Dinamik Sistem Geliştirme Yöntemi Dezavantajları 96
DÖRDÜNCÜ BÖLÜM 97

BÜTÜNLEŞİK YETENEK OLGUNLUK MODELİ 97
4.1. CMMI Yapısı 98
4.2. CMMI Model Gösterimleri 100
4.3. Türkiye'de CMMI Kullanımı 104
BEŞİNCİ BÖLÜM 106
YAZILIMLARIN BAŞARISIZ OLMA SEBEPLERİ 106
BULGULAR VE ANALİZ 110
TARTIŞMA VE YORUM 117
SONUÇ 124
KAYNAKÇA 126
ÖZET 135
ABSTRACT 137
ÖZGEÇMİŞ 138

ŞEKİLLER ve TABLOLAR

Şekil 1.1.1 Sistem Yazılımları ve Uygulama Yazılımları 6
Şekil 1.4.1 Yazılım Geliştirme Yaşam Döngüsü Aşamaları 13
Şekil 1.5.1 Doğrusal, Yinelemeli, Evrimsel Geliştirme Yaklaşımları 22
Şekil 1.5.2 Artımlı Geliştirme Yaklaşımı 23
Şekil 1.6.1 ISO/IEC 12207 Standardı 24
Şekil 2.1.1 Şelale Modeli 28
Şekil 2.2.1 V Şekli Modeli 32
Şekil 2.3.1 Helezonik Model 35
Şekil 2.5.1 Artımlı Model 46
Şekil 3.1.1 Scrum Yöntemi 53
Şekil 3.1.1.1 Scrum 56
Şekil 3.1.1.3.1 Ürün İş Listesi Örneği 63
Şekil 3.1.1.3.2 Koşu İş Listesi Örneği 63
Şekil 3.1.1.3.3 Koşu Kalan İş-Zaman Grafiği Örneği 64
Şekil 3.2.1.1 Ekstrem Programlama 68
Şekil 3.3.1 Toyota'dan orijinal bir kanban panosu Toyota 79
Şekil 3.3.2 Kanban Panosu Örneği 80
Şekil 3.4.1 Kristal Yöntemi 86
Şekil 4.1 CMMI Takımyıldızı 98
Şekil 4.1.1 CMMI Yapısı 99
Şekil 4.2.1 CMMI Gösterim Çeşitleri 101
Şekil 5.1 RTÜK için önerilen şema 123

Tablo 1 Yöntemlerin Karşılaştırılması 115

KISALTMALAR

a.g.e : Adı Geçen Eser

a.g.m : Adı Geçen Makale

BT : Bilgi Teknolojileri

CMMI : Bütünleşik Yetenek Olgunluk Modeli (Capability Maturity Model Integration)

DSDM : Dinamik Sistem Geliştirme Yöntemi (Dynamic System Development Method)

RTÜK : Radyo ve Televizyon Üst Kurulu

s : Sayfa

SDLC : Software Development Life Cycle (Yazılım Geliştirme Yaşam Döngüsü)

TDK : Türk Dil Kurumu

UML : Birleşik Modelleme Dili (Unified Modelling Language)

WIP : Work In Progress (Devam Eden İş)

XP : Ekstrem Programlama (Extreme Programming)

GİRİŞ

Dünya üzerinde en az 26 milyon adet yazılım geliştirici bulunmaktadır.[1] Her yazılımcının geliştirdiği yazılımlar, ürettiği ürünler vardır ve bu yazılım ürünlerinin neredeyse tamamı kullanıcıların kullanımına sunulmaktadır. Modern dünyayı yazılım olmadan hayal etmek imkânsızdır. Ulusal çapta geliştirilen yazılım projeleri, irili ufaklı yazılım projelerinin neredeyse tamamı bilgisayar tabanlı sistemler tarafından kontrol edilir, bilgisayar ve onu kontrol eden bir yazılım içerir. Yazılım sistemleri finans sistemleri, eğitim, sağlık, üretimde olduğu gibi müzik, bilgisayar oyunları, televizyon ve film gibi eğlence sektörleri dâhil neredeyse her alanda karşımıza çıkmaktadır. Ülkemizde bütün kamu kuruluşlarında bilgisayar tabanlı sistemler bulunmakta, gerek veri depolama gerek veri sağlama amacıyla olsun bu bilgisayar sistemlerinden yararlanılmaktadır. Başka ülkelerle olan iletişimimizin büyük çoğunluğu elektronik ürünlerle sağlanmaktadır. Hatta son zamanlarda bütün dünyayı etkileyen küresel bir salgının da patlak vermesiyle, çalışma ortamı olan ofislerin terk edilerek, fiziksel veya politik sınırlarla bağlı kalınmaksızın çalışılabilmesi olgusu, iletişimde yazılım tabanlı ürünlerin ağırlığının ne kadar arttığını açıkça gözler önüne sermektedir. Yani genel olarak 'yazılım', hem ulusal hem de uluslararası alan için vazgeçilmez ve hayati bir konumdadır.

Yazılım sistemleri yapısal olarak soyut ve elle tutulamaz ürünlerdir. Materyal, fiziksel kanunlar ya da ürün geliştirme gibi kıstaslarla bir kalıba sokulamazlar. Bu da yazılımın gelişme potansiyelinde doğal bir sınır olmadığı hakikatini doğurur. Fakat bu durum yani fiziksel sınırların olmaması yazılım sistemlerinin çok çabuk bir şekilde anlaşılması zor ve değiştirmesi güç bir hale gelebilmesine sebep olur. Bu durum kaçınılmak istenen bir durumdur. Dolayısıyla teknoloji ilerledikçe bu durumun önüne geçmek istenmiş, bir açıklık getirilmeye çalışılmıştır ve bu yönde ortaya çeşitli fikirler atılmıştır.

[1] https://evansdata.com/press/viewRelease.php?pressID=278, (23.01.2021)

Bir yazılım ürünü somut bir olgu değildir. Çevresel faktörlere açık bir şekilde var olur. Geliştirme süreci boyunca kullanıcı, geliştirme süresi, bütçe gibi çeşitli etkenler işin içine girer. Bu etkenlerin bazıları önceden tespit edilip düzenlenebilir fakat bütün sürece tamamen hâkim olmak mümkün değildir. Bazen belirsiz etkenler de sürece dâhil olarak gidişata müdahale eder. Yazılım bu gibi sebeplerle çok çabuk karmaşıklaşabilmektedir. Bu durum, yazılımla ilgilenen insanları çeşitli çözümler getirmek için çalışmaya güdülemiştir.

Yazılım ürünlerinin nasıl yapılması gerektiğini tek bir yolla ya da yöntemle anlatmak mümkün değildir zira yazılım tek bir çeşit olarak ele alınamaz. Yazılım sistemlerinin basit bir cihaz içindeki gömülü sistemden tutun dünya çapında kullanılan bilgi sistemleri gibi karmaşık büyük projelere kadar değişik türleri vardır. Her projenin kendi özgünlüğü, karşıladığı ihtiyaçları, sorunları ele alışı ve çözüm getirişi, teknikleri ve yöntemleri olduğu için bütün yazılımları aynı kurallar çerçevesinde ele almaya çalışmak güçtür. Örneğin ulusal çapta kullanılan E-Devlet gibi bir sistemi geliştirmek ile bir kamu kuruluşunda kullanılacak o kuruma özel bir kütüphane projesinin geliştirilmesi çok farklıdır. İki proje için de titiz bir çalışma gerekir fakat projelerin geliştirme süreleri, maliyetleri, geliştirilen yazılım ürünün özelikleri birbirinden oldukça ayrıdır. Bu çalışmaların hepsi yazılım mühendisliğine ihtiyaç duyar fakat hepsi aynı yazılım mühendisliği tekniklerine tabi olmak zorunda değildir. Bir yazılım geliştirmek kişilerin, şirketlerin, kurumların ve kullanıcıların yapısına göre farklı şekillerde ve onların değerli zamanlarını alarak gerçekleşir. İster küçük çaplı ve kısa sürede geliştirilebilen yazılım projeleri olsun ister büyük ve karmaşık, uzun zaman alan yazılım projeleri olsun bütün yazılım projeleri geliştirilmeleri sırasında belli adımlardan geçer. Bu adımlar sonucunda ortaya bir yazılım ürünü çıkar. Geliştirilen yazılım ürünlerinin istenilen talepleri nasıl karşıladığı, ne oranda karşıladığı, yazılımın kalitesi gibi daha birçok kıstas yazılım ürününün geliştirilmesi sürecinde başından geçenlerle direkt olarak bağlıdır. Bu yüzden yazılım süreci oldukça önemlidir. Bu durumun farkına varan yazılım geliştiricilerden bazıları karşılaştıkları yazılım ürünlerinin yapısına göre bu süreci şekillendirmeye çalışmıştır. Dolayısıyla zaman içerisinde çeşitli yazılım türlerine olan talepleri karşılayabilmek için çeşitli yazılım geliştirme yöntemleri ortaya atılmıştır. Bu yazılım geliştirme yöntemlerinin her biri

hem birbirine dair ortak özellikler hem de kendilerine özgü özellikler barındırmaktadır. Geliştirilecek olan yazılımı en iyi şekilde geliştirebilmek için tercih edilen yöntemler değişmektedir ve yazılıma uygun bir yöntem seçilerek yazılım sürecini yönetmek oldukça önemlidir.

TEZİN KONUSU

Bu tez çalışmasında, kamu kurumlarında yer alan yazılımların ortaya çıkmasında önemli bir yer alan Yazılım Geliştirme Yaşam Döngüsü ve Süreç Modelleri incelenerek geçmişten günümüze dek gelen yazılım geliştirme modelleri ve yöntemleri karşılaştırmalı analiz yöntemi ile ele alınacaktır. Bütünleşik Yetenek Olgunluk Modeli açıklanacaktır. Yapılan analiz araştırması çalışmasında Radyo ve Televizyon Üst Kurulunda yazılım geliştirilirken takip edilmesi gereken yol haritası değerlendirilecektir.

TEZİN AMACI

Bu tez çalışmasında amaç, Radyo ve Televizyon Üst Kurulunda bir yazılım ürünü geliştirilirken istenen şekilde kurallara uygun, kaliteli, talep edilen ihtiyaçları karşılayabilen bir yazılım ürününün ortaya çıkmasını sağlayabilecek bilgilerin elde edilmesi, yazılım geliştirme sürecinde izlenebilecek yöntemleri inceleyip sunarken takip edilmesi gereken en ideal olduğu değerlendirilen yolun önerilmesidir.

TEZİN ÖNEMİ

Günümüzde neredeyse her alanda bir yazılım ürünü kullanılmaktadır. Bir yazılım geliştirilirken kullanıcının istediği ürünü ortaya çıkarmak oldukça zorlayıcı olabilmektedir. Bu çalışma yazılım geliştirilirken talep edilen ürünü geliştirmek için gereken en önemli unsurlardan olan yazılım geliştirme yaşam döngüsü yöntemlerinin ne denli elzem olduğunun farkındalığının arttırılmasına katkıda bulunacaktır. Radyo ve Televizyon Üst Kurulunda bir yazılım ürününün yazılım geliştirme yöntemleri

çerçevesinde belli bir plan program içerisinde geliştirildiği takdirde hedeflenen sonucu vermeye daha çok yaklaşacağı, ortaya daha kaliteli yazılım ürünlerinin çıkacağı, olası maliyet ve zaman israfının önüne geçileceği kapsamında katkılar sunacağı düşünülmektedir.

TEZİN YÖNTEMİ

Bu tez çalışmasında yöntem olarak "Karşılaştırmalı Analiz Yöntemi" kullanılmıştır. Tezin başlıklarında yer alan konular detaylıca incelenmiş, yöntem ve modellerin sağladığı avantajlar ve dezavantajlar tezin amacı doğrultusunda bilimsel etik ve kurallara uygun biçimde aktarılmıştır.

TEZİN KAPSAMI VE SINIRLILIKLARI

Bu çalışmada, günümüzde oldukça yaygın olan yazılım geliştirme yaşam döngüsü ve süreç modelleri Radyo ve Televizyon Üst Kuruluna olan yararlılıkları bağlamında incelenmiştir. Bu bağlamda literatür taraması yapılarak projenin tipini, büyüklüğünü ve doğasını da ele alarak iyi ve kaliteli bir yazılım geliştirmek için dünya genelinde yaygın olarak kullanılan önde gelen yazılım geliştirme yaşam döngüsü yöntemleri ve modelleri ile Bütünleşik Yetenek Olgunluk Modeli teze konu edilmiştir. Yazılım geliştirme yöntemleri araştırılmış, dezavantajları ve avantajları belirlenmiştir.

BİRİNCİ BÖLÜM

TEMEL KAVRAMLAR

1.1.YAZILIM NEDİR?

Yazılım sözcüğü Türk Dil Kurumu Büyük Türkçe Sözlüğünde: "Bir bilgi işlem dizgesinin işleyişi ile ilgili bilgisayar izlencelerinin, yordamların, kuralların ve gerektiğinde belgelemenin tümü." olarak tanımlanmıştır.[2] Yazılım elektronik parçaların birbiriyle iletişime geçerek bir bilgisayar ürününe ne yapması gerektiğini söyleyen, bilgisayar sisteminin çalışmasıyla ilgili çeşitli programları, süreçleri, kütüphaneleri içeren bir yapıtaşı olarak ele alınabilir. Bilgisayar yazılımı ve donanımı ayrılmaz bir bütündür, zira çalışmak için birbirine ihtiyaç duyarlar. Geçmişten bugüne çeşitli şekillerde karşımıza çıkan yazılım, bütün ürünlerde hayatın her yerinde yer almaktadır. İster sistem yazılımı ister uygulama yazılımı olarak geliştirilsin; işlevi, boyutu veya amacı ne olursa olsun, bir ürün için yazılım projeleri ortaya çıkarılıp yazılım yapılmaya başlandığında çeşitli adımlardan oluşan bir süreç işlemektedir. Süreç kelimesinin TDK'deki açıklaması ise: "Aralarında birlik olan veya belli bir düzen veya zaman içinde tekrarlanan, ilerleyen, gelişen olay ve hareketler dizisi, vetire, proses."[3] olarak geçmektedir.

İyi bir yazılım geliştirilirken yazılımcıların aklından çıkarmaması gereken noktalar şunlardır:

- Kalite odaklı olmak
- Yazılım süreci
- Projede kullanılan yöntemler
- Proje geliştirilirken kullanılan araç-gereçler
- Projenin gereksinimlerinin en üst seviyede karşılanması

[2]Büyük Türkçe Sözlük, http://www.tdk.gov.tr, 24.12.2020
[3]a.g.e, 24.12.2020

Yazılımlar ikiye ayrılır:

1. Sistem Yazılımları
2. Uygulama Yazılımları



Şekil 1.4.1.1 Sistem Yazılımları ve Uygulama Yazılımları

1.1.1. Sistem Yazılımları

Bir bilgisayarın işletim sistemini oluşturan programlar ve dosyalara genel olarak sistem yazılımı denilmektedir. Bilgisayarın donanımı ve bilgisayardaki uygulama yazılımlarıyla iletişim kuran, bir bilgisayar sisteminin bu bileşenleri arasındaki görevleri koordine eden bir ara yüz olarak değerlendirilebilir. Microsoft Windows, Android, MacOS gibi işletim sistemleri, hizmet uygulamaları ve aygıt sürücüleri sistem yazılımlarına örnek verilebilir.[4]

[4]Murat Erten, *Bilgisayar Mühendisliği'ne Giriş*, Papatya Yayıncılık, İstanbul, 2016, s. 15-32.

1.1.2. Uygulama Yazılımları

Uygulama yazılımı; son kullanıcılar için geliştirilen, kodlandıkları üzere işletim sistemindeki belirli görevleri gerçekleştirmek için tasarlanan yazılımlara verilen genel addır. Belge kullanımı, hesaplama işlemleri, resim ve video düzenleme, veri tabanı yazılımları, internet tarayıcıları uygulama yazılımına örnek verilebilir.[5]

1.2.METODOLOJİ (YÖNTEMBİLİM) VE SÜREÇ MODELLERİ

Metodoloji veya Yöntembilim, kelime olarak yöntemin bilimi anlamına gelmektedir. Çeşitli amaçlara ulaşmak için ne gibi yöntemler izlenilmesi gerektiği konusunda tavsiye ya da önerilerde bulunarak yol gösterdiği söylenilebilir. Terim olarak ise "araştırmacıların veya diğer kullanıcıların yaptığı seçimlere rehberlik eden, görüşlere, inançlara ve değerlere dayanan tutarlı ve mantıksal bir plan içinde yapılan araştırma için bağlamsal bir çerçeve, belirli disiplinler özelinde, gerçekliğin anlaşılması ve açıklanmasına yönelik bilgilerin üretilmesinde kullanılan yol, yöntem ve yordamları inceleyen bilim dalı." şeklinde tanımlanmaktır.[6]

Yazılım geliştirme sürecinde genel olarak yöntembilimlerin amacı; projelerin düşük maliyetle, belli bir zaman çerçevesinde başarıyla yapılmasını sağlamaktır. Bir yazılım projesinin başarılı olması uygun bir yöntembiliminin kullanılmasıyla mümkündür. Bilişim sistemlerine ait bir projeyi yürütmek için yazılım geliştirme yaşam döngüsünü tanımlayan ve zaman içinde uluslararası standart haline gelmiş birçok yöntembilim ve bu yöntembilimlerin altında tanımlı olan çeşitli model, mimari, yöntem, teknik ve araç anlamında uygulamalar vardır.[7]

[5]a.g.e. s.15-32

[6]Helen Kara, *Creative research methods in the social sciences: A practical guide*, Policy Press, Bristol, 2015, s. 4.

[7]Mustafa Erhan Sarıdoğan, *Yazılım Mühendisliği*, Papatya Yayıncılık, İstanbul, 2014, s. 397.

Yöntem kelimesinin TDK'deki tanımı şu şekildedir; "Bilimde belli bir sonuca erişmek için bir plana göre izlenen yol, metot."[8], Model ise şu şekilde tanımlanır; "Örnek olmaya değer kimse veya şey." Süreç Modelleri olarak adlandırılan, bir yazılım geliştirilirken izlenebilecek çeşitli yöntemler ve modeller vardır. Yöntem ve model kavramı oldukça benzer dursa da aslında farklıdır. Bu farklılığı Boehm şu şekilde ifade etmiştir; yöntem veri belirleme, ihtiyaçların belirlenmesi, kontrol gibi adımları yönetir; model ise bir yazılım projesinin tamamlanması için yapılması gereken görevleri içeren belli aşamaların takip edilmesi hususunda rehberlik eder.[9] Yazılım geliştiriciler projenin özellikleri, zaman, bütçe gibi kısıtlara uygun olarak birini seçerler.

1.3.YAZILIM MÜHENDİSLİĞİ

Yazılım Mühendisliği bilgisayar biliminin genç alanlarından biridir. 1968-1969 yıllarında Almanya'nın Garmisch kentinde gerçekleştirilen bir konferansta kabul görmüştür. Yazılım Mühendisliği'ne dair çeşitli tanımlamalar yapılmaktadır. 1969 yılındaki konferansta Fritz-Bauer tarafından şu şekilde tanımlanmıştır: "Gerçek makineler üzerinde etkin ve güvenilir çalışan ekonomik yazılımlar geliştirilmesi amacıyla mühendislik ilkelerinin kullanılmasıdır". Bir başka tanımı ise şu şekildedir, "Yazılımın geliştirilmesi, işlenmesi ve bakımına dair sayılabilir, sistematik ve disiplinli bir uygulamasıdır."[10] Yani aslında Yazılım Mühendisliği, bir ekip veya mühendis ekipleri tarafından inşa edilecek kadar büyük veya çok karmaşık olan yazılım sistemlerinin oluşturulmasıyla ilgilenen bilgisayar bilimi alanıdır.

Yazılım mühendisliği bir yazılım ürününe dair var olan bütün her şeyi ele alan mühendislik dalıdır. Yazılım mühendisliği için yazılım tanımlaması, yazılım

[8]Büyük Türkçe Sözlük, 26.12.2020

[9]Barry Boehm, "A Spiral Model Of Software Development And Enhancement", *Computer*, Sayı: 21, No. 5, 1998, s. 61.

[10]IEEE, "IEEE Recommended Practice for Architectural Description for Software-Intensive Systems," *IEEE Std 1471-2000*, 2000, s. 1-30

geliştirilmesi, yazılım evrimi ve yazılım güvenilirliği önemlidir. Yazılım mühendisliği kullanılabilir bir yazılım geliştirme ve sunma hususlarıyla ilgilenir.

Yazılım mühendisliğinde karşılaşılan bazı zorlukların şunlar olduğu değerlendirilebilir;

- Projenin başından sonuna dek gelişen değişimlere uyum sağlama
- Azalan teslim sürelerine uyum sağlama
- Güvenilir bir yazılım geliştirme
- Kaliteli bir yazılım ürünü ortaya çıkarma
- Belirlenen maliyet ve zaman aralığına uyma

Yazılım mühendisliğinde, kullanılan yöntem ne olursa olsun, dört temel faaliyetten meydana gelen sistematik bir yaklaşım uygulanır.[11] Bu faaliyetler şunlardır;

- Müşteriler ve geliştiriciler tarafından, üretilecek yazılımın gereksinim ve kısıtlarının belirlendiği Sistem Tanımlaması.
- Yazılımın tasarlandığı ve kodlamanın yapıldığı Yazılım Geliştirme.
- Yazılımın, müşterinin talep ettiği ürün olduğunun onaylandığı Yazılım Doğrulama.
- Yazılımın, değişen müşteri ve pazar ihtiyaçları doğrultusunda güncellendiği Yazılım Evrimi

Yazılım mühendisliğinin amacı yüksek kaliteli, kullanıcı dostu, iyi belgelenmiş, bakım-onarıma müsait modeller ve yöntemler ortaya çıkarmaktır. Bir yazılım ürünü düşünüldüğünde başlayan ve ürünün kullanımı bittiğinde sona eren zaman aralığı bu süreçler içerisinde değerlendirilebilir. Yazılım Geliştirme Yaşam Döngüsü olarak adlandırılan bu süreçler genel olarak gereksinim aşaması, tasarım aşaması, uygulama

[11]Ian Sommerville, *Software Engineering 9th edition*, Pearson Addison-Wesley, Boston, 2011, s. 9.

aşaması, test aşaması, kurulum ve deneme aşaması, bakım-onarım aşaması ve bazen de iptal aşamasından oluşur.

Yazılım, bilgiyi analiz etmek veya bir problemi çözmek için kendi içinde bir ürün olarak kullanılan bir araçtan geliştirilmiştir. Erken programlama aşamaları bir sürü problem oluşturarak yazılımı, özellikle bilgisayar merkezli yazılımların geliştirilmesini zorlaştırmıştır. Yazılım, yazılım mühendisliği prosedürlerinin bir parçası olarak oluşturulmuş belge ve programlardan oluşur. Dahası yazılım mühendisliğinin amacı, yüksek kalitede programlar oluşturan uygun bir çalışma yaratmaktır. Seçilen yaşam döngüsü modeline bağlı olmaksızın farklı yaşam döngüsü modelleri farklı yazılımlar için temel geliştirme faaliyetlerini farklı bir şekilde meydana getirebilir.[12] Yazılım geliştirilirken yapılması gereken temel faaliyetler, farklı yaşam döngüsü modellerinde farklı sırayla gerçekleştirilebilecek şekilde tüm yaşam döngüsü modellerine dâhil edilmiştir.

1.4.YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

Yazılım endüstrisinde yüksek kalitede iyi bir ürün ortaya çıkarmak, geliştirmek ve test etmek için geliştirilen yazılımın sistematik bir süreçten geçmesi gerekir. Yazılım ürünlerinin oluşturulduğu veya değiştirildiği ve yazılım ürünlerini geliştirmek için kullanılan çeşitli modeller ve yöntemleri içeren sürece 'Yazılım Geliştirme Yaşam Döngüsü' adı verilir.[13] Yazılım fikri ortaya atıldığı andan itibaren zamanla büyüyeceği ve yazılım geliştirilirken ortaya çeşitli kriterler ve gereksinimler dahil olacağı için bu süreci bir döngü olarak ele almak gerekir. Bu döngü doğrusal olarak ilerlemek durumunda değildir zira yazılım geliştirilirken çeşitli sebeplerden ötürü geriye dönmek gerekebilir. Bu sebeple kullanıcılar için kullanıcıların beklediği kalitede adım adım, aşama aşama yazılım geliştirme süreci olan Yazılım Geliştirme Yaşam Döngüsüne

[12]Project Management Institude (1996), *A Guide to the Project Management Body of Knowledge*, http://www.cs.bilkent.edu.tr/~cagatay/cs413/PMBOK.pdf, (10.02.2021)

[13]Sheema Suresh Kute ve Surabhi Deependra Thorat, "A Review on Various Software Development Life Cycle (SDLC) Models", *International Journal of Research in Computer and Communication Technology*, Sayı: 3, No: 7, 2014, s. 778-779.

daha geniş bir çerçeveden bakılmalıdır. Bu çerçeve ISO/IEC 12207 Yazılım Geliştirme Yaşam Döngüsü Süreçleri Standardı ile çizilmektedir. Bu standart herhangi bir yazılım ürününün tedarikini, test edilmesini, geliştirilmesini, işletme ve bakım süresince uygulanması beklenen aşamaları ve aynı zamanda faaliyetlerini tanımlamaktadır. Bu standardın denetim faaliyetleri Türkiye'de Türk Standartları Enstitüsü tarafından yürütülmektedir.[14]

Yazılım Geliştirme Yaşam Döngüsü bir yazılım ürününü oluşturmak için planlanan ve gerçekleştirilen yapı bütünü olarak ele alınır ve bu yapıda daha önce bahsedildiği gibi çeşitli yazılım geliştirme modelleri ve yöntemleri kullanılmaktadır. Bu model ve yöntemlerde ise her biri farklı bir işlev gören çeşitli basamaklar bulunmaktadır. Yazılım geliştirme sürecinde seçilen modele veya yönteme göre bu basamaklar artmakta ya da azalmaktadır fakat hepsinde ortak olan birtakım basamaklar bulunmaktadır.

Yazılım geliştirme yaşam döngüsü modelleri ve yöntemleri projeyi uygun hedef ve sınırlar doğrultusunda en iyi sonucu verecek şekilde ortaya çıkarmak için seçilmiş çeşitli modeller veya yöntemlerdir. Genel olarak geliştirme sürecinin yanı sıra yazılım kalitesini artırmaya da yardımcı olurlar. Her biri çeşitli hedefler için geliştirilmiş birçok model ve yöntem bulunmaktadır. Yazılım Geliştirme Yaşam Döngüsü yazılım geliştirilmesi sırasında her adımda uygulanan aktiviteleri belirten bir süreçtir ve temelleri şunlardır; [15]

- Problemi anlama
- Çözüm için bir plan oluşturma
- Oluşturulan planı kodlama

[14]Türk Standartları Enstitüsü, https://www.tse.org.tr/IcerikDetay?ID=945&ParentID=1214 (25.02.2021)

[15]Rajendra Ganpatrao Sabale ve Dani, Dr. A. R.,"Comparative Study Of Prototype Model For Software Engineering With System Development Life Cycle", *IOSR Journal of Engineering*, Sayı: 2, No: 7, 2012, s. 21-24.

- Esas programı test etme
- Ürünün sürdürülmesini sağlama

Yazılım geliştirme süreci ne tür bir yazılımın geliştirildiğine göre oluşturulan yapıdır. Bir Yazılım Geliştirme Yaşam Döngüsü modeli seçmek bütün geliştirme yapısını ve daha sonra projenin kalitesini belirleyen önemli bir adımdır.[16] İyi bir yazılım kullanıcı için gereken işlevselliği ve performansı karşılamalı, üzerinde iyileştirme yapılabilir, güvenilir ve kullanılabilir olmalıdır.

Yazılım kalitesini, işlevselliğini arttırmak amacıyla yıllar içerisinde çeşitli modeller ve yöntemler geliştirilmiştir. Geçmişte birçok yazılım projesi planlama yanlışlığı, planlama eksikliği, proje yönetiminde çıkan hatalar, araç-gereç eksikliği gibi çeşitli sebeplerden ötürü tamamlanma konusunda sıkıntı yaşamış ve başarısız olmuştur. Yazılım şirketleri daha önce yaşanılan başarısızlıkları göz önüne alarak incelemiş, bu hususta ders çıkarmış ve ilerleyen teknolojinin de sayesinde yeni yaklaşımlar ve yöntemler üzerine yoğunlaşmışlardır. Günümüzde hala devam eden bu çalışmalar sonucunda çeşitli yöntemler geliştirilmiştir zira iyi bir proje yönetimi her zaman yazılımın başarısına yol açan önemli bir etmendir.[17]

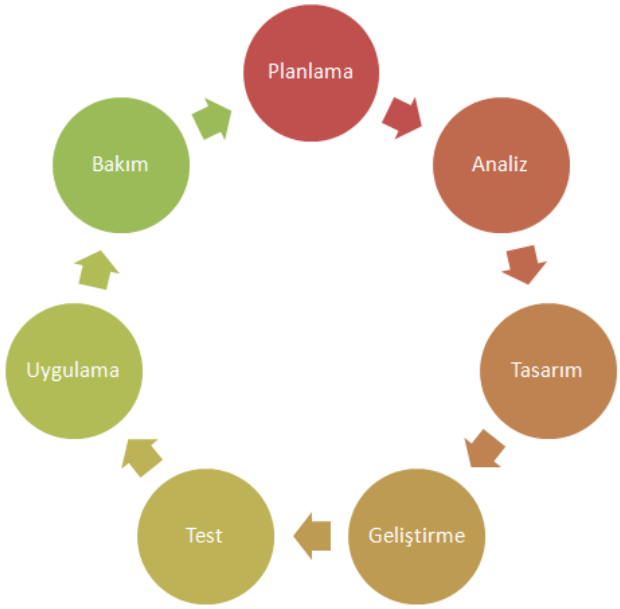
Geliştirilen her yazılım geliştirme yönteminin amacı bir önceki yöntemin eksiklerini kapatarak yazılım geliştirilirken daha iyi bir proje yönetimi sağlama, kaliteyi bir üst seviyeye taşımaktır. Temel olarak bütün yöntemler aynı adımlara sahiptir. Bazı yöntemlerde bir önceki yöntemlerde olmayan özellikleri eklemek adına ekstra adımlar yer alabilmektedir.

[16]Anastasiya Ivanovskaya, Aksyonova, Olga ve Ziomkovskaya, Polina, "Software Life Cycle Model Development at BPsim.DSS," *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 2020, s. 483-487.

[17]Kathy Schwalbe, *Information technology project management*, Cengage Learning, Boston, 2015, s. 8.

Yazılım Geliştirme Yaşam Döngüsü oldukça kolay, sık rağbet gören ve kullanışlı bir yöntemdir. Bir yazılım sistemini sıfırdan geliştirmenin karmaşıklığını minimum seviyeye indirme amacı güder. Bir proje fikrini işleyen bir yapı haline dönüştürmeye yardımcı olur ve bunu en kolay şekilde yapabilmek adına yazılım geliştirme sürecini aşamalar halinde bir döngü olarak ele alır. Bu döngü, herhangi bir yazılım ürününü geliştiren yazılım mühendislerinin takip etmesi gereken bir dizi aşamalardan oluşur. Döngünün aşamalarının sayısı izlenen yola göre 4 aşama ile 10 aşama arasında değişse de bir döngü ortalama 7 aşamadan oluşur.[18] Bu aşamalar Şekil 1.4.1'de de görülebileceği üzere şu şekildedir;

- Planlama
- Analiz
- Tasarım
- Geliştirme
- Test
- Uygulama
- Bakım



Şekil 1.4.1.1 Yazılım Geliştirme Yaşam Döngüsü Aşamaları

[18]Vanshika Rastogi, "Software Development Life Cycle Models-Comparison, Consequences", *International Journal of Computer Science and Information Technologies (IJCSIT)*, Sayı: 6, 2014, s.168-171

1.4.1. Yazılım Geliştirme Yaşam Döngüsü Aşamaları

1.4.1.1.Planlama

Yazılım mühendisleri tarafından bir projeye başlarken gerçekleştirilen ilk aşama planlamadır. Bu aşamada esas olarak, amaçlanan yazılımın ne yapacağı, hangi probleme çözüm getireceği üzerinde durulur; yazılımın hedeflenen işlevleri gerçekleştirirken nasıl bir ortamda çalışacağını ve kısıtlamalarını ele alır; gerçekleştirilmek istenen projenin planlama çalışmasına odaklanır. Başarılı bir ürün ortaya çıkarmak adına düzgün ve mantıklı bir planlama yapılması elzemdir zira eksik planlama ile analiz genellikle bir projenin başarısızlığa uğramasının temel nedenleri arasında gösterilmektedir.[19]

Planlama aşamasının amacı, projenin başarısını mümkün kılmak için projenin başından sona erdirilmesine kadar gereken tüm proje adımlarını ve faaliyetlerini planlamak ve kapsamlı bir plan serisi oluşturmaktır.

Planlama aşamasında şu kavramlar göz önüne alınmalıdır;[20]

- Kaynak belirleme
- Projenin kapsamı, süresi, risk ve maliyet tanımlaması
- Prosedürlerin yürütülmesi, incelenmesi, kontrol edilmesi ve sona erdirilmesinin tanımı
- Projenin nasıl ilerleyeceğine dair zaman çizelgesinin belirlenmesi
- Proje ile ilgili tüm alt planları koordine etmek için faaliyetlerin değerlendirilmesi ve açıklanması

[19]June Verner, Sampson, Jennifer ve Cerpa, Narciso,"What Factors Lead To Software Project Failure?"*Proceedings of the 2nd International Conference on Research Challenges in Information Science RCIS,*2008*,* s. 71 - 80.

[20]Ian Sommerville, *Software Engineering 7th Edition*, Pearson Addison-Wesley, Boston, 2004, s. 694.

- Maliyet hesaplaması
- Projeye dair ön hazırlık yapılması

Planlama aşamasının sonucu proje planları, zamanlamalar, maliyet tahmini ve ön gereksinimleri içerir. Bu aşamada toplanan veriler ve kaynaklar ile projenin ilerleyen aşamalarında kullanılacak olan çözümlere şekil verilir. Son zamanlarda yükselişte olan yazılım geliştirme endüstri trendlerinin takip edilip en güncel bilgiyle bu aşamayı gerçekleştirerek kaynakların ihtiyaçları karşılamasını sağlamak yerinde bir karar olur.

1.4.1.2.Analiz

Analiz aşamasında geliştirilecek olan sisteme dair ürün gereksinimleri analiz edilir ve proje hedefleri, yazılım mühendisinin geliştirmeyi planladığı tanımlanmış sistem fonksiyonlarına dönüştürülür.[21]Analiz aşamasında şu üç temel aktivite gerçekleştirilir:

- Proje diyagramları oluşturma
- Proje için gerekli işlevsel gereksinimlerini araştırma
- Projeye dair detaylı bir analiz ortaya çıkartma

İşlevsel gereksinimlerin araştırılıp bir araya getirilmesi Yazılım Geliştirme Yaşam Döngüsünün en önemli kısımlarından biridir. İşlevsel gereksinimlerin analizi, projenin başarılı olabilmesi için son kullanıcının projeye dair istedikleri yani sistemin karşılaması gereken işlevler kümesidir.

Projeye dair yapılacak olan analiz, bir projenin ihtiyaçlarının ne olduğunu ve bunların nasıl karşılanabileceğini, projenin hangi kısımlarından kimin sorumlu

[21]Nalkar Sanjivani Baban, "Processing Models of SDLC", 2016, s. 1-2.

olacağını ve proje gerçekleştirilirken ne tür bir zaman çizelgesi beklenmesi gerektiğini belirlemede hayati önem taşır.

1.4.1.3.Tasarım

Projeye dair planlama ve analiz aşamaları tamamlandıktan sonra belirlenen gereksinimler doğrultusunda Tasarım aşamasına geçilir. Artık projenin ne olduğuna dair bir olgudan projenin nasıl gerçekleştirileceği basamağına geçiş yapılan aşamadır. Tasarım sırasında proje kısıtları, bütçe, kullanılacak teknolojiler müşteri ile görüşülüp tartışılır. Kullanıcı ara yüzü tasarlanır, önceki aşamalarda toplanılan veriler incelenir.[22]Proje tasarımında kullanılan çeşitli araçlar vardır; akış şemaları, veri akış diyagramları, veri sözlüğü, karar ağacı, karar tabloları ve veri tabanları kullanılabilir.

1.4.1.4.Geliştirme

Geliştirme aşaması istenilen ürünü oluşturma aşamasıdır. Bu adımda gerçek bir geliştirme başlar, diğer aşamalarda oluşturulan bütün dokümanlar gerçek bir sisteme dönüştürülür. Bu aşamanın düzenli ve pürüzsüz ilerlemesi için bir önceki aşamada iyi bir tasarım ve dokümantasyon yapılması gereklidir.

Geliştirme aşamasında gerçekleştirilen üç ana olgu şunlardır;[23]

- Sistem tasarımına göre yazılım yapısını geliştirme
- Tasarım şemasına göre veri tabanını hazırlayıp geliştirme
- Kod geliştirme ve hataları düzeltme

[22]Suryanto Nugroho, Waluyo, Sigit Hadi ve Hakim, Luqman "Comparative Analysis Of Software Development Methods Between Parallel, V-Shaped And Iterative", *International Journal of Computer Applications,* Sayı: 169, No: 11, 2017, s. 8.

[23]a.g.m, s. 8.

Tasarım aşamasında, yalnızca yazılım altyapısının taslağı sağlanırken, geliştirme aşamasında yazılımı geliştirecek olan yazılım mühendisi yazılım altyapısını desteklemek için gerekli yazılım ve donanımları satın alıp yazılımın geliştirilmesi için gerekli ortamı kurar. Daha sonra verilen açıklamalara ve detaylara göre veri tabanının oluşturulması ve esas kodlama başlayabilir.

1.4.1.5.Test

Test aşaması proje için oldukça kritik bir öneme sahiptir. Bu aşama ürünün müşterinin beklentilerini ve isteklerini karşılayıp karşılamadığını belirlemek için yapılır. Etkili bir test aşaması; projeye dair doğru ve güvenilir sonuçlar ortaya koyar, yüksek kalitede bir yazılım ürününün ortaya çıkmasına sağlar. Dolayısıyla bakım maliyetleri düşük olur.[24] Test aşamasında genelde yazılım önce çalışma ortamına alınır ve test edilerek olası hatalar tespit edilir. Daha sonra yazılım gerçek veriyle çalıştırılarak test edilir.

Test aşamasında test ekibi ve hata veya kusurun önlenmesini hedefleyen Kalite Güvencesi ekibi çeşitli hatalar/eksiklikler bulabilir ve bunları yazılımı geliştiren ekibe iletebilir. Yazılım ekibi hataları/eksiklikleri düzeltip tekrar test etmeleri için test ekibine gönderir. Bu aşama yazılım eksiksiz, hatasız, tutarlı ve müşterinin isteklerine göre çalışan bir sistem haline gelene dek devam eder.

[24]Sujit Kumar ve Dubey, Pushkar, "Software Development Lıfe Cycle (Sdlc) Analytical Comparison And Survey On Traditional And Agile Methodology", *Abhinav National Monthly Refereed Journal Of Research In Science & Technology,* Sayı: 2, 2013, s. 23.

1.4.1.6.Uygulama

Uygulama aşamasında sistem kullanıcının gerçek ortamına aktarılır ve gerçek kullanıcı sistemi kullanmaya başlar. Bütün veri ve bileşenler ürünün ortamına yerleştirilir. Bu aşama teorinin pratiğe dönüştüğü aşama olarak görülebilir.

Uygulama aşamasında genel olarak şu adımlar gerçekleştirilir;

- Kullanıcı tarafına yazılım ve donanımın tedarik edilip yüklenmesi
- Kullanıcıların eğitimi
- Dokümantasyon
- Veri dönüşümü

Yazılım ürünün düzgün bir şekilde işlemesi için doğru yazılım ve donanım bileşenlerinin kurulması elzemdir. Geliştirilen yazılım kullanıcıların daha önce kullandıkları verilerin kaynak olacağı bir yazılımsa veri dönüşümü en kritik adımlardan biridir. Veri kaybı yaşanmaması için bu adımın hassas bir şekilde yürütülmesi gerekmektedir. Uygulama kurulmadan evvel gerekli güvenlik ve olası bir duruma hazır olmak için kurtarma önlemleri alınmalıdır.

1.4.1.7.Bakım

Bakım aşaması yazılımın çalışma ömrü boyunca ortaya çıkan/çıkabilecek hataların ortadan kaldırıldığı, gereken yazılım güncellemelerinin yapıldığı ve yazılımın çeşitli çalışma ortamlarına adapte edildiği aşamadır. Bakım aşamasında muhtemel güncellemeler, en son güncellemenin getirdiği uyumluluk kıstaslarını ve diğer işlevsel özelliklerin kapsamı karşılanmalıdır.[25]Genel olarak yazılım sistemlerinde her zaman not edilmesi ve düzeltilmesi gereken çeşitli hataların yer aldığı görülmüştür. Aynı

[25]Mohit Kumar Sharma, "A Study Of SDLC To Develop Well Engineered Software", *International Journal of Advanced Research in Computer Science,* Sayı: 8, 2017, s. 521.

zamanda çalışan yazılım sistemleri de zaman zaman gözden geçirilmeli varsa güncellemeler ve değişiklikler yapılmalıdır.

Yazılım sisteminin bakımında önemli olan noktalar şunlardır:

- Sistemin bütün özelliklerini bilmek
- Yapılması gereken değişiklikleri veya ek gereksinimleri bilmek
- Performans üzerine çalışmak ve performansı arttırmayı hedeflemek

Bakım aşamasında şunlar yapılır:

- **Hata düzeltme**, test aşamasında test edilmeyen senaryolar ya da gözden kaçan durumlar sonucu ortaya çıkan hatalar düzeltilir.
- **Sürüm Yükseltme**, uygulamada kullanılan yazılımın yeni bir versiyona yükseltilmesidir.
- **İyileştirme**, var olan yazılıma yeni özelliklerin eklenmesidir.

Bu aşamanın amacı sisteme dair en başında belirlenen ihtiyaçların karşılanmaya devam etmesi ve sistemin ilk aşamada belirtilen teknik özelliklere göre çalışmasının sürdürülmesinin sağlanmasıdır.

Bir sistemde büyük bir değişiklik gerekiyorsa, değişikliği gerçekleştirmek için yeni bir projenin oluşturulması gerekebilir. Yeni proje bakımdan önce bahsedilen tüm aşamalardan geçerek yeni bir Yazılım Geliştirme Yaşam Döngüsüyle ortaya çıkacaktır.

1.4.2. Yazılım Geliştirme Yaşam Döngüsüne Alternatif Bir Bakış Açısı

Yazılım Geliştirme Yaşam Döngüsü literatürde çeşitli şekillerde ele alınmıştır. Bu yaklaşımlardan biri Yazılım Geliştirme Yaşam Döngüsü adımlarını dörde indirir ve bu adımlar aşağıdaki şekilde sayılabilir; [26]

- **Planlama:** Yazılıma dair planlamanın bütün gereksinimlerinin tanımlandığı aşamadır. Projeye gereken zaman ve kaynak ayrımı, personel, proje yapılan kurumun bilişim seviyesi ve yeni kurulacak sistemin kurumda kullanılabilirliği, yapılan sistemin uzun vadede bakımı ve sürdürebilirliği, projede gereken yazılım ve donanımlar gibi çok sayıda parametre bu aşamada incelenir. Bu aşamanın tamamlanmasının ardından elde edilen bulgular ile bir uygunluk raporu hazırlanarak kurum yönetimine sunulur ve bilişim sisteminin yapacağı olumlu katkıların maliyet analizine göre katma değeri olduğu kesinleştirilir.

- **Sistem Analizi:** Bu aşama sistem analizi yapılarak ve geliştirilmekte olan bilgi sisteminin etkilediği diğer birimler ve mevcut işleyiş tahlil edilerek gerçekleştirilir. Örneğin insan kaynakları ve yönetim sistemi olarak geliştirilen yazılımımız ele alınırsa, maaş-bordro hesaplamaları kısımlarında personelin kıdemi, derecesi, bulunduğu kademe ve maaşına etken olan kalemler gibi çok sayıda etkilediği diğer kayıtlar tahlil edilerek bu yazılım sistemi ile olan etkileşimi çalışılır.

- **Sistem Tasarımı:** Bu aşamada planlama ve analiz aşamalarında elde edilen bulgulara dayanarak sistemin bütün bileşenleri ve genel yapısı tasarlanır. Çok çeşitli tasarım yöntemleri ve aşamaları vardır, genelde eldeki veriler mantıksal bir tasarım aşamasından geçilerek sistemin hedefleri ve etkileşimde olduğu alt bileşenleri arasındaki uyum gözetilecek bir yol izlenir. Bu aşamadan sonra elde

[26]Benjamin Seaver Blanchard ve Fabrycky, Wolter Joseph, *Systems Engineering And Analysis 4th Ed.,* Pearson Prentice Hall, New Jersey, 2006, s. 31.

edilen mantıksal tasarımın fiziksel tasarıma çevrilmesi aşamasına gelinir. Mantıksal tasarımdan fiziksel tasarıma çevrilim aşamasında, programlama dilinden kullanılacak olan teknolojilere ve sistemin bütün parçalarına kadar veri tabanı tasarımı, kullanıcı ekranları, veri akış yolları gibi sistem tasarımı tamamlanır.

- **Uyarlama ve İşletme Aşaması:** Bu aşama, şimdiye kadar yapılan kağıt üzerindeki bütün işlemlerin gerçeğe dönüştürüldüğü ve eldeki veriler ile tasarımın yazılım haline getirildiği, elde edilen yazılım ürününün kurumun isteklerine ve amaçlarına uygun olup olmadığının değerlendirildiği aşamadır. Kodlama, kod kontrolleri, proje testleri, güvenlik testleri, projenin kuruma kurulumu, kullanıcı eğitimi, proje yönetimi gibi işlemlerin tamamı bu aşamada yapılır.

1.5.DOĞRUSAL, YİNELEMELİ, EVRİMSEL VE ARTIMLI GELİŞTİRME YAKLAŞIMLARI

Yazılım geliştirme yaşam döngüsü süreç modelleri tarafından benimsenen, yazılım geliştirme konusunda ortaya atılmış çeşitli yazılım geliştirme yaklaşımları vardır. Bu geliştirme yaklaşımları şu şekildedir;[27]

- Doğrusal geliştirme yaklaşımında yazılım geliştirme yaşam döngüsü aşamaları belli bir sırada ilerler.

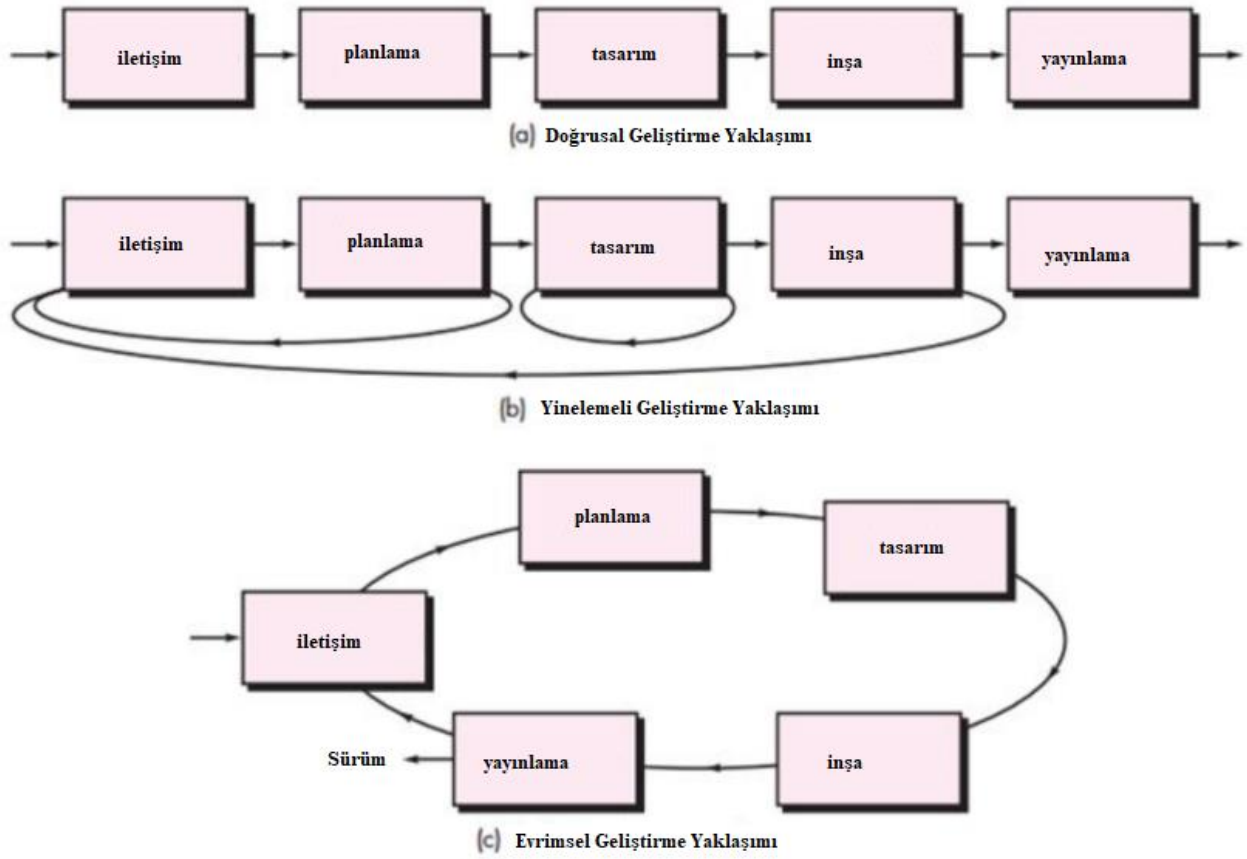
- Yinelemeli geliştirme yaklaşımında yazılım geliştirme yaşam döngüsü aşamalarının doğrusal bir şekilde takip edilmez, bazı aşamalar bir sonraki aşamaya geçilmeden önce bir veya birden fazla şekilde tekrar eder.

[27]Roger Pressman, *Software Engineering: A Practitioner's Approach 7th Edition*, McGraw-Hill, New York, 2010, s. 31-42.

- Evrimsel geliştirme yaklaşımında yazılım geliştirme yaşam döngüsündeki aşamalar döngüsel şekilde takip edilir.

- Artımlı geliştirme yaklaşımı yazılım geliştirme yaşam döngüsü içerisinde 'artım' isimli çıktılar oluşturularak, bu artımların kullanıcı için aşamalı olarak daha çok işlevsellik kazandırılmasıyla ilerleyen bir yapıya sahiptir.[28]

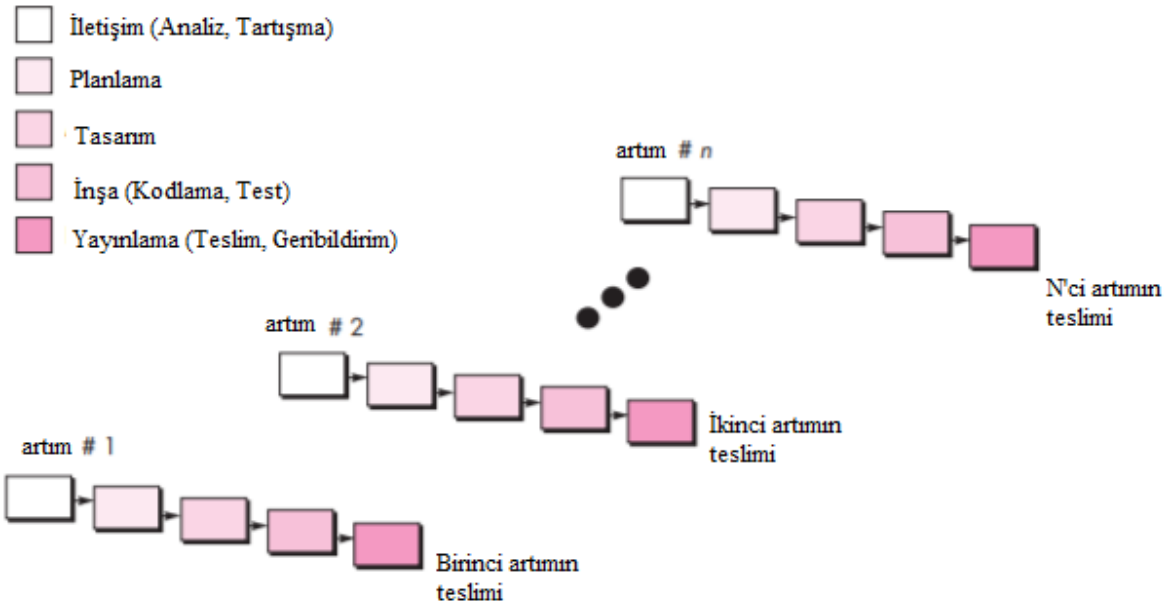
Doğrusal, Yinelemeli, Evrimsel Geliştirme Yaklaşımları Şekil 1.5.1'de, Artımlı Geliştirme Yaklaşımı Şekil 1.5.2'de görülebilir.



Şekil 1.5.1 Doğrusal, Yinelemeli, Evrimsel Geliştirme Yaklaşımları[29]

[28]a.g.e, s. 41-42

[29]Roger Pressman, *Software Engineering: A Practitioner's Approach 7th Edition*, McGraw-Hill, New York, 2010, s. 33.

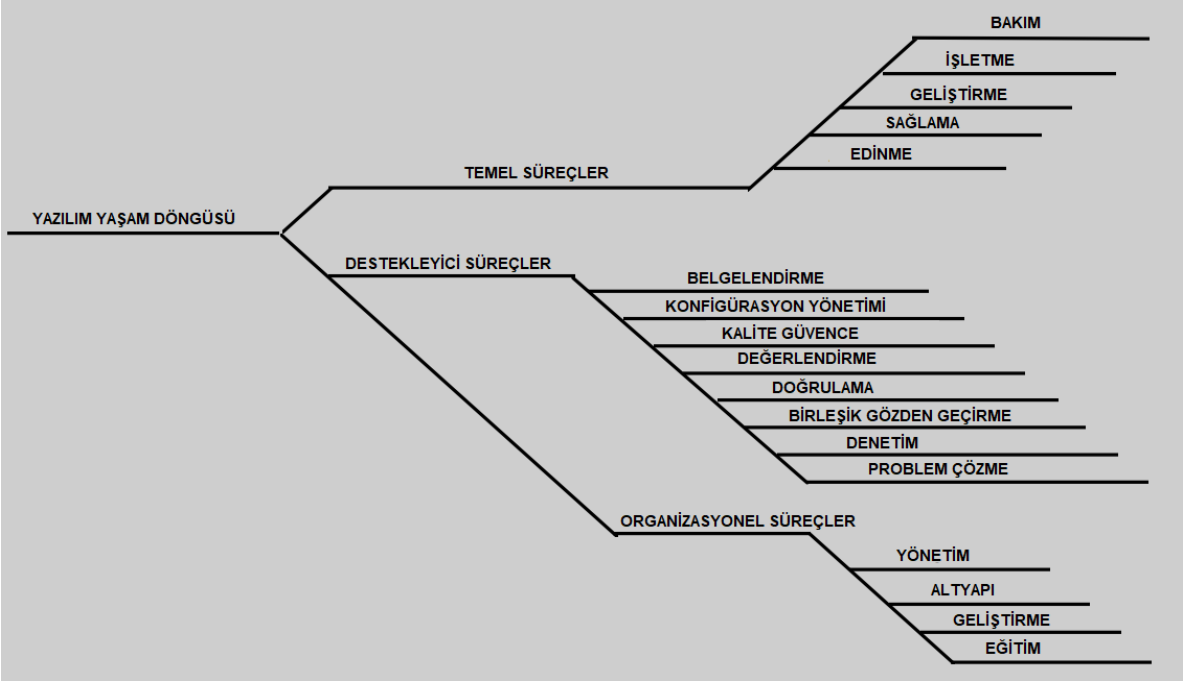


Şekil 1.5.2 Artımlı Geliştirme Yaklaşımı[30]

1.6 ISO/IEC 12207 YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ SÜREÇLERİ STANDARDI

ISO/IEC 12207 Yazılım Geliştirme Yaşam Döngüsü Süreçleri Standardı yazılımın ortaya çıkmasından ürün teslimatı sonrası bakım-destek vb. kadar bütün süreçleri kapsayan, aralarındaki bağlantıların tanımlandığı sistem mühendisliği ilkesine dayanır. Bu standart genel olarak herhangi bir yazılım yaşam döngüsü modeli ile birlikte kullanılabilir ve temel, destekleyici, organizasyonel olarak üç süreçten meydana gelir. Bu süreçlerin içerisinde süreçler boyunca gerçekleştirilen çeşitli faaliyetler yer alır. Bu faaliyetler bir araya gelerek bahsi geçen süreçlerin etkin bir şekilde yürütülmesini, ürünlerin/hizmetlerin kullanıcı beklentilerini karşılayan kalitede sunulmasını destekleyen süreçlerin belirlenmesi, tanımlanması, uygulanması ve ölçülüp değerlendirilerek sürekli iyileştirmesinin sağlanmasını hedefler. Şekil 1.6.1'de bu süreçlerin içerisinde yer alan çeşitli basamaklar görülebilir.

[30]Roger Pressman, *Software Engineering: A Practitioner's Approach 7th Edition*, McGraw-Hill, New York, 2010, s. 44.



Şekil 1.6.1 ISO/IEC 12207 Standardı[31]

Temel süreç yazılım yaşam döngüsünün başlıca basamakları edinme, sağlama, geliştirme, işletme ve bakım faaliyetlerinden oluşur. Destekleyici süreçler belgelendirme, konfigürasyon yönetimi, birleşik gözden geçirme, kalite güvence, değerlendirme, doğrulama, denetim ve problem çözme faaliyetlerinden meydana gelmektedir. Destekleyici süreç, özel bir işlevi yerine getirirken başka bir süreci destekler. Organizasyonel süreçler yönetim, altyapı, geliştirme ve eğitimden oluşur. Yazılımı geliştirecek olan personel bir yaşam döngüsü sürecini oluşturmak, kontrol etmek ve iyileştirmek için organizasyonel süreç kullanabilir.[32]

Yazılım yaşam döngüsü süresince meydana gelebilecek risklerin, sorunların ve yetersiz geliştirme sürecinin üstesinden gelmek ve kaliteyi iyileştirmek için bazı standartlaştırılmış yöntemlerin izlenmesine ihtiyaç duyulmaktadır. Bu nedenlerden dolayı ISO/IEC 12207, izlenmesi gereken adımları içeren güncel ve geçerli kullanım

[31]Raghu Singh, "International Standard ISO/IEC 12207 Software Life Cycle Processes", *Software Process: Improvement and Practice*, Sayı: 2, 1996, s. 36.

[32]A.g.m, s. 36.

standartlarından biri olarak öne çıkmaktadır. Yazılım geliştiricilerin standardın özünü kavrayarak ISO/IEC 12207'yi kendi projelerine uyarlama şansı vardır. Bu durum, riskleri azaltarak kalite ile ilgili faaliyetlere rehberlik sağlamak için iyi bir seçenektir, dolayısıyla proje başarısı için oldukça önemlidir. Uyarlama sırasında projeye dair projenin büyüklüğü, projenin bütçesi, projede kullanılabilecek teknolojiler, projenin zaman aralığı vb. gibi kıstaslar incelenebilir.

ISO/IEC 12207 Yazılım Geliştirme Yaşam Döngüsü Süreçleri Standardını en verimli biçimde kullanmak için göz önüne alınması gereken bir takım ön koşullar vardır. Bunlar şu şekildedir:[33]

- Nitelikli personel ihtiyacı
- Kurumun politikalarını anlama gereksinimi
- Proje ortamıyla ilgili deneyime sahip olma
- Standarda dair bir anlayış geliştirme

Yazılım yaşam döngüsü süreçleri ile ilgili ortak bir çerçeve oluşturmayı hedefleyen ISO/IEC 12207 Standardı çeşitli yazılım projelerinde kullanılabilir. Standardın mimarisi gelişen modern yazılım yöntemleri, teknikleri ve araçlarıyla birlikte kullanılmaya uygun olduğu için ortaya iyi bir yazılım çıkarmak adına göz önünde bulundurulması gerekmektedir.

[33]Ufuk Aydan, Yılmaz, Murat ve O'Connor, Rory, "Towards a Serious Game to Teach ISO/IEC 12207 Software Lifecycle Process: An Interactive Learning Approach." *15th International Conference SPICE*, Sayı: 526, 2015, s. 6.

İKİNCİ BÖLÜM

YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ: GELENEKSEL YÖNTEMLER

Yazılım geliştirme adımlarının sistematik bir şekilde ilerleyebilmesi için yazılım geliştirme yöntemlerine ihtiyaç duyulmaktadır. Böylece yapılan ürün sadece belirlenen zaman aralığında bitirilmiş olmakla kalınmaz aynı zamanda iyi kalitede ve kullanıcı dostu bir yazılım da ortaya çıkarılabilir.

Geleneksel yöntemler; proje geliştirmenin eksiksiz bir gereksinim setinin belirlenip belgelenmesiyle başladığı, ardından proje mimarisinin ve üst düzey tasarım geliştirme ile denetimin izlediği plan odaklı yöntemlere denilmektedir. Projeye dair iş yükünün ağır olmasından ötürü ağırsiklet (heavyweight) olarak da adlandırılmaktadır.[34]

Sommerville yazılım geliştirme yöntemlerinin yazılımın geliştirildiği süreye dair bir temsil olduğunu, her yöntemin belirli bir bakış açısından sunulan bir süreç olduğunu ve bu sebeple her modelin yalnızca o süreç ile ilgili kısmi bilgi sağladığını savunmaktadır. Örneğin bir yazılım geliştirme yaşam döngüsü modeli bir aktiviteyi ve onun aşamalarını kesin bir şekilde belirleyebilir fakat bu aktivitelere katılacak kişilerin rollerini belirleyemeyebilir.[35] Bu konu yazılım geliştirme sürecine dâhil olan kişilerin kararına bırakılabilir.

Günümüze bakıldığında birden fazla SDLC modeli ve yöntemi olduğu görülmektedir zira yıllar içerisinde değişen zaman, ihtiyaç ve taleplere göre bir sürü SDLC modeli ve yöntemi ortaya atılmış ve kullanılmıştır. Hepsinin birbirine göre

[34]Mahdi Javanmard ve Alian, Maryam, "Comparison Between Agile And Traditional Software Development Methodologies" *Cumhuriyet Üniversitesi Fen-Edebiyat Fakültesi Fen Bilimleri Dergisi*, Sayı: 36, No:3, 2015, s. 1386.

[35]I. Sommerville, 2004, s. xxvi

çeşitli avantajları ve dezavantajları bulunmaktadır ve her model ile yöntem farklı projeler için uygundur, tek bir modelin bütün yazılım projeleri için uygun olduğu savunulamaz.

2.1 ŞELALE MODELİ (WATERFALL MODEL)

Datyal'a göre Şelale Modeli, yazılım mühendisliğinin ilk ve en popüler Yazılım Geliştirme Yaşam Döngüsü modelidir. Bir önceki aşama bittikten sonra sıradaki aşamaya geçilen, oldukça kolay anlaşılabilen bir modeldir.[36] Günümüzde kendinden sonra gelen çoğu yöntembilimine kaynak ve temel olarak görülmektedir.[37] Bu model 1970 yılında Royce tarafından sunulmuştur.[38] Yazılım mühendisliğinin ortaya çıkardığı en temel modelidir. Büyük şirketlerin projeleri ve hükümet projelerinde geniş çaplı bir şekilde kullanılan en eski modellerden biri olan Şelale Modeli ayrıca 'Doğrusal Sıralı Model' ve 'Klasik Yaşam Döngüsü' olarak da isimlendirilmektedir.

Şelale Modeli diğer modellere temel sağladığı kabul edilen klasik bir modeldir. Bu nedenle bu modelde diğer modellere örnek teşkil edip aktarılan çeşitli yazılım projelerini yönetmede kolaylık sağlayacak basamaklar bulunmaktadır. Şelale Modeli genel olarak İhtiyaç Analizi, Tasarım, Uygulama, Test ve Bakım şeklinde beş basamaktan oluşmaktadır. Şekil 2.1.1'de de görülebildiği gibi bu basamaklar tekrar edilmemek üzere tasarlanmıştır ve aşağıya doğru şelale şeklinde akar gibi ilerlerler. Daha sonra Royce bu modeli; sistem gereksinimleri, yazılım gereksinimleri, analiz, program tasarımı, kodlama, test ve uygulama olarak revize edip basamak sayısını yediye çıkarmıştır.[39]

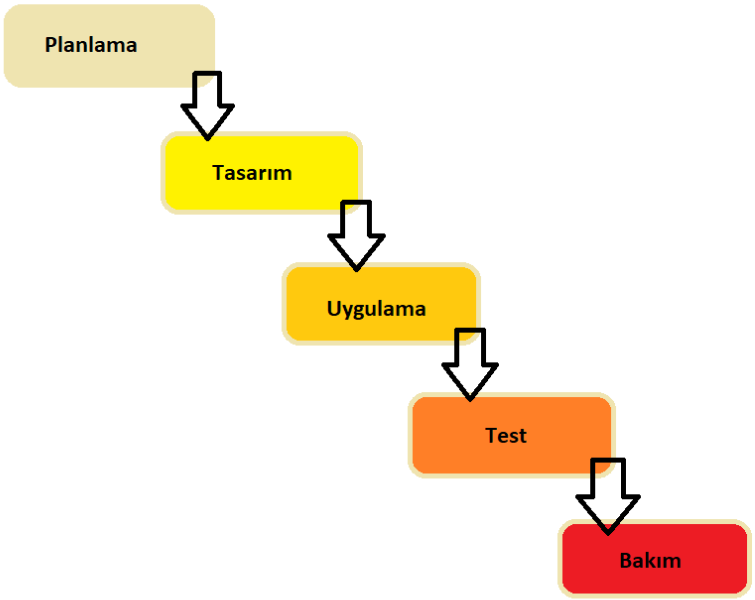
---

[36]Deeksha Datyal, "Proposed Model to Overcome the Problems in Waterfall Model", *International Journal of Computer Science & Information Technology*, Sayı: 2, No. 2, 2015, s. 29-32.

[37]Nabil Mohammed Ali Munassar ve Govardhan, Dr. A. "A Comparison Between Five Models Of Software Engineering", *International Journal of Computer Science Issues - IJCSI*, Sayı: 7, No: 5, 2010, s. 95.

[38]Winston Royce, "Managing The Development Of Large Software Systems: Concepts And Techniques", *Proceedings of the 9th international conference on Software Engineering,*1987, s. 328–338.

[39]A.g.m., s. 330.



Şekil 2.1.1 Şelale Modeli

Şelale Modeli katı bir yapıya sahiptir. Yazılım projeleri yapı olarak genelde esnek ve değişmeye uygun yapılar olduğundan Şelale Modeli çoğu yazılım projesi için pek uygun değildir. Şelale Modelinin temel amacı; ortaya çıkarılacak ürüne dair proje süresince değişikliklere izin vermeden kapsam, süre ve kaynaklar gibi faktörleri en baştan sabitlemek ve büyük bir risk faktörü olan 'değişimi' etkisiz kılmaktır. Yani bu modelde bir yazılım projesinin planı tamamlandıktan sonra ne kadar müdahale edilirse projenin o kadar karmaşıklaşacağı, sürecin uzayacağı dolayısıyla maliyetin aşılacağı değerlendirilmektedir. Bu durum Şelale Modelinin ilk ortaya çıktığı zamanlara bakıldığında anlaşılabilir bir durum olarak değerlendirilebilir zira iş süreçleri günümüzde olduğu gibi ilerlemiyor ve projelerin hemen bitirilmesi gerekmiyordu. Teknoloji ve kaynak sınırlı olduğu için bilgi işlem işleri donanım üzerinden ilerliyor ve yazılım ile de genel olarak bu donanımların depolama kapasitesi ve işlemci gücü gibi kısıtlı kaynakları en optimum şekilde kullanmak hedefleniyordu.

Şelale Modeli ile bir projenin tasarlanması sırasında benimsenen yaklaşımlar şunlardır:

- Bütün aşamalar sıralı bir şekilde ve tek yönlü gerçekleştirilmektedir.
- Projenin gereksinimleri sürecin başında oldukça iyi bir şekilde belirlenmelidir. Müşteri gereksinimlerin belirlenmesi sırasında projeye dâhil olur; Tasarım, Uygulama ve Test sırasında yer almaz. Daha sonra tekrar Bakım-Entegrasyon aşamasında ortaya çıkar.
- Sisteme dair bütün gereksinimlerin önden bilinebileceği ve daha sonra bu gereksinimlerin herhangi bir şekilde değişmeyeceği varsayılarak proje ortaya çıkarılmalıdır.
- Analiz, tasarımdan evvel tek seferde yapılır.
- Tasarım kodlama safhasından evvel tek seferde yapılır.
- Bütün kodlamalar yapıldıktan sonra test ve bütünleştirme safhasına geçilir.
- Bütün testler tamamlandıktan ve bütünleştirme aşamaları yapıldıktan sonra ürün kullanıma alınır.[40]

2.1.1. Şelale Modelinin Avantajları

- Anlaması kolay bir modeldir.
- Projenin başında gereksinimler anlaşılır ve net olarak tanımlanır.
- Sınırları belli olan, gereksinimleri iyi tanımlanmış projeler için uygundur.
- Doğru bir zamanlama ve maliyet tahmini projedeki belirsizlikleri azaltır.
- En eski modellerden biri olduğu için yazılım mühendisleri tarafından genel olarak teoride iyi bilinir.
- Model aşama aşama ilerlediği ve bir aşama bitmeden diğeri başlamadığı için projenin ilerleyişini takip etmek kolaydır.
- Modelin aşama aşama olması eş zamanlı ilerleyen süreçler olmaması durumunu ortaya çıkardığından kaynak takibi kolay yapılır.
- Tasarım aşamasından önce tanımlamalar yapılır.

[40]Steve Palmquist, Lapham, Mary Ann, Miller, Suzanne, Chick, Timothy ve Ozkaya, Ipek, *Parallel Worlds: Agile and Waterfall Differences and Similarities*, Carneige Mellon University, Pensilvanya, 2013, s. 5-6.

- Kimin neyi nasıl ve ne zaman yapacağı belirli olduğundan, basit ve sınırlı projeler için tercih edildiğinden tecrübesi olmayan proje elemanları ve proje yöneticileri içeren proje ekipleri için uygun bir modeldir.
- Şelale modeli disiplinli ve katı bir model olduğu için projede çalışan proje elemanlarına ve yöneticilerine projenin işleyişi ile ilgili güven verir, bilinçli bir şekilde yönetildiği ve her şeyin kontrol altında olduğu hissiyatını verir.
- Her aşamada dokümantasyon yapıldığı için aşamaların takibi kolaydır.
- Müşterinin projeye ne zaman ve nasıl dâhil olacağı bellidir, bu sayede müşteriye dair belirsizlikler ortaya çıkmaz.
- Bir proje elemanı kendi aşamasını bitirdikten sonra bir daha projeye dönmeyeceği için birden fazla projede görev alan personelin bulunduğu kurumlarda, personelin işlerinin üst üste binmesi engellenir.
- Hazırlanan detaylı ve kapsamlı dokümantasyon sayesinde büyük ekipler arasında iletişim kurulur ve bütün elemanlar aynı ortamda olmadan da proje ilerlemeye devam eder.

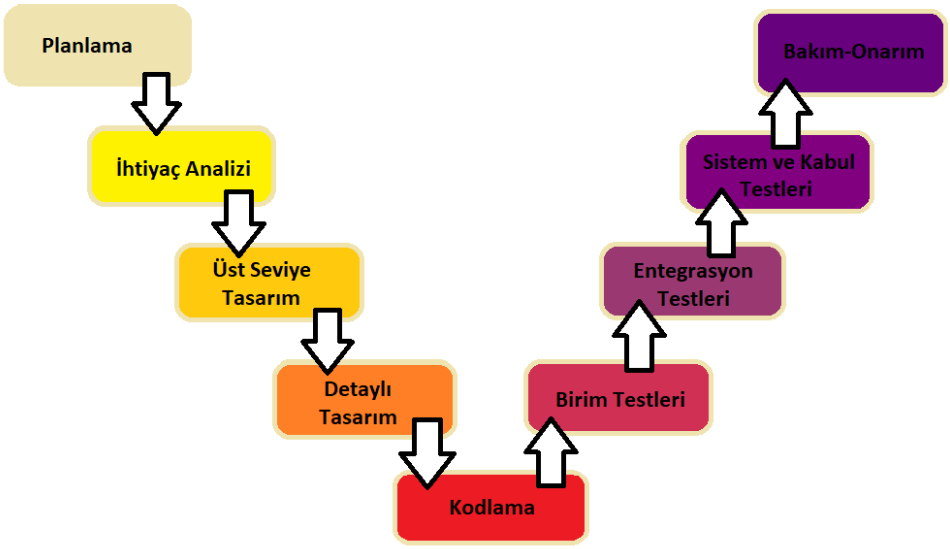
2.1.2. Şelale Modelinin Dezavantajları

- Şelale modeli katı bir yapıda olduğu için projenin herhangi bir yerinde değişiklik yapmak oldukça zordur.
- Sistemin tüm performansı kodlama aşamasına geçilene kadar bilinemez ve yazılım ürününde oluşan problemler ancak test aşamasında ortaya çıkar.
- Geriye dönülmeden ilerlendiği için herhangi bir aşamada meydana gelen bir problem diğer bütün aşamaları etkiler.
- Gereksinim analizi iyi yapılmadığı takdirde istenilen ürüne ulaşılamaz. Genel olarak projeye dair bütün gereksinimlerin ta en başından tamamen eksiz belirlenmesi mümkün değildir. Müşterilerin gereksinimlerini tam ve doğru ifade edememesi, zaman içerisinde farklı ihtiyaçları ve isteklerinin doğması proje bitiminde eksik ya da hatalı bir ürün ortaya çıkmasına sebep olur.
- Proje üzerinde değişiklik yapmak, iyileştirme yapmak, geliştirme yapmak ya da düzeltme yapmak için geriye dönmek zor ve maliyetlidir. Bu sebeple gereken değişikliklere hemen cevap veremez ve gelen istekler genelde geri çevrilir.

- Gerektiği halde yazılımda ve tasarımda değişiklik yapılması zordur.
- Yapısal olarak katı ve projede neyin ne zaman yapılacağı, kimin neyi ne zaman yapacağı sıkı kontrollerle belirlendiği için esnek olmayan, hantal bir yapısı vardır.
- Proje elemanlarının proje gereksinimlerini müşteri/kullanıcı ile iletişime geçmeden analiz ve tasarım aşamalarında hazırlanan dokümanlardan okumaları gereksinimlerin anlaşılması hakkında sorun çıkarır ve istenilen ürünü ortaya çıkarma konusu zorlaşır.
- Her aşama dokümantasyon gerektirir, bu da fazla belge oluşturmaya ve karmaşıklığa sebep olur. Genelde geliştiriciler tarafından yük olarak görülen bir faaliyettir ve dokümanlar aksatılır.
- Dokümantasyonun kullanıcı/müşteri tarafından tamamen anlaşılması ve dolayısıyla uygulanması zordur.
- Aşamalar sırayla ilerlediği için projenin herhangi bir yerinde meydana gelen bir problem sebebiyle diğer aşamalar sekteye uğrayabilir. Bu da diğer aşamalarda yer alan proje elemanları için zaman israfına sebep olur.
- Proje büyüdükçe proje aşamalarındaki ve dokümantasyonlarındaki karmaşıklık artar.
- Yazılım geliştirmek yaratıcılık ister fakat şelale modelinin katı ve disiplinli yapısı bunun önüne geçer.

2.2. V-ŞEKLİ MODELİ

V-Şekli Modeli aşamaların ardı ardına gerçekleştirildiği Yazılım Gerçekleştirme Yaşam Döngüsü modelidir. Şekil 2.2.1'de de görülebileceği gibi bu modelde aşamalar doğrusal bir şekilde aşağı inmek yerine kodlama aşamasından sonra yukarı bükülerek tipik V şeklini oluşturur. V-Şekli Modelinde üretilen yazılımın belirlenen kıstaslara uygun olup olmadığını ve ortaya çıkan ürünün müşterinin ihtiyaçlarını karşılayıp karşılamadığını kesinleştirmeye yarayan kontrol işlemleri olan onaylama ve doğrulama ön plana çıkar.



Şekil 2.2.1 V Şekli Modeli

Vee Modeli olarak da bilinen V-Şekli Modeli NASA tarafından geliştirilmiş olup ilk olarak 1991 yılında Chattanooga, Tennessee'de gerçekleştirilen NCOSE sempozyumunda kamuoyuna sunulmuştur. Forsberg ve Mooz V-Şekli Modeli "proje döngüsünün teknik yönü" olarak tanımlar.[41] Bu model Şelale Modeline benzemekte olsa da yapısal olarak farklılık gösterir. Şelale Modelinde olduğu gibi planlama, ihtiyaç belirleme, üst seviye ve alt seviye tasarımlar vardır ve sonrasında kodlamaya geçilir. Buraya kadar Şelale Modeli ile benzerlikler bulunmaktadır çünkü aşağı doğru akan bir proje yönetimi söz konusudur fakat kodlama aşamasından sonra yukarı doğru çıkılmaya başlanır ve test aşamalarına geçilir. Bu modelde öne çıkan özellik test kavramıdır. Modelin sağ tarafında yukarı ilerleyen kısmında çeşitli testler aracılığıyla ile olası hataları tespit edip erkenden ortadan kaldırma amacı güdülmüştür. Testler geliştirilmekte olan ürünün, ihtiyaç analizi aşamasında belirlenen gereksinimleri karşılayıp karşılamadığını kontrol etmek için kullanılır. Geliştirilen ürüne birim testleri yapılır. Her üretilen modül testten geçirilir. Daha sonra bu modüllerin/alt

[41]Kevin Forsberg ve Mooz, Harold "The Relationship of System Engineering to the Project Cycle", *The 12th INTERNET World Congress on Project Management*, 1991, s. 4.

seviye tasarım ürünlerinin birbiri ile çalışma durumu ile ilgili entegrasyon testleri yapılır ve kabul testleri aşamasında müşterinin kabul edip etmemesi ile ilgili müşteriye gidilir.[42] Müşteri uygulamayı test eder. Sonra bakım başlar. Yani kısaca modelin sol tarafı üretim sağ tarafı ise test aşamalarını içerir.

V-Şekli Modeli ürüne dair belirsizliklerin az olduğu, gereksinimlerin iyi tanımlandığı, iş tanımlarının açık olduğu bilişim projeleri için uygun bir modeldir. Bu model ile bir projenin tasarlanması sırasında benimsenen yaklaşımlar şunlardır:

- Yazılım gereksinimleri herhangi bir eksik olmadan açık ve anlaşılır bir şekilde tanımlanmalı ve yazılım geliştirme sürecinde yer alan yazılımcılar tarafından iyi bilinmelidir.
- Ürün tanımı kesin olmalı, değişiklik göstermemelidir.
- Yazılımı geliştiren yazılımcılar yazılım geliştirme teknolojilerini ve araçlarını kullanmayı iyi bilmelidir.
- Ürünün geliştirilmesi yapılmadan önce sistem test planı oluşturulmalıdır. Test planı gereksinimlerin toplanması sırasında belirlenen, projenin işlevselliğini karşılamaya odaklanır. [43]

2.2.1 V-Şekli Modeli Avantajları

- Modelin kullanımı Şelale Modeline göre basit ve kolaydır.
- Test adımları kodlamadan önce planlanır. Bu şekilde hem zaman kazanılır hem de başlangıç aşamasında projenin iyi anlaşılmasını sağlar.
- Her aşamanın spesifik bir çıktısı vardır.
- "Test" kavramını projenin başında yaşam döngüsüne dâhil ettiği için Şelale Modelinden daha başarılı olma ihtimali taşımaktadır.
- Küçük çaplı projelerde daha iyi sonuç verir.

[42]V. Rastogi, , 2014, s.168-171

[43]Gourav Khurana ve Gupta, Sachin, "Study & Comparison of Software Development Life Cycle Models". *International Journal of Research in Engineering & Applied Sciences*, Sayı: 2, No: 2, 2012, s. 1516-1521

- Bu model sadece yazılım ürünü için değil bütün iç ve dış sonuçlar için doğrulama ve onaylama yapmayı destekler.
- Müşteriyi/Kullanıcıyı projeye daha çok dâhil eden bir yapısı vardır. Bu sayede müşterinin/kullanıcının istediği gibi bir ürünün ortaya çıkma olasılığı daha yüksektir.
- Gereksinimler açık, anlaşılır ve kolay anlaşılacak şekilde belirtildiğinde iyi bir sonuç verir.
- Ürünün doğrulama ve onaylaması ürün geliştirmenin erken safhalarında yapılır.
- V-Şekli model, sistemi tasarlamadan önce gereksinimlerin tanımlanmasını, bileşenleri oluşturmadan önce yazılımın tasarlanmasını teşvik eder.
- Geliştirme sürecinin oluşturması gereken ürünler iyi tanımlanmalıdır, her sonuç test edilebilir olmalıdır.

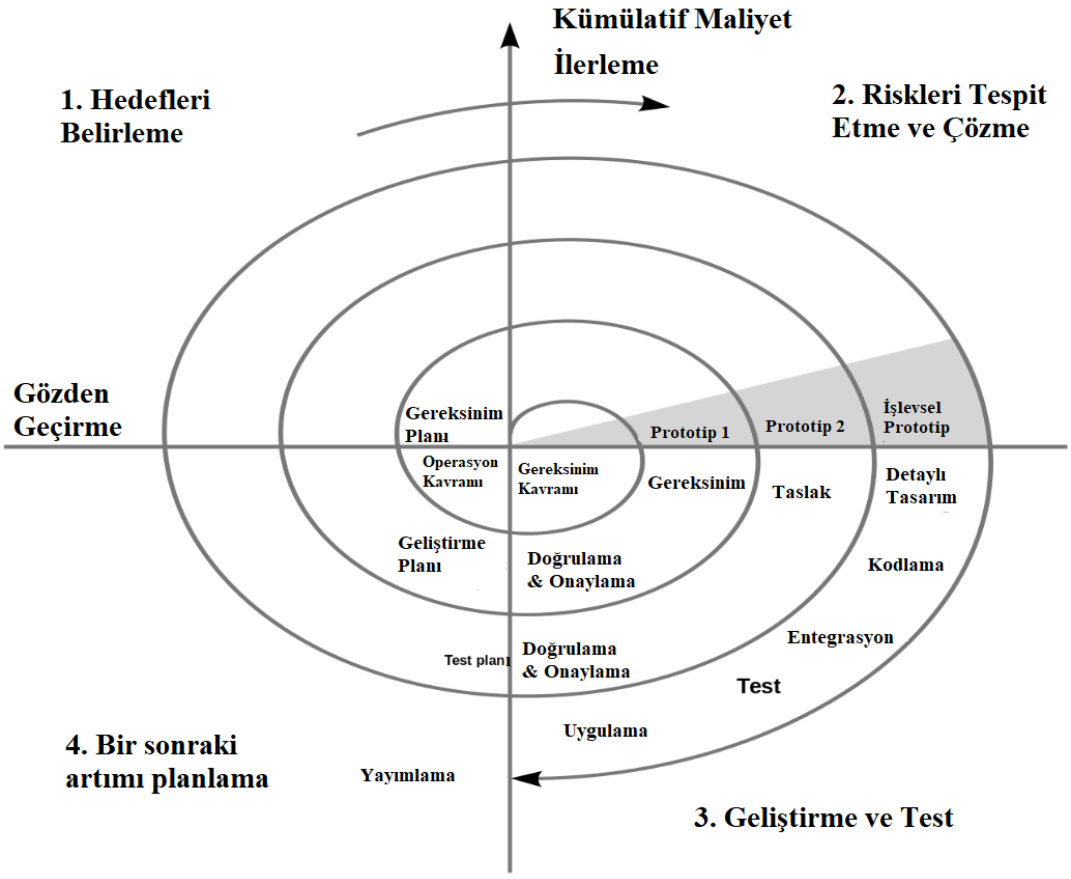
2.2.2 V-Şekli Modeli Dezavantajları

- Bu model Şelale Modelinden devşirme olduğu için esnek olmayan bir yapıya sahiptir. Değişikliğe cevap verme konusunda zorlanır, Şelale Modeli gibi katı bir modeldir.
- Projenin kapsamını ayarlamak zor ve maliyetlidir.
- Yazılım kodlama aşamasında geliştirilir yani yazılımın erken prototipleri yoktur.
- Test sırasında bulunan problemlere yönelik ne yapılacağına dair açık bir yol gösterilmemiştir.
- Detaylı bir planlama beraberinde daha fazla maliyet ve zaman getirir.
- Kullanışsız test yöntemleri oluşturur.
- Projenin ileri safhalarında herhangi bir değişiklik yapıldığı zaman gereksinim dokümanlarının da güncellenmesi gerekir.

2.3 HELEZONİK (SPİRAL) MODEL

Helezonik Model, risk odaklı bir yazılım geliştirme yaşam döngüsü modelidir. Yoğun sistemlerde gerçekleşen eşzamanlı mühendisliklere rehberlik etmek için

kullanılır. İki ana ayırt edici özelliği vardır. Bunlardan biri, sistemin tanım ve uygulama derecesini kademeli olarak artırmak için döngüsel bir yaklaşım benimsemiş olmasıdır. Diğeri ise hem üretici hem de kullanıcının onaylayacağı, uygulanabilir ve iki tarafı da tatmin edecek sistem çözümleri oluşturmak adına çeşitli dönüm noktaları oluşturmasıdır. Şelale yaklaşımı tanımlanmamış adımların kullanılmasını zorunlu kılsa da doğrusal olarak ilerleyen doğası, bu yöntemin en büyük sorunudur. Model herhangi bir ara işlemden ortaya çıkan beklenmedik çıktılara nasıl yanıt verileceğini tanımlamaz. Barry Boehm bu soruna çözüm olabilmesi için Helezonik Modeli sunmuştur.[44] Bu modelde her şelale aşaması risk değerlendirme ve prototip aktivitesi ile son bulur. Şekil 2.3.1'de modelin aşamaları görülebilir.



Şekil 2.3.1 Helezonik Model[45]

[44]B. Boehm, 1988, s. 61- 72.

[45]Nayan Ruparelia, "Software Development Lifecycle Models", *ACM SIGSOFT Software Engineering Notes*, Sayı: 35, No: 3, 2010, s. 10.

Helezonik Model, diğer modellerden farklı olarak oluşturulacak ürününün, yazılım geliştirme sürecini oluşturan aşamalardan tekrar tekrar geçilmesini ve her geçişte projenin ilerleme kat etmesini hedeflemektedir. Diğer modeller tasarımı doğrusal bir aşama olarak görürken Helezonik Model süreci bir spiral olarak ele alır, yineleyici döngüler genişleyen bir spiral olarak temsil edilir. Bu model yinelemeli modelin niteliklerini alıp bu avantajları Şelale Modelinin sadeliğine ek ağır bir risk analizi özelliği ile birleştiren sistematik ve yapılandırılmış birleşim modelidir. Bu model, yazılım ürün sürümlerinin artımlı olarak güncellenmesini sağlar. Helezonik model risk-güdümlü bir model olup, prototip geliştirmeler yaparak bu prototipler sayesinde risklerin bulunması ve çözümlenmesinin ardından geliştirmeye geçilmesini önerir. Oluşturulan prototipler projenin istenilen şekilde ilerleyip ilerlemediğini, önceki aşamalara dönülüp dönülmemesinin tespitini ve projenin sonlandırılıp sonlandırılmayacağını belirlemek için kullanılır. Helezonik Modelde iç kısımlar genelde gereksinimlerin rafine edilip prototipleme yapılmasıyla beraber gereksinim-ihtiyaç analizinin erken evresini, dış kısımlar ise yazılımın tasarım aşamasını temsil eder. Her helezondaki her adım için bir risk değerlendirme aşaması vardır. Risk odaklı yapısı Helezonik Modelin adımlarının tanım odaklı, prototip odaklı, simülasyon odaklı, otomatik transformasyon odaklı herhangi bir yazılım geliştirme yaklaşımına uyum sağlamasını mümkün kılar. [46]

Prototip model literatüre 1980'li yıllarda kazandırılan bir modeldir. Helezonik model de prototopi modele benzer bir yapı benimsemektedir. Her döngünün mevcut spiralin gözden geçirilmesi ve takip eden bir sonraki aşamanın planlanması için gözden geçirilme aşaması vardır. Helezonik Modelin çalışması dört faza ayrılmaktadır; planlama, risk yönetimi, üretim ve kullanıcı değerlendirmesi. Proje sona erene dek bu dört faz tekrar edilir. Örneğin bir yazılım projesinde her döngüde yeni bir prototip ortaya çıkarılarak toplamda 4 prototip oluşturulursa bu 4 prototipi ortaya çıkarmak için 4 döngüde ilerlemesi gerektiğinden proje içeriden dışarıya doğru spiral şeklinde aşama aşama ilerlenerek gerçekleştirilmelidir.

[46]B. Boehm, 1988, s. 61- 72.

Helezonik modelin ilk fazında amaçlar, alternatifler ve sınırlar belirlenir. Daha sonra eldeki verilerle riskler incelenip alternatifler değerlendirilerek bir prototip oluşturulur. Bu prototipin iş gereksinimlerine uygunluğuna bakılır. Ardından diğer faza geçilerek risk analizi, değerlendirmesi ve çözümlemeleri yeniden yapılır ve yeni bir prototip elde edilir, yazılım ihtiyaçlarına uygunluğu kontrol edilerek ihtiyaçlara uygun bir çözüm olup olmadığı belirlenir. Daha sonra geliştirme fazına geçilir. Geliştirme fazında kodlama, risk analizi, risk değerlendirmesi ve çözümlemeleri yapıldıktan sonra yeni bir prototip oluşturulur ve bu prototip üzerinden yazılım ürün tasarımı, tasarım onaylama ve doğrulaması ile entegrasyon ve test planlanır. Bu çıktıların risk analizi, değerlendirmesi ve çözümlemesinden sonra işleyen bir prototipe ulaşılır. Ayrıntılı tasarım ve kodlama; birim, entegrasyon ve kabul testleri yapılarak ürün gerçek ortama alınır ve kullanıcı değerlendirilmesine sunulur.

Helezonik model, son derece kişiselleştirilmiş bir yazılım ürünü ortaya çıkarmak için en uygun modeldir çünkü bu modelde kullanıcı etkileşimi ve değerlendirmesi geliştirmenin ilk aşamasından başlar, ancak sürekli değişiklik, güncelleme istemeye açık olması spiral şeklinde devam eden bir proje yaratma riski taşımasına uygun bir yapı oluşturmaktadır. [47]

Helezonik Modelin öne çıkan özellikleri şunlardır;

- Yazılım geliştirme süreci arka arkaya devam eden sıralı aktiviteler şeklinde gösterilmek yerine spiral şekilde ifade edilir.
- Bu modelde önemli olarak risk analizi ön plana çıkmaktadır.
- Asıl amaç her bir artırımın güvenli ve hızlı yapılmasını sağlamaktır.
- Helezonik modelde her bir artırımda yazılımın yeni bir prototipi ortaya çıkar.
- Geliştirme, ortada çalışan bir sistem varsa herhangi bir helezon sonucunda sona erdirilebilir.

[47]Sahil Barjtya, Sharma, Ankur ve Rani, Usha, "A Detailed Study Of Software Development Life Cycle (SDLC) Models" *International Journal of Engineering and Computer Science*, Sayı:6, No: 7, 2017, s. 22099.

- Müşteri ile iletişime geçilerek gereksinimlerin belirlenir. Gereksinimler iyi, açık ve anlaşılır bir şekilde belirlendiği zaman modelin ilerlemesi daha kolay olur.
- Planlama aşamasında kaynaklar, zamanlama, proje sırasında kimin ne ile ilgileneceği, yapılacaklar, vb. gibi konular belirlenir.
- Her aşama sonunda teknik, mali ve politik riskleri belirleyen risk analizi yapılır, alternatifler belirlenir. Döngü boyunca risk değerlendirilmesi yapılır.
- Analiz ve Tasarım aşamalarında projenin gereksinim analizi ve tasarımı yapılır.
- Gerçekleme ve Kurulum aşamasında proje dokümantasyonları düzenlenir, kullanıcı eğitimleri verilir ve sonunda ürün müşterinin değerlendirmesine sunulur.
- Modelde evrimsel bir yazılım geliştirme yaklaşımı vardır.
- Modelin her bir döngüsü kendi içinde yapılması gereken çeşitli etkinliklere ayrılır ve bu etkinlikler projenin büyüklüğüne ve karmaşıklığına göre değişebilir.
- Başarı ihtimali diğer modellere kıyasla daha fazladır.
- Her döngü bir aşamayı ifade eder. Aşamalar doğrudan analiz, planlama, tasarım, kodlama şeklinde sıralanmaz. Bir aşama bittikten sonra yapılan risk değerlendirmesine göre bir sonraki aşamada gerçekleştirilecek faaliyete karar verilerek ilerlenir.

2.3.1 Helezonik Model Avantajları

- Helezonik Model Yazılım Geliştirme Yaşam Döngüsü modelleri arasında en esnek modellerden biridir. Geliştirme aşamaları projenin karmaşıklığına göre proje müdürü tarafından belirlenebilir.
- Sistematik olarak oluşturulan çıktıyı yineleyip geliştirerek ilerleme yaklaşımını benimsemiştir.
- Her yineleme her aşamada yapıldığı için projeyi izlemek kolaydır. Bu izleme uzmanlar veya ilgililer tarafından yapılır. Bu da modeli daha şeffaf hale getirir.
- Helezonik model müşteri katılımı güçlü bir onay modelidir.
- Bu model ile oldukça öznel bir ürün geliştirilebilir.
- İş gereksinimleri değişken olan yüksek riskli projeler için uygundur.
- Oluşturulan prototipler tekrar kullanılabilir, bu sayede daha çok verimlilik sunar.

- Risk yönetimi bu modelin esas özelliğidir ve bu özellik Helezonik Modeli diğer modelle karşısında daha tercih edilebilir kılar.
- Risk analizi sayesinde olası hataları ilk aşamalarda tespit edip ortadan kaldırır.
- Kullanıcılar oluşturulan prototipler üzerinden sistemi erken görebilirler.
- Bu modelde geliştirilme küçük parçalara bölünür ve en riskli kısımlar önce gerçekleştirilir.
- Modelin benimsediği riske duyarlı yaklaşım potansiyel zorlukları engeller. Döngülerin herhangi bir aşamasında bir değişiklikle karşılaşıldığında bu ihtiyacın karşılanması zor olmaz.
- Hataları erken gidermeye odaklanan bir yapısı vardır.
- Yaşam döngüsünün ileri safhalarında bir değişiklik ihtiyacı olduğunda, proje müdürü bu değişikliği kolayca halledebilir.
- Prototipler küçük parçalar halinde gerçekleştirildiği için projenin ilerlemesi sırasında maliyet tahmini kolaylaşır ve müşteri/kullanıcı sistemin yönetimi üzerinde kontrol kazanabilir.
- Model son aşamasına yaklaştığında müşterinin daha çok dahil olması sağlanır ve bu sayede müşterinin ihtiyaçlarını karşılayan bir ürün ortaya çıkar.

2.3.2 Helezonik Model Dezavantajları

- Bu model ortaya istenilen şekilde bir ürün çıkana kadar döngü içinde ilerleyeceği için kesin bir zaman ve maliyet tahmini yoktur.
- Her aşamada yapılan risk analizinin bedeli oldukça yüksektir. Dolayısıyla küçük ve düşük riskli projeler için pahalı bir yöntemdir.
- Projenin büyüklüğü dolayısıyla döngüler arttıkça karmaşıklık da artar.
- İstenilen sistem ortaya çıkana kadar döngü devam edeceği için spiral sonsuza gidebilir yani projenin asla bitmeme ihtimali vardır.
- Ara adımlar fazla olduğu için çok fazla dokümantasyon gerektirir.
- Risk analizi öznel risk değerlendirme deneyimine dayanır ve risk yönetimi uzmanlarına ihtiyaç duyulur.
- Yüksek riskli öğelere yoğunlaşmak, yüksek riskli öğelerin doğru belirlenmesini gerektirir ve belirlenmediği halde proje çıkmaza girebilir.

- Bu modelin etkili bir şekilde ilerlemesi için kurallar ve protokoller iyi takip edilmelidir fakat projenin geliştirilme süresi boyunca kuralları ve protokolleri izlemek oldukça zordur.
- Kullanıcıdan/müşteriden gelen çeşitli öznel düzenlemeler sebebiyle aynı prototipi bir sonraki aşamada kullanmak zorlaşır.
- Projedeki belirsizliklerin değerlendirilmesi için kapsamlı bir yeteneğe ihtiyaç duyar.
- Maliyetin daha yüksek olduğu, daha üst düzeyde karmaşıklık içeren sistem önkoşulları olan büyük projeler için verimli sonuç verir.

2.4. PROTOTİP MODELİ

Klasik Yazılım Geliştirme Yaşam Döngüsü modeli aşamaları temelde mantıklı olsa da iki büyük zayıflığı bulunmaktadır. Birincisi aşamalar tek tek ilerlediği için projenin zaman çizelgesinin belirsiz olması, ikincisi projenin değişen ihtiyaç ve taleplere göre geriye dönmesi gerekmesidir. Yazılım mühendisleri zaman içerisinde yazılımların değişimini gözlemleyerek ara değişikliklerin istisna değil bir kural olması gerektiği sonucuna varmış ve sistem tüm iş gereksinimlerini karşılayana kadar ara değişiklikleri de dahil etmek adına bir dizi yineleme yoluyla bir sistemin geliştirilmesini öneren Prototip fikrini ortaya atmışlardır.[48]

Prototip modeli, dokümantasyona odaklanmak yerine gerçek yazılımı oluşturmayı ön planda tutar. Bu şekilde, gerçek yazılım önceden yayınlanabilir. Bir prototip konsepti gösteren, tasarım seçeneklerini sunan ve problemleri tespit edip çözümler öneren, geliştirilmekte olan yazılım sisteminin ilk versiyonudur. Maliyetin kontrol altında tutulması ve sistem paydaşlarının yazılım sürecinin erken aşamalarında prototip üzerinde deneme yapabilmesi için hızlı ve yinelemeli bir prototip geliştirilmesi esastır.[49]

[48]Gerald Everett ve McLeod, Raymond, *Software Testing: Testing Across the Entire Software Development Life Cycle*, Wiley-IEEE Press, New Jersey, 2007, s.31-32.
[49]I. Sommerville, 2004, s. 43-44.

Prototipleme daha fazla kullanıcı katılımı gerektirir ve bu sayede kullanıcı ürünün gidişatı hakkında fikir sahibi olabilir. Bu yöntemle kullanıcının prototipi görüp proje ile etkileşimde bulunması sağlanır, daha iyi ve daha eksiksiz geri bildirimler ve isterlerde bulunabilen kullanıcı kendi istediği ürüne ulaşmak için geliştiriciye yardımcı olabilir. Kullanıcı tarafından incelenen prototipin varlığı, her iki taraf arasında ortaya çıkabilecek yanlış anlaşılmaların önüne geçer. Bu sayede geliştirme bittikten sonra ortaya çıkan nihai ürünün kullanıcının görünüm, his ve performans arzusunu tatmin etme olasılığı daha yüksektir.

Prototip yöntemi yineleme ile yazılım geliştirme olgusunu merkezde tutar. Yazılım geliştirme, gereksinim analizinden sonra başlangıç prototipi oluşturarak başlar ve döngüsel etkileşimler halinde ilerler. Döngünün boyutu süreç için önemlidir zira daha küçük bir döngü sırasında test ve hata ayıklama işlemleri daha kolay yapılır. Aynı zamanda risk yönetimi de daha kolaydır, çünkü riskli parçalar yapılan döngü tekrarları sırasında tespit edilir ve düzeltilir.

Prototip modelinin temel fikri gereksinim tespiti ve analizini tasarım ya da kodlama aşamasına geçmeden önce, gereksinimleri daha iyi kavrayan harcanabilir bir prototip oluşturmaktır.[50]

Prototip Modelinin aşamaları şu şekildedir:[51]

1. **Gereksinim Analizi:** Bu aşamada ürün gereksinimleri detaylıca tanımlanır. Özellikle kullanıcı ara yüzüne dair temel ürün gereksinimlerini anlamayı içerir. Gelecekte ürünü kullanacak olan kişilerle görüşmeler yapılır. İç tasarım ve performans ile güvenlik gibi dış hususlar göz ardı edilebilir.

[50]S. Kute ve D. S. Thorat, 2014, s. 778-779

[51]Kazim Ali, "A Study of Software Development Life Cycle Process Models", *International Journal of Advanced Research in Computer Science*, Sayı: 8, 2017, s. 15-23

2. **Başlangıç Prototipini Oluşturma:** Orijinal tasarımdan hedeflenen yazılım ürününün bir prototipi oluşturulur. Çok temel gereksinimleri ve kullanıcı ara yüzünü karşılayan bir başlangıç prototipi geliştirilir. Ürünün bütün özellikleri tam ya da olması gerektiği gibi çalışmayabilir ve geliştirilen prototipte müşteriye istediği ürünün aynı görünümünü ve hissini vermek için geçici çözümler kullanılır.

3. **Prototipin Değerlendirilmesi:** Geliştirilen ürün bütün güncellemeleri geçtikten sonra kullanıcıya ya da projenin paydaşlarına sunulur. Kullanıcı/paydaşlardan gelen değerlendirmeler ileride sistem geliştirmelerinde kullanılmak üzere organize bir şekilde bir araya toplanır. Ürünün değerlendirilmesi bittikten sonra zaman, iş gücü, bütçe gibi faktörlere göre geliştirilmeye alınır. Aynı zamanda esas uygulamanın teknik yapısı da kontrol edilir.

4. **Prototipin Gözden Geçirilmesi ve Geliştirilmesi:** Geribildirim ve yorumlar tartışılır ve esas ürünün zaman, maliyet kısıtları, teknik uygunluk gibi hususlar kullanıcıyla görüşülür. Oluşturulan başlangıç prototipi gelen yorumlara göre tekrar döngüye alınır ve geliştirilen yazılım müşterinin isteklerine uyana dek bu döngü devam eder.

Son sistem detaylı bir şekilde değerlendirilip test edilir. Büyük ölçekli arızaları önlemek ve kesinti süresini azaltmak için periyodik olarak bakım yapılır.

Prototip modeli istenilen sistemin son kullanıcı ile çok fazla etkileşimde bulunduğu yazılım projelerinde kullanılmalıdır. Örnek olarak çevrimiçi sistemler, çok fazla kullanıcılı ile etkileşim gerektiren web ara yüzleri Prototip Modeli için uygun projelerdir. Kullanım kolaylığı sağlayan ve son kullanıcı için minimum eğitim gerektiren bir sistemin kurulması biraz zaman alabilir. Prototipleme son kullanıcıların sürekli olarak sistemle çalışmasını ve kullanılabilir bir sistem oluşturmak için

prototipe dahil edilen geri bildirimler olmasını sağlar. Bu model iyi bir insan-bilgisayar ara yüz sistemi tasarlamak için mükemmeldir.

2.4.1. Yazılım Prototipleme Türleri

Yazılım sektöründe kullanılan farklı yazılım prototipleme türleri vardır. Prototip modeline göre yazılım geliştiren her yazılımcı elindeki projeye göre uygun bir prototipleme türü seçer. Bunlardan bazıları şunlardır:[52][53]

1. **Hızlı Prototipleme:** Bu prototipleme türü minimum gereksinim analiziyle az bir çaba kullanarak bir prototip geliştirir. Esas gereksinimler anlaşıldığı zaman prototip elden çıkarılır ve kullanıcı gereksinimleri daha iyi kavranarak gerçek sistem geliştirilir.

2. **Evrimsel Prototipleme:** Evrimsel prototipleme başlangıçta minimum işlevsellik ile gerçekten çalışan prototipler geliştirmeye dayandığı için aynı zamanda 'devre tahtası prototipleme' olarak da adlandırılır. Geliştirilen prototip, tüm sistemin üzerine inşa edildiği gelecekteki prototiplerin kalbini oluşturur. Evrimsel prototipleme kullanılırken gereksinimlerin tamamının anlaşılmış olmadığı bilinerek yalnızca iyi anlaşılmış gereksinimler prototipe dahil edilir ve projeye dair gereksinimler anlaşıldıkça yazılıma eklenir.[54]

3. **Artımlı Prototipleme:** Artımlı prototip oluşturma, çeşitli alt sistemlerin birden çok işlevsel prototipini oluşturmayı ve ardından mevcut tüm prototipleri eksiksiz bir sistem oluşturmak için birbirine entegre etmeyi temel alır.

[52]John Marciniak, *Encyclopedia of Software Engineering, 2 Volume Set, 2nd Edition*, John Wiley & Sons, Inc., New York, 2001.

[53]K. Ali, 2017, s. 22.

[54]Alan Davis, "Operational Prototyping: A new Development Approach", *IEEE Software*, 1992, s. 71.

4. **Aşırı Prototipleme:** İnternet geliştirme alanlarında kullanılır. Üç sıralı aşamadan oluşur. Öncelikle var olan bütün sayfaları içeren html formatında bir prototip oluşturulur. Daha sonra bir prototip sistem katmanı kullanılarak veri işleme gerçekleştirilir. Son olarak servisler uygulanır ve son prototipe entegre edilir. Bu sürece Aşırı Prototipleme adı verilir ve bu işlemin ikinci aşamasına dikkat çekmek için kullanılır, burada gerçek servisler çok az dikkate alınarak tamamen işlevsel bir kullanıcı ara yüzünün geliştirildiği sürecin ikinci kısım ön plana çıkar.

2.4.2. Prototip Modelinin Avantajları

- Kullanıcıyı yazılım geliştirme sürecine daha fazla dâhil eder ve bu sayede ürüne dair gereksinimler daha açık bir şekilde elde edilebilir.
- Yanlış kullanıcı gereksinimlerini ve kullanıcı-geliştirici arasındaki yanlış veya eksik iletişimi en aza indirir.
- Sistemi ve özelliklerini genel olarak temsil eden ön ürünler olan prototip oluşturulur ve kullanıcıya sunulur. Bu sayede kullanıcı erkenden ürünü değerlendirmeye başlayabilir ve geribildirimde bulunabilir.
- Prototipler sayesinde kullanıcının ihtiyaç ve talepleri daha iyi belirlenebilir.
- Yazılımın başarısız olma ihtimalini azaltır zira yazılımın kalitesini arttırmak için hata tespitleri daha efektif bir şekilde yapılarak olası riskler ortadan kaldırılır.
- Sistemdeki geliştirme kapsamı nedeniyle yeni gereksinimler ürüne sorunsuz bir şekilde eklenebilir.
- Kullanıcıdan sürekli bir geri dönüş alınarak son ürüne ulaşana dek sistem üzerinde iyileştirme ve geliştirme yapıldığı için bakım maliyetini azaltır.
- Kullanıcı ve geliştirici ekip arasında devamlı bir iletişim bulunması proje boyunca iyi bir yazılım geliştirme ortamı sunar.
- Kullanıcının çalışan bir yazılım için çok beklemesi gerekmez.
- Kullanıcının geribildirimi periyodik bir şekilde alındığı için istenilen değişiklikler son dakikada ortaya çıkmaz.

2.4.3. Prototip Modelinin Dezavantajları

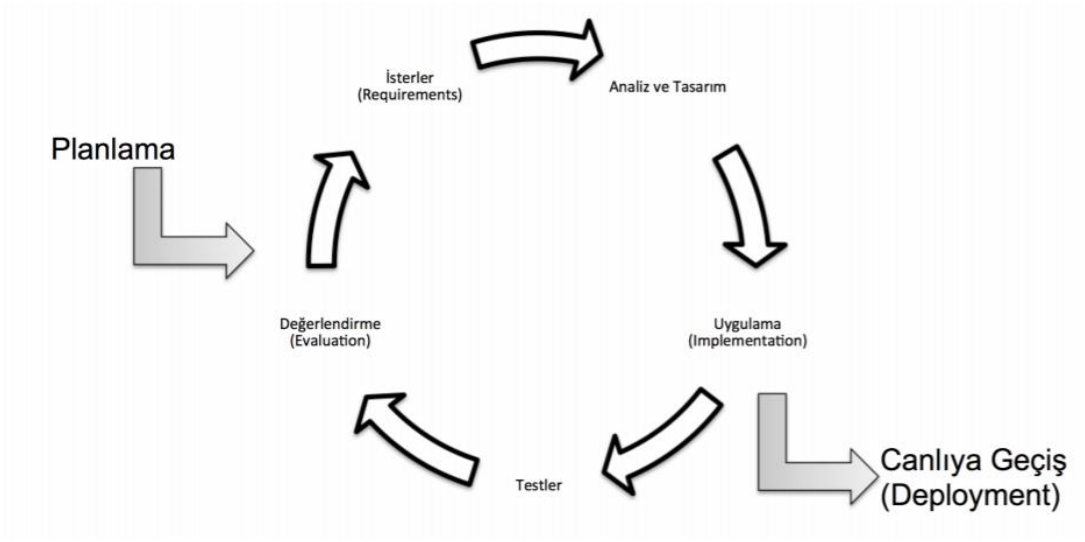
- Prototip kullanarak kullanıcıdan geri dönüş alıp üründe değişikliğe gidildiği için kullanıcının istediği ürün ortaya çıkana dek bu döngü tekrarlanır. Dolayısıyla sistemin ne zaman tamamlanıp teslim edileceği tam olarak kestirilemez.
- Yavaş bir yazılım geliştirme yaşam döngüsü modelidir.
- Kullanıcı prototipleri son ürün olarak kabul edebilir. Bu da bitmemiş bir yazılım projesinin ortaya çıkmasına sebep verir.
- Kullanıcının çok fazla işin içine girmesi ve her aşamada bulunması geliştirici tarafından tercih edilmeyebilir.
- Projede son ürüne ulaşana kadar çok fazla değişiklik olması yazılım geliştirme ekibini rahatsız edebilir.
- Geliştirici kodlama zafiyetinde bulunup birkaç değişiklikle prototipi son ürün halinde sunabilir.
- Bir ürün geliştirilirken projenin kapsamı artarsa sistem tahmin edilenden daha karmaşık hale dönüşebilir.

2.5. ARTIMLI MODEL (INCREMENTAL MODEL)

Artımlı model, tekrarlı olarak uygulanan doğrusal sıralı Şelale Modelinin öğeleri ile yinelemeli bir şekilde prototip felsefesini harmanlar.[55]Şelale Modelinin zayıflıklarının üstesinden gelmek için geliştirilmiştir. İlk planlama ile başlar ve aradaki döngüsel etkileşimlerle dağıtımla biter. Şekil 2.5.1'den görülebileceği gibi bu yöntemin arkasındaki temel fikir, tekrarlanan döngülerle (yinelemeli) ve bir seferde daha küçük bölümlerde (artımlı) bir sistem geliştirerek, yazılım geliştiricilerin sistemin önceki parçalarının veya sürümlerinin geliştirilmesi sırasında öğrenilenlerden yararlanmasına olanak tanımaktır.[56]

[55]S. S. Kute ve S. D. Thorat, 2014, s. 778-779

[56]Craig Larman, ve Basili, Victor, "Iterative And İncremental Developments. A Brief History", *Computer*, Sayı: 36, No:6, 2003, s. 49.



Şekil 2.5.1 Artımlı Model[57]

Bu model yapılması hedeflenen ürünün basit bir işlevini karşılayıp zamanla üstüne koyarak müşterinin istediği ürüne ulaşan bir yazılım oluşturacak şekilde ilerler. Yani yazılım sürecinin belli kısımlarında yazılım sürümleri üretilmekte ve üretilen her yazılım sürümü birbirini kapsayıp giderek artan sayıda işlev içerecek şekilde geliştirilmektedir. Son safhasına gelene kadar iki haftada bir kontrol edilen Bitirme Projeleri bu modele bir örnek olarak verilebilir.

Artımlı Model Prototip Modeli ve diğer evrimsel yaklaşımlar gibi yinelemeli bir yapıya sahiptir. Ancak Prototip Modelinin aksine Artımlı Model her sürümde işlevsel bir ürün ortaya çıkarmaya odaklanır.[58]İlk artımlar son ürünün sadeleştirilmiş halidir fakat hem kullanıcıya ürünün zamanla evirileceği kapasiteyi sunar hem de kullanıcının değerlendirmesi için bir platform sağlar. Artımlı geliştirme özellikle proje için belirlenen teslim tarihine kadar uygun sayıda personel bulunamadığında yararlıdır. İlk sürümler daha az kişiyle geliştirilebilir. Çekirdek ürün iyi karşılanırsa bir sonraki geliştirmeye ihtiyaç duyulan durumda daha fazla personel projeye dâhil edilebilir. Ek olarak sürümler teknik riskleri idare etmek üzere planlanabilir. Örneğin büyük bir

[57]Sadi Evren Şeker, Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü (SDLC), YBS Ansiklopedisi, 2015, s. 18 – 29, http://ybsansiklopedi.com/cilt-2-sayi-2-eylul-2015/, (13.02.2021)
[58]S. S. Kute ve S. D. Thorat, 2014, s. 778-779

sistem, geliştirilmekte olan ve teslim süresi belli olmayan yeni bir donanımın kullanımına ihtiyaç duyabilir. Yazılım geliştirme sürecinin başlarında gerçekleştirilen artım, bu donanımın kullanımından kaçınacak şekilde proje planlaması yapmaya müsaade eder, böylece kısmi işlevlerin aşırı gecikme olmaksızın son kullanıcılara teslim edilmesi sağlanır.[59]

Artımlı modelin altında hedef yazılım ürününün bir kısmını geliştirmeyi ertelemeye dair bilinçli bir karar yatar.[60] Tüm problem, herhangi bir ara ürün geliştirilmeden önce alt gruplara ayrılır. Proje süresi de artırım (veya döngü) olarak tanımlanan küçük zaman dilimlerine bölünür. Her ara geliştirmenin ürünü gerçek ortamda kullanılacak ve büyük olasılıkla herhangi bir bakım çalışması uygulanmayacaktır.[61]

Artımlı model orta ve az risk taşıyan projelerde, uzun geliştirme sürecine sahip projelerde kullanılabilir. Genel olarak yeni teknolojilerin kullanılmasına uygun bir yapısı vardır. Yani yeni teknolojiye sahip bir projede, kullanıcının büyük yeni bir ürüne atlamak yerine küçük artımlı adımlarla sisteme uyum sağlamasına imkân kılar. Bütün sistemin gereksinimleri açık bir şekilde tanımlandığında ve anlaşıldığında daha etkili bir biçimde kullanılabilir.[62]

Artımlı Modelin güçlü yanları şunlardır:[63]

- Önceki sürümlere dair geribildirim o anki sürüme dâhil edilebilir.

[59]Roger Pressman, *Software Engineering: A Practitioner's Approach 5th Edition*, McGraw-Hill, New York, 2005, s. 36.

[60]Edward Bersoff ve Davis, Alan "A New Look at the C3I Software Life Cycle," *SIGNAL*, Sayı: 41, No:8, 1987, s. 85-93.

[61]Linda Alexander, ve Davis, Alan, "Criteria for the Selection of a Software Process Model," *Fifteenth IEEE International Conference on Computer Software and Applications (COMPSAC)*, 1991, s. 521-528.

[62]Adel Alshamrani ve Bahattab, Abdullah, "A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, And Incremental/Iterative Model", *International Journal of Computer Science Issues (IJCSI)*, Sayı: 12, No:1, 2015, s. 106-107.

[63]N. Ruparelia, 2010, s. 8-13.

- Paydaşlar sürümler aracılığıyla projeye dâhil olabilir, bu da mimari riskleri daha erkenden tespit etmeye yardımcı olur.
- Her yinelemede eksiksiz bir özellik setine dönüşen erken, artımlı sürümlerle ürünün teslimini kolaylaştırır.
- Artımlı uygulama, değişikliklerin izlenmesi ve risklerin azaltılması için sorunların izole edilmesini ve çözülmesini sağlar.

2.5.1. Artımlı Modelinin Avantajları

- Şelale Modelinden daha gelişmiş bir yazılım modelidir.
- Müşteri daha fazla geribildirimde bulunabilir.
- Gereksinimlerin kesin olarak belli olmadığı projelerde kullanılabilir.
- Belge oluşturmaya yönelik bir modeldir.
- Haftalık ekiplerde iyi bir sonuç verir.
- Yazılım geliştirme yaşam döngüsünün erken safhalarında aktif olarak işleyen bir yazılım ürünü ortaya koyar.
- Gereksinim değişikliklerine adapte olma konusunda daha uygundur ve dolayısıyla daha az maliyetlidir.
- Küçük artımlı projelerde test ve debug daha kolay yapılır.
- Bu modelde kullanıcı her artıma yorum yapabilir.
- Teslim maliyeti daha azdır.
- Her artımda üzerinden geçildiği için hatalarla uğraşmak daha kolaydır.
- Oluşturulan yazılıma dair bazı önemli fonksiyonlar çabuk bir şekilde uygun hale getirilebilir.
- Döngüyle her defasında bir artım oluşturup bu artımı görüşe açarak geribildirim aldığı için sonuçlara daha çabuk ve kısa sürede ulaşılabilir.
- Gereksinim değişiklik süreci bütçeye dâhildir.

2.5.2. Artımlı Modelinin Dezavantajları

- Yüksek kalitede bir planlama ve tasarım gerektirir.
- Artımlı şekilde ürün geliştirebilmek ve parçalara ayrılabilmesi için bütün sisteme dair açık bir yorum gerekmektedir.

- Toplam maliyet şelale modelinden daha fazladır.
- Projelerin dönüm noktaları kesin değildir.
- Projenin herhangi bir safhası tam olarak bitmez.
- Gereksinim değişikliği maliyeti düşük olsa da gereksinim değişikliği için pek uygun değildir.
- Detaycı yapısından ötürü projenin daha çok yönetilmesi gerekir fakat idare edilmesi kolay değildir.
- Başta bütün gereksinimler belli olmadığında sistem tasarımı sorunları ortaya çıkar.
- Artımlar için bütün sistemin tanımlanması gerekmektedir.
- Küçük projeler için ideal değildir.
- Risk analizi ve tanımlamalar için yetkin personele ihtiyaç duyar.
- Projenin ilerleyişi risk analizine bağlıdır.[64]

[64]K. Ali, 2017, s. 15-23.

ÜÇÜNCÜ BÖLÜM

YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ: ÇEVİK (AGİLE) YAZILIM GELİŞTİRME YÖNTEMLERİ

Çevik Yazılım Geliştirme, gereksinimlerin ve çözümlerin kendi kendini organize eden, işlevlerin ekipler arasındaki işbirliği yoluyla geliştiği yinelemeli ve artımlı geliştirmeye dayalı bir yazılım geliştirme yöntemleri grubudur.[65] Çevik yöntemler, geleneksel yöntemler olarak da adlandırılan daha önceki modellerin hızla değişen ortamlarla başa çıkma konusundaki yetersizliğine, eksik kaldığı yerlere çözüm sunup alternatif görüşler getirmek üzere ortaya atılmıştır.[66]

Yinelemeli yazılım geliştirme metodu temel alınarak geliştirilen çevik yöntemde yazılım ürünü küçük artımlı yapılara bölünür ve sık aralıklarla parça parça yazılım teslimatı ile değişikliği teşvik edilir. Artımlar yinelemeli bir şekilde tamamlanır ve her yinelemede geliştirme ekibi planlama, gereksinim analizi, tasarım, kodlama, birim testi ve kabul testleri üzerine çalışır. Her yineleme sonunda üzerinde çalışılan artım kullanıcı veya müşteriye iletilir ve onlardan gelen geribildirim baz alınarak değişen gereksinimler doğrultusunda geliştirme adımları tekrarlanır. Bu adımlar müşterinin istediği ürün ortaya çıkana kadar devam ettirilir.

Çevik model, çalışan yazılım ürününün hızlı teslimi ile geliştirme süresinin uyarlanabilirliğini, yazılım geliştirme ekibi içindeki iletişimin arttırılmasını, değişen gereksinimlere uyum sağlamayı, test odaklı bir yazılım geliştirmeyi ve müşteri memnuniyetine odaklı bir yapıyı teşvik eder. Çevik yöntemler, geleneksel sistem geliştirmenin doğasında yer alan sorunları iki bilimsel kavramı kullanarak ele alır: Bir yöntem "deneysel süreç kontrolü" kullanmaktır. Bunun "yumuşak sistem düşüncesine"

[65]Harish Rohil ve Syan, Manisha, "Analysis of Agile and Traditional Approach for Software development*", International Journal of Latest Trends in Engineering and Technology*, Sayı: 1, No: 4, 2012, s. 1-3.

[66]M. Javanmard ve M. Alian, 2015, s. 1390-1391.

dayandığı da söylenebilir. Diğer yöntem ise, yazılım sistemi gelişimini "karmaşık uyarlanabilir sistemler" olarak görmek ve ekip yapıları ile yöntemleri bu olgunun etrafında tasarlamaktır.[67]

Eason (2016) bir sürü değişik çevik model olduğunu fakat hepsinin çevik manifesto tarafından tanımlanmış olan aynı vizyon ve değerleri taşıdığını belirtmiştir.[68]Bu oldukça doğal bir durumdur zira çevik manifesto; Scrum, Kanban, XP gibi yöntemleri ortaya atan sektörün ileri gelen çevik yazılım geliştiricilerinin yazılım geliştirmenin daha iyi yollarını uygulayıp başkalarına yardım ederek ortaya çıkarmak amacıyla 2001 yılında bir araya gelmesiyle ortaya çıkmıştır. Çevik Yaklaşım bireyler ve etkileşimlerin yazılım süresi ve aletlerden, ağır bir dokümantasyondansa çalışan bir yazılımın, sözleşme pazarlıklarındansa müşteri iş birliğinin, bir planı takip etmektense değişikliğe açık olmanın önemini vurgulamaktadır. Yazılım geliştiriciler "Çevik Yazılım Geliştirme Manifestosu"nu ve "Çevik Yazılımın 12 Prensibi"ni yayınlayarak çevik yöntemlerin genel olarak yazılım projelerine olan yaklaşımlarını ifade etmişlerdir.

Manifestonun temel prensipleri şunlardır:[69]

- En önemli öncelik kıymetli yazılımın erken ve sürekli teslimi ile müşteri memnuniyeti sağlamaktır.
- Geliştirmenin ileri safhalarında dahi projeye dair değişiklikler kabul edilir. Çevik yöntemler müşterinin rekabetçi üstünlüğü için değişimden yararlanır.
- Tercihen kısa zaman aralıkları ile birkaç haftadan birkaç aya, çalışan bir yazılımı düzenli olarak müşteriye sunulmalıdır.
- Proje paydaşları ve geliştiriciler proje süresince günlük olarak birlikte çalışmalıdır.
- Proje motive kişilerin etrafında inşa edilmelidir. Onlara gereken destek ve ortam verilmeli, işi bitirecekleri konusunda güven duyulmalıdır.

[67]A.g.m. s. 1390.

[68]Atiku Victor Usman ve Nonyelum, Ogwueleka Francisca, "SDLC Models as Tools in the Development of MIS: A Study", *IUP Journal of Information Technology*, Sayı: 14, No: 4, 2018, s. 52-59.

[69]Principles behind the Agile Manifesto, https://agilemanifesto.org/principles.html, (27.01.2021)

- Bir proje ekibinin ve ekip içindeki bilgi alışverişinin en etkili ve verimli yolu yüz yüze iletişimdir.
- Çalışan yazılım, projenin ilerleyişinin ana ölçütüdür.
- Çevik yöntemler, sürdürülebilir geliştirmeyi destekler. Proje sponsorları, geliştiricileri ve kullanıcıları süresiz olarak sabit bir tempoyu devam ettirebilmelidir.
- Teknik mükemmellik ve iyi tasarıma sürekli olarak dikkat etmek çevikliği arttırır.
- Basitlik, yapılmayan işin miktarını arttırma sanatı, gereklidir.
- En iyi mimari, gereksinimler ve tasarımlar kendisini organize edebilen ekiplerden çıkar. Düzenli aralıklarda proje ekibi nasıl daha etkili olabileceği üzerine düşünür ve daha sonra davranışlarını buna göre ayarlayıp düzenler.

Çevik Yöntem zaman içerisinde sektörde kendine daha çok yer bulmaya başlamıştır. Gerek dünya çapında olsun gerek ülkemiz baz alınarak olsun çevik yöntemlerin kullanımı hızla yaygınlaşmaktadır. AgileTurkey tarafından yayınlanan 2019 Türkiye Çeviklik Raporuna göre incelenen organizasyonların %94'ü çevik yöntemleri kullanmaktadır.[70]

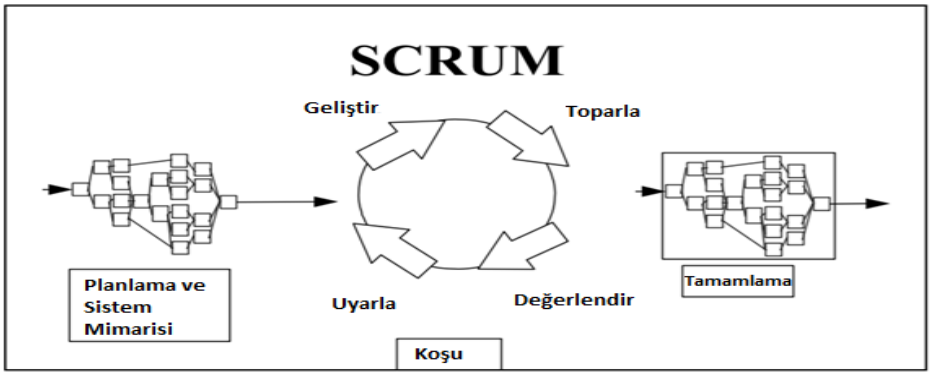
Çevik geliştirme yöntemleri benzer hususları temel alarak kendilerine özgü yazılım geliştirme yöntemleri ortaya atmışlardır. Her bir yöntem bir diğerine göre temel olarak benzer fakat ilerleyiş ve işleyiş olarak farklı bir bakış açısı içermektedir. Başlıca Çevik Yazılım Geliştirme Yöntemleri şunlardır;

- Scrum
- Ekstrem Programlama (Extreme Programming)
- Kanban
- Kristal Yöntemler
- Dinamik Sistem Geliştirme Yöntemi

[70]8th Annual Agility Report, http://www.agileturkey.org/ (24.02.2021)

3.1. SCRUM

Scrum 1990'lı yıllarda Jeff Sutherland ve Ken Schwaber'in çalışmaları sonucu ortaya çıkan bir çerçevedir. Scrum insanların, ekiplerin ve kuruluşların karmaşık problemler için çözüm üretmelerine imkan tanıyan bir yapı benimsemiştir.[71] Tekrar eden ve yinelemeli bir yöntem olması artımlı projeleri yönetmek ve kontrol etmek için geniş uygulanabilirliğe sahip olmasını sağlamaktadır. Ekip tabanlı bir şekilde müşteri ihtiyaçlarına hızlı bir şekilde adapte olup cevap verebilecek şekilde tasarlanmıştır. Scrum, basitliği ve kanıtlanmış üretkenliği sebebiyle yazılım camiasında giderek daha popüler hale gelmektedir. Genel olarak gözleme dayalı bir yaklaşımı vardır. Yazılım geliştirmeyi koşu (sprint) adını verilen kısa artımlar içerisinde gerçekleştirir. Her koşuda tasarım, test, müşteri görüşü gibi yazılım geliştirme yaşam döngüsü modellerinin aşamaları bulunur.[72] Scrum temelde sorunun tam olarak anlaşılamayacağını benimser ve geleneksel yöntemlerin aksine koşu içerisinde gerçekleştirilen analiz, tasarım ve geliştirme gibi safhaların öngörülemez olduğunu savunur. Riski ve öngörülemezliği yönetmek için bir kontrol mekanizması kullanılır. Şekil 3.1.1'de yapısı görülebilen Scrum için esneklik, çabuk cevap verme ve güvenirlik ön plandadır.[73]



Şekil 3.1.1 Scrum Yöntemi

[71]The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game, https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf, (27.02.2021)

[72]Gurpeet Singh Matharu, Mishra, Anju, Singh, Harmet ve Upadhyay, Priyanka, "Empirical Study Of Agile Software Development Methodologies: A Comparative Analysis", *ACM SIGSOFT Software Engineering Notes*, Sayı: 40, No: 1, 2015, s. 2-3.

[73]Ken Schwaber, *Scrum Development Process*, Springer, London, 1997, s.10.

Scrum Yönteminin yapısı Schwaber'e göre şu şekildedir:[74]

- İlk ve son aşamalar (Planlama ve Kapatma) tüm süreçlerin, girdilerin ve çıktıların iyi tanımlandığı tanımlanmış süreçlerden oluşur. Bu süreçlerin nasıl yapılacağı açık bir şekilde belirtilmiştir. Akış, planlama aşamasındaki bazı yinelemeler ile doğrusal ilerler.

- Koşu(Sprint) aşaması deneysel bir süreçtir. Koşu aşamasındaki süreçlerin çoğu tanımlanmamış ya da kontrol edilemezdir. Dışarıdan kontrol edilmesi gerekilen bir kara kutu gibi ele alınır. Bu sebeple kaosu engelleyip esnekliği en üst düzeye çıkarabilmek için risk yönetimi dâhil kontroller koşu aşamasının her bir yinelemesine yerleştirilir.

- Koşular doğrusal değildir ve yapısal olarak esnektir. Mümkün olan yerlerde açık süreç bilgisi kullanılır; aksi takdirde, süreç bilgisini oluşturmak için zımni bilgi ve deneme yanılma yöntemleri ile ilerlenir. Koşular son ürünü elde etmek için kullanılır.

- Proje Kapatma aşamasına kadar proje ekibine olduğu gibi kullanıcılara da yani çevreye açıktır. O aşamaya kadar elde edilen çıktı Planlama ve Koşu aşamaları sırasında istenildiği zaman değişikliğe uğrayabilir. Bu süreçler boyunca proje rekabet, zaman, kalite, mali baskı gibi çevresel karmaşıklıklara açık kalır.

- Çıktı, çevreye göre çevreden gelen geri bildirimlere göre proje sırasında belirlenir.

Scrum yazılım geliştirme yöntemi önceki yazılım geliştirme yöntemlerine kıyasla çevrenin etkisine daha açıktır. Proje süresince açık süreç bilgisi kullanıldığı için şeffaftır ve bu sayede çevre daha aktif bir şekilde projede yer alabilir. Yazılım geliştiriciler arasındaki iletişim ön planda tutulur ve böylece bilgi alışverişi daha üst düzeyde yapılır. Schwaber'in Scrum yapısında belirttiği üzere Scrum deneysel bir felsefe benimsemiştir ve koşular içerisinde gerçekleşen tasarım, analiz, kodlama gibi aşamaların keskin tarifleri yoktur, bundan ziyade Scrum aşamalara belli bir çerçeve

[74] K. Schwaber, 1997, s. 10.

çizerek yazılım mühendislerine ve işletmelere kendi yöntemlerini uygulama imkânı sunar.

Deneysel süreç kontrol teorisine dayanan Scrum yönteminin başarılı bir şekilde işleyebilmesi için önemli olan üç faktör vardır.[75] Bu faktörlerin proje geliştirme aşamalarının farklı safhalarında uygulanması gerekmektedir. Bahsi geçen faktörler şunlardır:

1. **Şeffaflık:** Yazılım geliştirme süreci projeye dahil olan herkese açık olmalıdır. Şeffaflığı düşük olan yapılar, değeri azaltan ve riski artıran kararlara yol açabilir bu da projeyi yarım kalmaya veya daha ileri giderek başarısızlığa götürebilir.

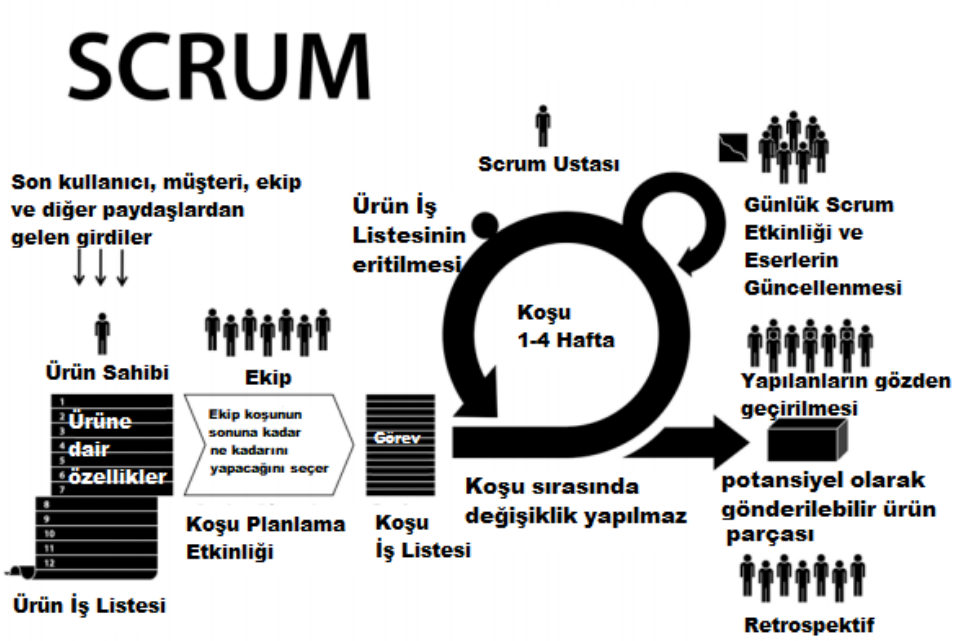
2. **Gözlem:** Scrum kullanıcıları problemleri erken safhalarda tespit edebilmek için sık sık çıktıları kontrol etmelidir.

3. **Uyum:** Şayet çıktıyı inceleyen biri projenin bazı yönlerinin kabul edilemez olduğunu ve proje kapsamının dışında olduğuna karar verirse, ileride olabilecek hataları önlemek için süreç yeniden ayarlanabilir.

3.1.1. Scrum İçerikleri

Scrum yapısı Scrum ekipleri, olayları, eserleri ve kurallarından oluşur. Kurallar proje sırasında ekipleri, olayları ve eserleri birbirine bağlamak için elzemdir. Ayrıca proje içinde meydana gelebilecek çatışmaları çözmek için onaylanabilir bir yapı ortaya koyar. Şekil 3.1.1.1'de Scrum yapısı görülebilir.

[75]Howard Lei, Ganjeizadeh, Farnaz, Jayachandran, Pradeep Kumar ve Özcan, Pınar, "A Statistical Analysis Of The Effects Of Scrum And Kanban On Software Development Projects", *Robotics and Computer-Integrated Manufacturing*, Sayı: 43, 2017, s. 60.



Şekil 3.1.1.1'de Scrum[76]

3.1.1.1. Scrum Ekibi

Scrum'un temel birimi az sayıda kişiden oluşturulmuş bir Scrum Ekibi'dir. İçerisinde hiyerarşi ya da alt-grup yer almayan ekip bir Scrum Ustası, Ürün Sahibi ve Geliştirme Ekibi'nden oluşur. Her seferinde tek bir hedefe odaklı çalışan, kaynaşmış profesyonellerden oluşan bir ekiptir. Ekip üyelerinin her biri her koşuda gereken değeri yaratabilecek kadar yeterlidir. Üyeler kendi kendilerini yönetir yani neyi ne zaman nasıl yapacaklarına kendileri karar verirler. Çeviklik ön planda olduğu için koşulara hızlı yanıt verebilecek şekilde ekip içindeki üye sayısı genelde on ya da daha az bir sayı şeklinde belirlenir. Bu durumda küçük ekiplerin birbiriyle daha iyi iletişimde bulunması ve dolayısıyla süreci daha iyi işleterek üretken olmaları etken olmuştur.

1. **Ürün Sahibi:** Genelde tipik olarak, projeyi etkinleştirmek için neyin inşa edilmesi gerektiğini ve yapı sırasının nasıl ilerlemesi gerektiğini bilen işlevsel bir birim

[76] Jeff Sutherland, *Jeff Sutherland's Scrum Handbook*, Scrum Training Institute, 2010, s. 11.

yöneticisidir.[77]Ürün sahibi, Ürün İş Listesini, ürünün gereksinim listesini yönetmekten ve projenin değerini maksimize etmekten sorumludur. Rolleri arasında ayrıca Ürün İş Listesi maddelerini ve projenin hedeflerini Geliştirme Ekibine açıklamak, ekibin bu hedefleri anlamasını ve üst düzeyde performans göstermesini sağlamak bulunmaktadır.

Ürün Sahibinin özellikleri şunlardır[78]:

- Müşterinin temsilcisidir.
- Ekiple birlikte çalışarak, Geliştirme ekibi ile müşteri arasında bir köprü oluşturup iletişim sağlar.
- Ürün İş Listesini yürütür ve ürün gereksinimlerine öncelik verir.
- Ürüne dair olması istenen özellikleri tanımlar.
- Ürünün içeriğine ve yayınlanma tarihine karar verir.
- Kârdan ve ürüne verilecek son onaydan sorumludur.
- Gerektiği zaman ürünün özelliklerini değiştirebilir.
- Ürün çıktılarını kabul ya da reddetme yetkisine sahiptir.

2. **Geliştirme Ekibi:** Ürün sahibi tarafından kararlaştırılan koşu bitim süresinde belirlenen görevleri yerine getirmekle yükümlüdür. Her koşunun sonunda ortaya yinelenebilir bir ürün çıkarmaları gerekir. Geliştirme ekibi son ürünün uygulanmasını kontrol eder. Ekip üyelerinin veri tabanı yönetimi, ara yüz tasarımı gibi hususlarda yetenek yelpazesinin geniş olması önemli bir konudur. Kendi işlerini kendileri yönetirler, kendi kendilerine organize olurlar, ekip içerisinde unvan yoktur ve aralarında alt ekipler kurmazlar. Ekip içerisindeki kişi sayısı önemli bir konudur zira küçük bir ekip beceri eksikliğinden muzdarip olabilirken büyük bir ekip geliştirme karmaşıklığından muzdarip olabilir. Her projede elzem sayıda ve yetkin kişilerden oluşan bir geliştirme ekibinin oluşturulması projeyi başarıya götürecek yapıtaşlarından biridir.

[77]Frank Cervone, "Understanding Agile Project Management Methods Using Scrum", *OCLC Systems & Services: International digital library perspectives*, Sayı: 27, 2011, s. 20.

[78]M. Mahalakshmi ve Sundararajan, Mukund "Traditional SDLC Vs Scrum Methodology–A Comparative Study", *International Journal of Emerging Technology and Advanced Engineering*, Sayı:3, No:6, 2013, s. 193.

The Scrum Guide'a göre geliştirme ekibinin genel özellikleri şunlardır:[79]

- İhtiyaç duyulan ürünü geliştirir.
- Her ekip üyesi kendi işini gerçekleştirmekle yükümlüdür.
- Her koşuda ellerinden gelen en iyi şekilde katkıda bulunmalılardır.
- Ürünün başarılı ya da başarısız olmasından sorumludurlar.
- Genelde on veya daha az üyeden oluşurlar.
- Ekip üyeleri test uzmanı, programcı, tasarımcı vb. kişilerden oluşabilir.
- Takım çalışmasına uygun, iletişime açık, sorumluluk sahibi olmaları beklenir.
- Koşu ve Koşu İş Planı için bir plan oluşturmaları beklenir.

3. **Scrum Ustası:** Ürün İş Planı'nı yönetir ve Geliştirme Ekibinin açık Ürün İş Planı çıktıları oluşturmasını sağlar. Scrum Ustası ekibin projeye dair uzun soluklu planları anlamasını sağlayan kişidir. Scrum'un etkili bir şekilde gerçekleşmesinden sorumludur.

Barry Overeem'e göre Scrum Ustası şu şekilde hareket eder:[80]

- Geliştirme ekibi üyelerinin ve çalıştıkları kişilerin ihtiyaçlarına önem vererek ürünün istenilen şekilde ortaya çıkmasını sağlayan **hizmetkar lider**
- Ekip içerisinde gereken konularda yardımcı olup yol göstererek Scrum'un gerçekleştirilmesini sağlayan **koç**
- Scrum ve diğer ilgili yöntemlerin anlaşılıp uygulandığına emin olan **öğretmen**
- Çevik bilgisini ve tecrübesini ekibe aktaran bir **usta**
- Süreçleri yönetip ekibin sağlığıyla ilgilenen, olası problemlerle ilgilenen ve ekibin sınırlarını idare eden **yönetici**

[79] The Scrum Guide, 2020, s. 5.

[80] 8 Stances Of Scrum Master, https://www.scrum.org/resources/8-stances-scrum-master, s. 2-3, (01.03.2021)

- Geliştirme ekibinin kendi organize olmasını göz önünde bulundurarak Ekibin ilerlemesinin önünde duran engelleri çözen **engel kaldırıcı**
- Ekibin iş birliği yapması için ortamı ayarlayıp açık sınırlar çizen bir **iş kolaylayıcı**
- Scrum ekibinin verimli bir şekilde çalışması için çevreyi ayarlayan **değişim ajanı**

Scrum Ustasının genel olarak özellikleri şunlardır:

- Ekip üyelerine yardımcı olup meydana gelen engelleri ortadan kaldırarak ürünün geliştirilmesinin sürdürülmesini sağlar.
- Ekip üyeleri arasında lider vasfını yüklenmiştir.
- Süreci yönetmekle sorumludur.
- Günlük scrum, koşu ve planlama toplantılarını yönetmekle yükümlüdür.
- Gerektiği halde dokümantasyonla ilgilenir.
- Ekibi dış müdahalelerden korur.

3.1.1.2. Scrum Etkinlikleri

Scrum, proje geliştirme ve proje planlama aşamalarında daha verimli ve etkili sonuçlar elde edilebilmek için en fazla otuz gün süren belirli zaman aralıkları olan zaman kutusu olaylarını kullanır.[81]Scrum'daki etkinlikler, Scrum eserlerini incelemek ve projenin problemlerine çözüm bulmak adına yeni yöntemler bulmak ve projeye uyarlamak için tasarlanmıştır. Bu etkinliklerin amacı Scrum'un ana faktörleri olan geliştirme sürecinde şeffaflık, uyum ve gözlemi sağlamaktır. Etkinliklerin düzenli ve düzgün bir şekilde ilerlememesi projenin seyrinde aksamalara yol açabilir çünkü etkinlikler Scrum'da tanımlı olmayan toplantı ihtiyacını en aza indirmek ve bir süreklilik sağlamaktır. İdeal olarak bütün etkinlikler karmaşıklığı azaltmak adına aynı anda aynı yerde yapılır.

[81]C. Larman ve V. Basili, 2003, s. 47-56.

1. **Koşu Etkinliği:** Scrum'un kalbi olan Koşu, kullanılabilir bitmiş bir ürün oluşturmak için kullanılan bir zaman kutusu olayıdır. Her koşu ne yapılması ve nasıl yapılması gerektiği planlanan bir aylık bir proje olarak düşünülebilir. Genel olarak tutarlılık oluşturmak adına koşu süreleri kısa tutulur çünkü süreç uzadıkça proje karmaşıklığının artma olasılığı vardır. Her koşunun geliştirme hedefleri koşu sırasında değişiklik göstermemelidir fakat koşu sonunda Ürün Sahibi ve Geliştirme Ekibi proje kapsamını diledikleri gibi yeniden tanımlayabilirler. Ürün Sahibi Kurumunun direktifleri, piyasa gereksinimleri ya da teknoloji doğrultusunda Koşuları iptal edebilir.

2. **Koşu Planlama Etkinliği:** Etkinlik Ürün Sahibi, Scrum Ustası ve Geliştirme ekibi ile gerçekleştirilir. Her koşu iki safhada tamamlanan koşu planlama ile başlar. İlk safhada Ürün Sahibi ve Scrum Ustası çok önemli olan Ürün İş Listesini oluşturup görevleri gözden geçirir. Geliştirme ekibinin ürün ihtiyaçlarını açıkça anlamasını sağlamaya yardımcı olmak için yüksek önem taşıyan görevlerin amaçlarına ve bağlamına karar verirler. Etkinliğin bu kısmı genel olarak ürüne odaklanır. İkinci fazda ise ekip elindeki işi koşu sonuna kadar nasıl yapacağına odaklanır. Ekip, Ürün Sahibinin kararına bakmaksızın görevin tamamlanma olasılığını inceler. Daha sonra ekip, işi belirlenen sürede tamamlamayı taahhüt eder.[82] Genel olarak koşu içerisinde hangi işin ne şekilde yapılması gerektiğine karar verilir.

3. **Günlük Scrum Etkinliği:** Geliştirme ekibi günde 10 ya da 15 dakika süreliğine toplanarak ürüne dair ne aşamada olduklarını tartışır. Sürecin ilerlemesine mâni olan herhangi bir durum tespit edilirse bu duruma göre pozisyon alırlar. Bu etkinliklerde her ekip üyesi şu soruları cevaplamaya çalışır:[83]

   • Son ekip toplantısından beri ne yaptın?

   • Hangi sorunlarla karşılaşıyorsun?

   •Bir sonraki ekip toplantısına dek elindeki göreve neyi yapmayı hedefledin?

[82]Faiza Anwer, Aftab, Shabib, Muhammad Shah, Syed ve Waheed, Usman "Comparative Analysis Of Two Popular Agile Process Models: Extreme Programming And Scrum", *International Journal of Computer Science and Telecommunications*, Sayı:8 No:2, 2017, s. 5.

[83]Roger Pressman ve Bruce Maxim, Software Engineering: A Practitioner's Approach 8th Ed, *McGraw-Hill Education,* New York, 2014, s. 79.

İlk ve üçüncü sorular ekip üyelerinin elindeki görevin bulunduğu noktayı, ilerlemelerini ve hedeflerini ölçer. Koşu içerisinde işi bitirmek gerektiği için projenin herhangi bir şekilde aksamaması ve herkesin üzerine düşeni yapması oldukça önemlidir. İkinci soruyla proje sürecinin bitirilmesini geciktirecek ya da engel olabilecek hususları tartışmak ve varsa sorunlara çözüm bulmak amaçlanır.

4. **Koşu Değerlendirme Etkinliği:** Scrum'un ana fikri gözlem ve uyumdur. Ne olduğunu görüp öğrenmek ve daha sonra geribildirime dayanarak ilerlemek ve bunu bir döngü şeklinde tekrarlamaktır. Koşu değerlendirme etkinliği ürün için uyum ve gözlem aktivitesidir.[84]Her koşu sonunda geliştirme ekibi üzerine düşen görevleri yapmış olmalı ve bir ürün geliştirilmelidir. Koşu sonu geliştirilen ürünün bir değerlendirilmesi yapılır. Koşu değerlendirme etkinliğinin amacı koşuların çıktılarını gözden geçirmek ve gelecek adaptasyonları belirlemektir. Koşunun çıktısı Ürün Sahibi aracılığıyla proje paydaşlarına sunulur ve paydaşlar üründe neler yapıldığını ortamda nelerin değiştiğini inceleyip geliştirmenin ihtiyaçlara göre gidip gitmediğine karar verirler. Ürün Sahibi, Scrum Ustası ve Ekip üyeleri arasında gerçekleşen ayrıntılı görüşme, ürüne dair geliştirme yönlerini değiştirebilecek geri bildirimler alınmasına yardımcı olur. Ürün İş Listesi alınan yorumlara ve isteklere göre yeniden şekillenebilir.

5. **Koşu Retrospektif Etkinliği:** Koşu Değerlendirme Etkinliğinde ürün için uyum ve gözlem yapılırken ardından gelen Koşu Retrospektif Etkinliğinde proje sürecinin uyum ve gözlem durumu ele alınır. Geliştirme ekibi, müşteriler, Ürün Sahibi, Scrum Ustası gibi bütün proje paydaşlarının katıldığı bu görüşmede deneyimler ve karşılaşılan sorunlar tartışır. Projede daha iyi ilerlemek için ne yapılıp ne yapılmaması gerektiği konuşulur, bir sonraki koşuda nasıl yaklaşımlarda bulunulması gerektiği tartışılır. Bu etkinlik sürecin gelişmesi için oldukça yararlıdır zira kimin çalışıp çalışmadığı, hangi işin yapıldığı ve nelere yönelmek gerektiği anlaşılır. [85]

[84]J. Sutherland, 2010, s. 27-29

[85] Jeff Sutherland ve Schwaber, Ken, "The Scrum Papers Nuts, Bolts and Origins of an Agile Process", 2007, s. 27.

3.1.1.3. Scrum Eserleri

Scrum Eserleri yapılan işi ve üretilen değeri gösteren dokümanlardır. Herkesin anlayabileceği şekilde tasarlanan her eserde şeffaflığı artıran bilgiler sağlandığından emin olmak için çaba gösterilir ve bu eserler hangi ilerlemenin ölçülebileceğine odaklanmayı sağlar:

- Ürün İş Listesi için Ürün Hedefi,
- Koşu İş Listesi için Koşu Hedefi
- Koşu Kalan İş-Zaman Grafiği için Koşu-Süre Hedefi
- Ürün Parçası için Tamamlandı Tanımı

Bu eserler deneyciliği ve Scrum Takımı ile paydaşlara Scrum değerlerini pekiştirmek için mevcuttur. [86]

1. **Ürün İş Listesi:** Ürünü geliştirmek için gerekenleri yani ürüne dair gereksinimler, fonksiyonlar, geliştirmeler ve gereken düzenlemeleri içeren listedir. Şekil 3.1.1.3.1'de bir örneği görülebilir. Ürünün teknik ve işlevsel yanlarını açık bir şekilde gösterir. Bu listenin oluşturulmasından ve ekibe açıklanmasından Ürün Sahibi sorumludur. Ürün İş Listesi dinamik bir yapıya sahiptir, projede bir ilerleme yapıldığı zaman gelişip değişebilir. Ürün İş Listesi, ürünün gelecekteki bir durumunu tanımlayan Ürün Hedefi'ne hizmet eder. Ürün Hedefi ekibin ulaşmayı amaçladığı son ürünü ifade eder.

[86] The Scrum Guide, 2020, s. 10.

					Koşu İtibariyle Geriye Kalan Yeni Çaba Tahminleri		
Ürün	**Detaylar**	**Öncelik**	**Değer Tahmini**	**Başlangıç Çaba Tahmini**	**1**	**2**	**3**
Kullanıcı olarak sepete ürün ekleme	....	1	7	5			
Kullanıcı olarak sepetten ürün çıkarma	....	2	6	2			
Bütün sunucuları Apache 2.4.48'e yükseltme	....	4	6	20			

Şekil 3.1.1.3.1 Ürün İş Listesi Örneği[87]

2. **Koşu İş Listesi:** Belirli bir koşu için seçilen Ürün İş Listesi kalemlerini içeren listedir. Geliştirme Ekibi bir sonraki koşuda yapılacak ürünün fonksiyonlarını ve yapılması gereken işi bu listede belirtir. Koşu sırasında Geliştirme Ekibi daha fazla çalışma gerektiğini anlarsa o işi Koşu İş Listesine ekler. Koşu İş Listesindeki kalan işler, koşunun ilerlemesini yönetmek için ekip tarafından izlenebilir. Bu listede Ürün İş Listesindeki maddeler ele alınır ve Koşu Hedefi'ne ulaşarak ürünün belli bir kısmını bitirmek amaçlanır.

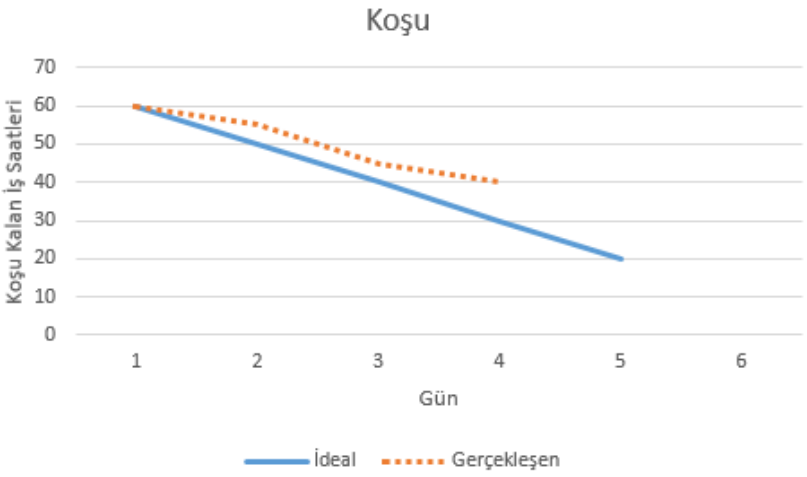
			Gün itibariyle geriye kalan koşu tahminleri						
Ürün İş Maddesi	**Koşu görevi**	**Görevi Yapacak Kişi**	**Koşu Çaba Başlangıç Tahmini**	**1**	**2**	**3**	**4**	**5**	**6**
Kullanıcı olarak sepete ürün koymak istiyorum	veritabanını düzenle		5						
	kullanıcı yardım sayfasını güncelle		3						
	web sayfası oluştur (javascript logic)		13						
	Otomatik kabul testleri yaz		13						
İşlemi gerçekleştirme performansını iyileştirme	DCP kodunu birleştir		5						
	pRank için makine sırasını tamamla		8						

Şekil 3.1.1.3.2 Koşu İş Listesi Örneği[88]

3. **Koşu Kalan İş-Zaman Grafiği:** Bir koşu içerisinde kalan Koşu İş Listesi maddelerini süreyle gösteren bir grafiktir. Günlük olarak güncellenir. Ekibin

[87] J. Sutherland, 2010, s. 18.
[88] J. Sutherland, 2010, s. 22.

ürünle ilgili hangi noktada olduklarını anlamaya yarayan bir araçtır. Bu sayede koşu içerisindeki zaman yönetimi, kalan işlerin yetişip yetişmeyeceği gözlemlenebilir. Ekip grafiği kullanırken kendisi organize olur.



Şekil 3.1.1.3.3 Koşu Kalan İş-Zaman Grafiği Örneği

4. **Ürün Parçası:** Üretim ortamında yayımlanmaya hazır olan küçük bir ürün birimini ifade eder. Scrum ekibi her koşu sonucunda müşteri için anlamlı, çalışan bir yazılım parçası ya da bir ürün ortaya çıkarmayı hedefler ve buna Ürün Parçası denir. Her koşu sonucu ortaya çıkan ürün parçaları birbirine eklenir. Ürün parçası koşu değerlendirme toplantısında paydaşların değerlendirilmesine sunulur ancak 'tamamlandı' olarak tanımlanmadığı sürece tam ürün olarak kabul edilmemelidir. Tamamlandı ifadesi geliştirilen ürün parçasının beklentileri karşılaması sonucu verilen resmi tanımdır.

Scrum genel olarak takım çalışmasına, müşteriyle olan irtibata önem veren, Scrum ekibi içerisinde özgürlük ve oto-kontrol imkânı sunarak karmaşık ürün ve projelerin gerçekleştirilmesi ve sürdürülmesi için kullanılan çevik bir yöntemdir.

3.1.2. Scrum Avantajları

- Çevik Scrum hızlı ve anlaşılabilir yapısı sayesinde kurumların zamandan ve paradan tasarruf etmesine yardımcı olur.
- Scrum yöntemi iş gereksinim belgelerinin ölçülmesinin zor olduğu projelerin başarıyla geliştirilmesini sağlar.
- Ürün geliştirme sürecinde hızlı bir şekilde kodlama yapılıp ürün test edilebilir ve olası hatalar kolayca tespit edilip giderilebilir.
- Sık sık yapılan toplantılar ile projenin ilerleyişini kontrol eden bir yapı benimsemektedir. Bu da proje geliştirme sürecinin şeffaf bir şekilde ilerlemesini sağlar. Paydaşlar ve ekip projede olan biteni açıkça görebilirler.
- Diğer çevik geliştirme yöntemleri gibi yinelemeli bir doğası vardır. Kullanıcıdan sürekli olarak geri bildirim beklenir. Dolayısıyla kullanıcı ile sürekli iletişim halinde bir geliştirme süreci geçirilir. Bu da ortaya kullanıcının istekleri doğrultusunda bir ürün çıkmasına yardımcı olur.
- Kısa koşular ve paydaşlardan sık sık alınan geribildirim sayesinde, ürüne dair meydana çıkan değişikliklere uyum sağlamak daha kolaydır.
- Günlük toplantılar yazılım geliştiricilerin kişisel verimliliğini ölçmeyi mümkün kılar. Bu da yazılım ekibinin her üyesinin verimliliğini arttırmayı sağlar.
- Günlük toplantılar sayesinde karşılaşılan sorunlar detaylıca tanımlanır, tartışılır ve hızlı bir şekilde çözülebilir.
- Belirli bir zaman çerçevesi içerisinde kaliteli bir ürün ortaya çıkarmak daha kolaydır.
- Scrum herhangi bir teknoloji ya da programlama diliyle iyi çalışabilir fakat özellikle hızla ilerleyen internet teknolojileri ve medya projeleri için yararlıdır.
- Süreç ve yönetim açısından genel maliyet minimum tutulduğu için daha hızlı bir şekilde sonuca götürür.
- Ürünün geliştirme süresi koşulara bölünerek gerçekleştirildiği ve her koşuda ürün iş listesi maddelerinden oluşan koşu iş listesi tamamlanmaya çalışıldığı için koşuların ilerlemesi ölçülerek projenin bitiş süresi daha kolay belirlenebilir.

3.1.3. Scrum Dezavantajları

- Çevik Scrum önde gelen yazılım geliştirme modellerinden biridir çünkü projelere dair kesin bir bitiş tarihi öngöremez ve bu durum, proje paydaşlarının projeyle ilgili sürekli olarak yeni özellik taleplerinde bulunmasını mümkün kılar.
- Dokümantasyon çok azdır.
- Bir görev iyi tanımlanmadığı takdirde projenin süresi ve maliyeti tam olarak belirlenemez. Böyle bir durumda, görev çeşitli koşulara yayılabilir ve hem maliyet artışı hem de zaman israfı meydana gelir.
- Scrum'da projeye bağlı olmak önemlidir. Yazılım ekibi üyeleri kendilerini projeye adamadığı takdirde proje ya asla tam olarak bitmez ya da başarısız olur.
- Küçük çaplı ekiplerle daha iyi sonuç alındığı için küçük, hızlı ilerleyen projeler için daha uygundur.
- Bu yöntem tecrübeli ekip üyelerine ihtiyaç duyar. Kullanılan teknolojiyle yeni tanışan üyelerden oluşan bir ekip kullanıldığı takdirde proje zamanında tamamlanamayabilir.
- Scrum, Scrum Ustası yönettiği ekip ile iyi çalışırsa başarılı olur. Eğer ekip üyeleri üzerinde çok sıkı bir kontrol kurarsa bu durum ekip içerisinde moral bozukluğuna, isteksizliğe yol açabilir ve dolayısıyla projeyi başarısızlığa iter.
- Koşu içerisinde her ekip üyesi kendi işinden sorumludur. Proje sırasında takım üyelerinden biri projeden ayrılırsa bunun projeye negatif bir etkisi olabilir.
- Test ekibi her koşudan sonra hata testi yapmazsa proje kalite yönetimini uygulamak ve ölçmek zor olur.
- Ekip üyeleri arasındaki uyum çok önemlidir. Eğer üyeler birbiriyle iyi bir şekilde iş birliği yapmaz ve iletişim kuramazsa bu durum projeyi başarısızlığa sürükleyebilir.

3.2. EKSTREM PROGRAMLAMA (EXTREME PROGRAMMING)

Ekstrem Programlama 1996 yılında bir proje üzerinde çalışırken Kent Beck tarafından ortaya atılan bir çevik yazılım geliştirme yöntemidir. Belirsiz veya hızla değişen gereksinimleri yönetme becerisine sahip, hafif, daha esnek ve düşük riskli, disiplinli bir yazılım geliştirme yaklaşımı olarak kabul edilir.[89]Küçük ve orta ölçekli ekipler için daha uygundur.[90]

Disiplinli bir şekilde uygulanan değerler, ilkeler ve uygulamalar koleksiyonudur ve Ekstrem Programlamanın 'ekstrem' olmasının sebebi yüksek kalitede bir yazılım elde etmek için bu uygulamaları aşırıya götürmesindir.[91] Bu yöntem kullanıcı memnuniyetini oldukça ön planda tutar ve yazılım geliştirme sürecinde müşterilerle yoğun etkileşimlerde bulunmasıyla bilinir. Müşterilerden devamlı ve hızlı bir şekilde geribildirim alır ve sık aralıklarla ortaya bir ara ürün çıkarır. Bu şekilde de hem müşterilerin katkısı hem de ara ürünlerin incelenmesiyle olası sorunlar daha rahat bir şekilde tespit edilip giderilebilir. Dolayısıyla hata oranı azalır ve bu da geliştirme maliyetini düşürerek daha uygun bir fiyata daha kabul edilebilir bir son ürün ortaya çıkmasını mümkün kılar.

Ekstrem Programlama günümüzde en yaygın kullanılan çevik yazılım geliştirme yöntemlerinden biridir. Çevik Manifesto'nun yazılım geliştirme ile ilgili belirlediği sınırlara uyar fakat daha ileri giderek kolay bir uygulama serisi çizer. Birçok popüler yöntembilimi 'Bir yazılım projesinde ihtiyaç olabilecek uygulamalar neler?' sorusunu cevaplamaya çalışırken, Esktrem Programlama, 'Yazılım projesinde ihtiyacım olabilecek en basit uygulamalar hangileri ve ihtiyaçlarımı bu uygulamalara göre sınırlamak için ne yapmalıyım?' sorusunu sorar.[92]

---

[89]James Newkirk, "Introduction To Agile Processes And Extreme Programming", *Proceedings of the 24th International Conference on Software Engineering*, 2002, s. 695-696.

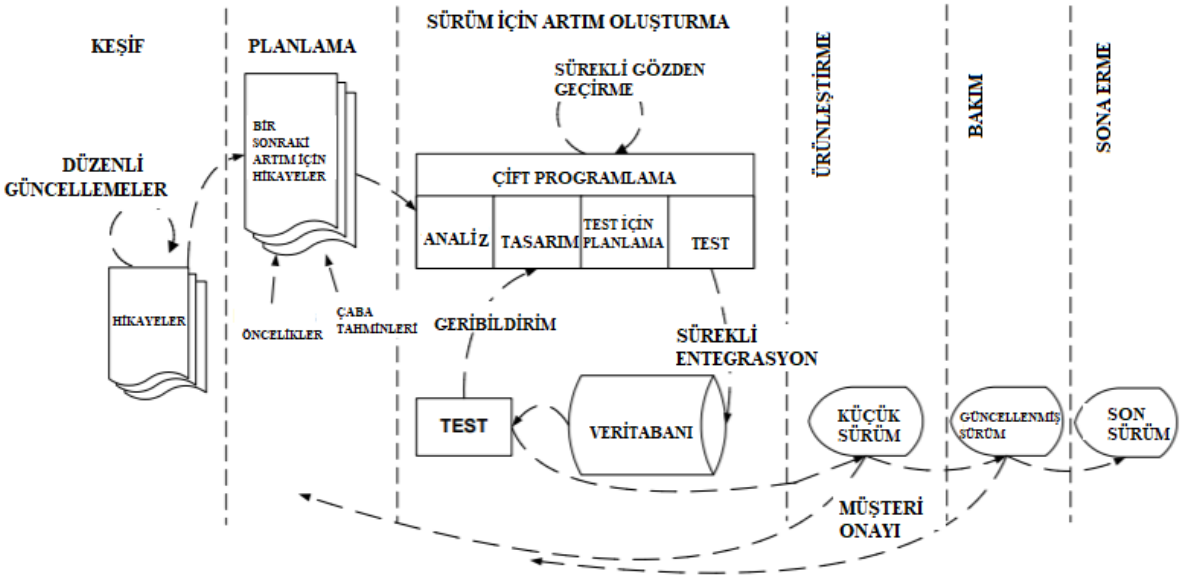
[90]Ernest Mnkandla ve Dwolatzky, Barry, "A Survey Of Agile Methodologies", *The Transactions Of The SA İnstitute Of Electrical Engineers*, Sayı:3, 2004, s. 236-247.

[91]Mark Paulk, "Extreme programming from a CMM perspective." *IEEE Software*, Sayı:18, No:6, 2001, s. 19-26.

[92]Lowell Lindstrom ve Jeffries, Ron, "Extreme Programming And Agile Software Development Methodologies", *Information systems management*, Sayı: 20, No: 3, 2004, s. 43.

3.2.1. Ekstrem Programlama Yaşam Döngüsü

Ekstrem Programlama yaşam döngüsü Keşif, Planlama, Sürüm İçin Artım Oluşturma, Ürünleştirme, Bakım ve Ölüm olarak altı aşamadan oluşur.[93] Şekil 3.2.1.1'de Esktrem Programlama Yaşam Döngüsü görülebilir.



Şekil 3.2.1.1 Ekstrem Programlama [94]

1. Keşif

Keşif aşamasında müşteriler ilk sürüme dahil olmasını istedikleri hikâye kartlarını yazarlar. Her hikâye kartı programa dahil edilmesi gereken bir özelliği belirtir. Proje ekibi projede kullanacakları teknoloji, araç ve tekniklere aşina olur. Kullanılacak teknoloji test edilir ve sistemin bir prototipi oluşturularak sistemin mimari olanakları

[93]Pekka Abrahamsson, Outi Salo, Jussi Ronkainen ve Juhani Warsta, "Agile Software Development Methods: Review and Analysis", *Proc. Espoo 2002*, 2002, s. 21.

[94] Abed Alsahli ve Ramadan, Nagy, "A Proposed Approach for Tracing the Progress of XP Projects" *Indian Journal of Science and Technology,* Sayı: 12, 2019, s. 3.

araştırılır. Keşif aşaması, büyük ölçüde teknolojinin programcılara ne kadar tanıdık geldiğine bağlı olarak birkaç hafta ila birkaç ay sürer.

2. Planlama

Planlama aşamasında hikâye kartları için öncelik sırası belirlenir ve ilk oluşturulacak küçük sürümün içeriği hakkında ortak bir karara varılır. Programcılar ilk olarak her hikayenin ne kadar çaba gerektireceğini tahmin eder ve akabinde bir proje takvimi oluşturulur. İlk sürümün yayınlanması genelde iki ayı geçmez. Planlama aşaması birkaç gün sürer.

3. Sürüm İçin Artım Oluşturma

Sürüm İçin Artım Oluşturma aşaması, ilk sürümün ortaya çıkmasından önce sistemde birkaç artım gerçekleştirilmesini içerir. Planlama aşamasında belirlenen program, her birinin uygulanması bir ila dört hafta sürecek bir dizi yinelemeye bölünmüştür. İlk yinelemede tüm sistemin mimarisiyle bir sistem oluşturulur. Bu, tüm sistem için yapıyı oluşturacak hikâyelerin seçilmesiyle elde edilir. Müşteri, her bir yineleme için seçilecek hikâyelere karar verir. Müşteri tarafından kararlaştırılan fonksiyonel testleri her yinelemenin sonunda çalıştırılır. Son yinelemenin sonunda sistem üretime hazırdır.

4. Ürünleştirme

Ürünleştirme aşaması sistem müşteriye sunulmadan önce sistemin performansını ölçmeye ve ekstra testlere ihtiyaç duyar. Bu aşamada da projeye eklenmesi gereken değişikliklerin bulunması mümkündür ve o anki sürüme eklenip eklenmeme kararının verilmesi gerekir. Yinelemelerin arasındaki süre düşürülebilir. Ertelenen fikirler ve öneriler daha sonra eklenmek için ya da bakım aşamasında projeye dâhil edilmek için belgelendirilir.

5. Bakım

İlk sürüm müşterinin kullanımı için uygun hale getirildikten sonra da XP ekibi sistemi ayakta tutmaya ve yeni artımlar oluşturmaya devam etmelidir. Bunu yapmak için Bakım aşamasında müşterinin görevleri gözden geçirmesi, geri dönüşte bulunması gerekmektedir. Böylece, sistem üretime geçtikten sonra geliştirme hızı yavaşlayabilir. Bakım aşaması, yapılacak işlemlerin içeriğine göre ekibe yeni kişilerin dâhil edilmesini ve ekip yapısının değiştirilmesini gerektirebilir.

6. Sona Erme

Bu aşama müşterinin sisteme dâhil edilecek hikâyesi kalmadığı zaman yaklaşır. Projenin müşterilerin teknik ihtiyaçlarının yanı sıra performans ve güvenilirlik gibi ihtiyaçlarını da karşılayabiliyor olması gerekmektedir. Bu aşama sistemde, mimaride, tasarımda ya da kodda değişiklik yapılmasına gerek kalmayıp gereken dokümantasyonların oluşturulduğu aşamadır. Sona erme proje istenilen talepleri karşılamadığı zaman ya da projenin maliyeti beklentiyi aştığında da gerçekleşebilir.

3.2.2. Ekstrem Programlama Değerleri

Ekstrem programlama başarılı bir yazılım için değerleri esas tutar. Bir başka deyişle ekstrem programlama değerleri ekstrem programlama prensiplerinin doğru yolda ilerlediğine dair bir garanti sunar. Bu değerler şunlardır:[95]

1. **İletişim:** Ekstrem programlama programcı-programcı ya da müşteri-programcı ile yapılan birim testi, programcı-müşteri ya da programcı-yönetici ile yapılan görev tahmini gibi iletişim gerektiren teknikler içerir. Bir ürün geliştirilirken ekip birlikte bir uyum içinde çalışmalıdır. Bir ekipte programcı, müşteri, test yapan kişi, yönetici bulunabilir. Gerektiği zaman birbirlerine danışabilmeli,

[95]Radmila Juric, "Extreme Programming And Its Development Practices", *Proceedings of the 22nd International Conference on Information Technology Interfaces*,2000, s. 97 - 104.

bilgi alışverişi yapabilme yeteneğine sahip olmalılardır. Projenin başarıya ulaşabilmesi için iletişim oldukça önemlidir.

2. **Basitlik:** Ekstrem programlama basitliği ön planda tutar. Programcılar projeyi yazarken kodu en basit seviyede tutmaya çalışırlar. Bir tasarım yapılırken sade bir şekilde yapılır ve o tasarım üzerinden yine en sade şekilde geliştirme yapılır. Ekip üyelerinin üzerinde çalışacağı işler en basit şekilde ayarlanmaya çalışılır.

3. **Geri Bildirim**: Ekstrem programlama yöntemiyle yazılan projede yapılan en küçük kod parçası, kodda yapılan en ufak değişiklik bile birim testi adı verilen testlere tabi tutulur. Dakikalık, günlük ya da aylık olarak farklı zaman aralıklarında geri bildirim almak projenin gideceği yolu çizmesini sağlar. Programcılar sistemin durumu hakkında dakikası dakikasına geri bildirim bulunurlar. Hikâyelerinin ne durumda olduğuna dair müşteriye bilgi verilir. Takipçi programın ilerleyişine dair gözlem yapar ve belli bir zaman kutusu varsa ona uygun olarak geribildirimde bulunur.

4. **Cesaret:** Ekstrem Programlamada sistemin daha iyi bir şekilde geliştirilmesi için keskin değişiklikler yapılabilir. Bir kod parçası beğenilmediği zaman atılıp baştan yazılabilir. Çok emek harcanan bir tasarım şartlara uygun olarak komple değiştirilebilir. Ekstrem programlamanın adının ekstrem olmasının sebebi bu tür kararları çekinmeden alma olgusunu benimseyen bir yapıda olmasıdır.

3.2.3. Ekstrem Programlama Prensipleri

XP değerleri projenin başarıya ulaşması için ortak bir hedef belirleyen motive olmuş ekip üyelerine belli bir çerçeve çizer. Prensipler ise ekstrem programlama değerlerinden yola çıkarak bu çerçeveyi daha da kalınlaştırır. Prensiplerin sayısı değişiklik göstermektedir fakat en yaygın prensipler şunlardır:

1. **Hızlı Geribildirim (Rapid Feedback):** Ekip üyeleri kendilerine yapılan geribildirimi çabuk bir şekilde kavrayıp geribildirime göre hareket ederler.

2. **Basitliği Varsaymak (Assume Simplicity):** Sistem geliştirilirken basitlik, hız ve sadelik ön planda tutulur. Hızlı geribildirim üzerine geliştirme yapılır. Geliştiricilerde buna göre hareket ederler ve gelecekte oluşabilecek değişiklikleri düşünmeden sadece o an ne gerekiyorsa o geliştirmeyi yapmalıdır.

3. **Artımlı Değişiklik (Incremental Change):** Bir sistem her ne kadar basit şekilde geliştirilse bile zamanla karmaşıklaşabilir. Yapılan en küçük değişiklik bile bir hataya yol açabilir bu yüzden kontrol altında tutulması gerekmektedir. Değişiklikler sık ve küçük şekillerde yapılmalıdır.

4. **Değişimi Kabul Et (Embracing Cshange):** Eğer müşteri ürünle ilgili bir şeyin değişmesi gerektiğine karar verirse programcılar bu kararı desteklemeli ve değişikliğin projeye ne şekilde dahil edilmesi gerektiğini planlamalıdır.

5. **Kaliteli İş (Quality Work):** İyi çalışan bir ekip ortaya gurur duyabilecekleri bir proje çıkarabilir.

6. **Kazanmak İçin Oyna (Play To Win):** Ekstrem programlama yönteminde her proje başarılı bir şekilde bitirilecek şekilde planlanır.

7. **Somut Deneyler (Concrete Experiments):** Sistem düzenli olarak hem birim testleriyle hem de bütün olarak sistem testleriyle kontrol edilir. Bu sayede ürün geliştirmede ne durumda olunduğu, nasıl ilerlendiği tespit edilir.

8. **Açık ve Samimi İletişim (Open and Honest Communication):** Ekstrem programlama sistemin geliştirilmesini ekip ile yürütür. Proje geliştirilirken ekip içerisinde birbirine güven duymak, şeffaf olmak projenin sağlıklı bir şekilde ilerlemesi için önemlidir.

3.2.4. Ekstrem Programlama Rolleri

Ekstrem programlama ekibi sistemi her zaman çalışır halde ve entegre bir şekilde tutar. Ekstrem Programlama yöntemi ekip içinde gerçekleştirmeleri gereken nitelikleri ve sorumlulukları olan çeşitli ekip üyeleri rolü tanımlar.[96] Beck'e göre daha verimli bir sonuç almak için bir ekip en fazla 12 kişiden oluşmalıdır ve her ekip üyesi üzerine düşeni iyi bir şekilde yerine getirmelidir.[97]

1. **Kullanıcı:** Kullanıcı geliştirme ekibine yol gösterecek, ne yapılması gerektiğini ve nasıl yapılması gerektiğini tanımlayan kullanıcı hikâyelerini yazar. Ayrıca kullanıcı hikâyeleri, kullanıcı tarafından yazıldığı için kullanıcı gereksinimlerden de sorumludur. Kullanıcılar fonksiyon testlerini hikâyeler üzerinden kontrol edip doğrular. Ekstrem Programlama 'Ekipteki Müşteri' diye özel bir kurala sahiptir. Sistemin muhtemel bir kullanıcısı olan uzman bir müşteri temsilcisi geliştirme ekibinin bir parçasıdır ve doğal olarak geliştirme ortamında bulunur. Ekibe uygun olan kullanıcı hikâyelerini açıklar, fonksiyon testlerini türetir. Projenin gerçek bir kullanıcısı yoksa örneğin arka planda çalışacak bir yazılım geliştiriliyorsa bu rolü oynaması için deneyimli bir kullanıcı tutulabilir ya da pazarlama ekibinden biri seçilebilir.

2. **Programcı:** Ekstrem Program programcı odaklı bir yöntemdir. Analizci, yazılım mimarları ya da tasarımcıları yerine iş programcılar tarafından gerçekleştirilir. Programcının iyi bir iletişim yeteneği olması gerekir. Kodlama konusunda iyi olması beklenir ve başka ekiplerle çalışabilmelidir.

3. **Koç:** Ekstrem Programlamanın yönetim kısmı iki role ayrılmıştır, Koç ve Takipçi. Koç projenin teknik ilerlemesinden sorumludur. Teknik olarak donanımlı, ilişkiler

[96]Thomas Dudziak, "Extreme Programming An Overview", *Methoden und Werkzeuge der Software: produktion WS*, 1999, s.15-16.

[97]Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-wesley professional, Boston, 2000, s. 157.

konusunda iyi biri olmalıdır. Koç projeyle ilgili yönetim toplantılarında yer alır. Koçun görevleri şunlardır;

- Proje sürecini yönetime ve müşteriye açıklamak
- Projenin geneline hâkim olmak
- Özellikle ekibin yeni üyeleri için geliştirme partneri olarak uygun olmak
- Test yapabilme, formatlama gibi teknik yeteneklere sahip olma

4. **Takipçi:** Proje içerisindeki ilerleyişi hangi noktada olunduğunu işin bitirme programı ile uyuşup uyuşmayacağının takibini yapan ekip üyesidir. Geliştirme ekibinin olduğu noktayı inceler, kullanıcıya fonksiyon testleriyle ilgili yardımcı olur. Genelde Koç ya da Programcı tarafından üstlenilen bir görevdir.

5. **Danışman:** Kurallar ve uygulamalardan ötürü Ekstrem Programlama ile geliştirilen bir projede uzmana pek rastlanmaz fakat bazen daha fazla bilgiye ihtiyaç duyulan projeler olabilir. Bu gibi durumlarda bir danışman tutulur ve geliştirme ekibine, müşteriye bilgileriyle yardımcı olması beklenir.

3.2.5. Ekstrem Programlama Teknikleri

Ekstrem Programlama teknikleri kullanıcı ve ekibin ekstrem programlama değer ve prensiplerini uygulama konusunda yardımcı olur.[98]

1. **Ayakta Toplantı (Stand-up Meeting):**Genellikle 15 dakika süren toplantılardır. Ekip günlük olarak toplanıp ne yaptıklarına ne yapacaklarına dair konuşurlar. Herhangi bir sorun varsa tartışılıp çözüm bulmaya çalışılır.

2. **Oyunu Planlama (Planning game):**Projenin her adımı için yapılan bir planlamadır. Müşteriler, programcılar tarafından sağlanan tahminlere göre her

[98]Kent Beck, "Embracing Change With Extreme Programming", *Computer,* Sayı: 32, No: 10, 1999, s. 70-77.

sürümün kapsamına ve zamanlamasına karar verir. Programcılar yalnızca o yinelemedeki hikayelerin talep ettiği işlevselliği uygular.

3. **Küçük Sürümler (Small releases):** Ekstrem programlamada yapılan her değişiklik işlenerek küçük sürümler çıkarılır. Küçük bir sistem hızlı bir şekilde üretime geçirilir şeklinde ifade edilebilir. Sistem bütün problemi çözmeden evvel birkaç ayda yapıma koyulur. Yeni sürümler günlük ya da aylık olarak sık sık yapılır ve müşteri tarafından incelenir. Karşılanmayan talepler varsa, bir sonraki sürümde karşılanır.

4. **Metafor (Metaphor):**Sistemin mimarisinin, şeklinin müşterinin ve programcıların anlayabileceği bir şekilde tanımlanmasıdır. Örneğin bir hesap makinesi programı geliştirilirken gerçek bir hesap makinesi göz önüne getirilip onun üzerinden sistemin nasıl çalışacağı tartışılabilir.

5. **Basit Tasarım (Simple design):**Ekstrem Programlamada yalın bir tasarım yapılır, bu sayede yazılımın üzerine koymak geliştirmek daha kolay olur. Tasarımlar her an testlerden geçirilebilir, programcının iletişim kurmak istediği şeylerle iletişim kurar. Tekrarlayan kod içermez ve mümkün olan en az sayıda muhtemel sınıflar ve metotlar içerir. Bu kural 'Her şeyi sadece ama sadece bir kere söyle' olarak özetlenebilir.

6. **Test:** Programcılar sürekli bir şekilde dakikası dakikasına birim testleri yazarlar. Bu testler bir araya toplanıp doğru bir şekilde çalıştırılmalıdır. Müşteriler bir artım içindeki hikâyelere fonksiyon testleri yazarlar. Bu testlerin hepsi sorunsuz çalışmalıdır fakat bazen hatalı bir ürünü yayınlamanın maliyeti ile erteleme maliyeti karşılaştırılarak bir karar verilmesi gerekebilir.

7. **Yeniden Yapılandırma (Refactoring):** Sistemin tasarımı var olan ve bütün testleri uygun bir şekilde uygulayabilen tasarımın değişimleriyle evrilir. Duplikasyona yer vermemek, kodu geliştirmek ve basitleştirmek, ürüne esneklik

katmak, sistem karmaşıklığını azaltmak için yeniden yapılandırılma gerçekleştirilir.

8. **Çift Programlama (Pair programming):** Bütün kodlama tek bir ekran/klavye/fare kullanılarak iki kişi tarafından yazılır. Birlikte çalışan programcılar deneyimlerini birbiriyle paylaşır, gerektiği zaman gereken yerde birbirlerine destek olurlar. Kod yazılımı sırasında birbirlerini gözlemledikleri için projenin ilerleyişini de kontrol edebilirler.

9. **Sürekli Entegrasyon (Continuous integration):** Oluşturulan yeni kod birkaç saat geçmeden var olan sisteme entegre edilir. Entegrasyon yapılırken sistem sıfırdan gözden geçirilmeli ve bütün testlerden olumlu sonuç alınmalıdır, aksi takdirde yapılan değişiklikler ortadan kaldırılır.

10. **Ortak Sorumluluk (Collective ownership):** Her programcı bir fırsat gördüklerinde sistemin herhangi bir yerinde herhangi bir zaman herhangi bir kodu geliştirebilir. Ekstrem Programlamada bir kişinin kodun bir bölümünü sahiplenmesi gibi bir durum söz konusu değildir, herkes eşit şekilde projeden sorumludur. Yani programcıların hepsinin projenin her köşesini bilir ve daha etkili bir şekilde geliştirme yapabilirler.

11. **Ekipteki Müşteri (On-site customer):** Bir müşteri sürekli ekiple birlikte aynı yerde bulunur. Bu sayede müşteri gözünden projenin ilerleyişini takip edebilir, ekip ile birlikte çalışarak belirsizliklerin kolay bir şekilde ortadan kaldırılmasına imkan kılar ve proje daha düzgün ilerler.

12. **40 Saatlik Haftalar (40-hour weeks):** Bu pratik hiç kimsenin iki hafta üst üste mesaiye kalarak verimli bir şekilde çalışamayacağını belirtir. Eğer böyle bir şey gerçekleşirse, bu durum bir sorun olarak ele alınır.

13. **Açık Ofis (Open workspace):** Ekip üyeleri büyük bir odada etrafı açık olan masalarında çalışır. Çift programcılar merkeze kurulmuş bilgisayarlar üzerinden kodlama yaparlar.

14. **Kod Standartları (Coding Standarts):** Proje geliştirilirken izlenmesi gereken kod yazma standartlarının dokümantasyonu yapılır. Programcıların kodları takip etmesini ve ortak bir dil kullanarak yazılım geliştirmesini sağlar, daha sonra projeye dahil olan birinin ise işi kolaylaştırılmış olur.

15. **Sadece Kurallar (Just Rules):** Proje geliştirilirken ekibin kendi içinde koyup takip ettiği kurallar bulunur. Bu kurallar duruma göre sürecin daha iyi ilerlemesi için ortak olarak düşünüp taşınılarak her an değiştirilebilir.

3.2.6. Ekstrem Programlama Avantajları

- Kısa süre içerisinde ortaya çalışan bir ürün çıkarma çabasında olduğu için hızlıdır.
- Müşteriye sık sık sistemin ilerleyişiyle ilgili bilgi vererek müşteri memnuniyetini amaçlar. Müşteri ile ilişkileri ön planda tutarlar.
- Sürekli olarak entegrasyon ve geliştirme yapılır.
- Ekipte müşteri tarafının isteklerini iyi bir şekilde açıklayacak birisi olduğu için gereken yönlendirmeler ile istenilen ürüne ulaşabilirler.
- Ekipteki müşteri sayesinde ürünü sık sık müşteriye sunma döngüsü azaltılmıştır, doğal olarak değişiklik yapılırken harcanan maliyet de azalmış olur.
- Müşterinin ekipteki temsilcisi ürünün geliştirilmesi için müşterinin isteklerini tanımlamak adına hikâye kartları hazırladığı için gereksinimler daha açıktır.
- Sık yapılan toplantılar sayesinde hatalar tespit edilip kolayca çözülür.
- Ekipte herkes birbirine karşı şeffaftır. Bu da bilgi paylaşımını ve ekip içi bağları arttırır.
- Ekipte herkes kendi işini yapsa da bütün projeye hâkimdir ve bir sorun çıktığında müdahale edebilirler. Bu da zaman konusunda yardımcı olur.
- Basitlik ön plandadır. Programcılar her an müdahale edilebilecek basit kodlar oluştururlar.

3.2.7. Ekstrem Programlama Dezavantajları

- Açık ofiste çalışmayı ön plana aldığı için coğrafi olarak aynı ortamda bulunmayan programcılar için uygun bir yöntem değildir.
- Tasarımdan ziyade koda önem verilen bir yazılım geliştirme yöntemidir. Müşteri için genel olarak görsel ön planda olduğu için son ürün hoşlarına gitmeyebilir.
- Üründe yapılan değişikliklerin her biri belgelenmez. Bir hata düzeltilse bile dokümantasyon olmadığı için tekrar aynı hata ile karşılaşmak mümkündür.
- Kısa sürede çabuk ve çalışan bir ürün çıkartılmaya çalıştığı için ekip üyeleri strese girebilirler. Bir ürün ortaya çıkarırlar ama ürünün kalitesi beklenilen şekilde olmayabilir.
- Sık müşteri katılımının gerekmesi bir dezavantajdır zira müşteri her zaman uygun olmayabilir. Bu da zaman ve ilerleme konusunda projeyi beklenen şekilde ilerlemekten alıkoyabilir.
- Sistemin başarılı olması gereksinimlerin doğru tanımlanması ve yeterli olmasına bağlıdır. Kullanıcıdan geri dönüş alınması oldukça önemlidir.

3.3. KANBAN

Kanban Japoncada pano anlamına gelen, belirlenen süre içerisinde kullanılan araç-gereçleri ve yapılan işlerin kontrolünü sağlamak amacıyla kullanılan bir çizelge yaklaşımıdır.[99] 1953 yılında Taiichi Ohno tarafından Toyota'nın üretim verimliliğini arttırmak için ortaya atılmıştır. Şekil 3.3.1'de ilk kullanımına dair bir örnek görülebilir. 2000'li yılların başında ise David Anderson tarafından yazılım alanında uygulanmaya başlanmıştır.[100]

[99]Chun-Che Huang ve Andrew Kusiak, "Overview of kanban systems", *Int. J. Computer Integrated Manufacturing*, 1996, s. 169.

[100]Ahmad, Muhammad Ovais, Markkula, Jouni ve Oivo, Markku, "Kanban In Software Development: A Systematic Literature Review." *2013 39th Euromicro conference on software engineering and advanced application IEEE*, 2013, s. 9-16.



Şekil 3.3.1 Toyota'dan orijinal bir kanban panosu Toyota[101]

Kanban, yazılım geliştirme sürecinde devam eden çalışmayı görselleştirmeyi ve sınırlandırmayı temel alan bir yöntemdir. Belli bir zaman içerisinde yazılım geliştirmeyi benimseyen bir yapısı vardır. Kanban uygulama için tam zamanında yazılım ürününün teslimini kolaylaştırmak için iş planlamasına vurgu yapar. Ana odağı, hangi işin yapılması gerektiğini ve ne zaman yapılması gerektiğini doğru bir şekilde belirtmektir. Bunu, görevleri önceliklendirerek ve iş akışının yanı sıra teslimata kadar olan teslim süresini tanımlayarak yapar.[102] Kanban süreci, projenin tamamlanma riskini azaltmak ve ayrıca projedeki diğer görevler arasında esnekliği artırmak için en çok dikkat edilmesi gereken en önemli görevleri açıkça sunar. Günümüzde giderek popülerlik kazanan bir yöntemdir.

Kanban oldukça basit bir fikri temel almaktadır: Devam Eden İş (Work In Progress, WIP) sınırlı olmalı ve yeni bir şey yalnızca mevcut iş teslim edildiğinde veya işlevleri tamamlandığında başlatılmalıdır. Devam Eden İş sınırı, geliştirme sürecindeki her adımın kapasitesini, her iş akışı durumunda devam eden iş öğelerinin sayısı açısından tanımlar. Uygun Devam Eden İş sınırları, bir sisteminin aşırı

[101]Toyota Global, https://www.toyotaglobal.com/company/history_of_toyota/75years/text/entering_the_automotive_business/chapter1/section4/item4.html, (21.02.2021)

[102]M. O. Ahmad, J. Markkula, Jouni ve M. Oivo, 2013, s. 9-16.

yüklenmemesini sağlar, böylece sürdürülebilir bir geliştirme hızı ile projeyi ilerletmeyi mümkün kılar.

Kanban yazılım geliştirmede kullanılabilmesi için işi küçük parçalara böler. Kanban çoğunlukla çevik yöntemlerle birlikte kullanıldığından, her iş öğesi genellikle kağıt not kartına yazılan bir kullanıcı hikayesi olarak temsil edilir. Gerektiği takdirde kullanıcı hikayeleri daha küçük parçalara bölünür. [103] Planlama ve performans ölçümünü daha iyi yönetmek için tüm iş öğelerinin yaklaşık olarak aynı boyutta olması önerilir. Ancak Kanban yöntem olarak çalışma öğelerinin ne yapısını ne de boyutunu belirler. Çalışma öğeleri, geliştirme sürecinin çeşitli aşamalarında işin nasıl aktığını gösteren görsel bir kontrol mekanizması olarak hizmet veren bir Kanban panosunda sunulmalıdır. Şekil 3.3.2'de bir örneği görülebilen Kanban panosu genellikle üzerinde yapışkan notlar bulunan beyaz bir tahta ya da elektronik bir tahta olarak ele alınır.



Şekil 3.3.2 Kanban Panosu Örneği[104]

Kanban panosu, geliştirme sürecinde bir iş öğesinin bulunabileceği çeşitli durumları temsil eden bir dizi sütundan oluşur. Geliştirme yaşam döngüsü boyunca ilerleyen iş, kartlar son sütunda bitene kadar bir durumdan diğerine geçer. Her sütunun

[103]Mike Cohn, *User stories applied: For agile software development*, Addison-Wesley Professional, Boston, 2004, s. 23.

[104]Serdar Baysan ve Durmuşoğlu. Bülent "*Ürün Geliştirmede Yalın Süreçlere Dönüşüm*", İstanbul Teknik Üniversitesi, 2013, s. 7.

üstünde, herhangi bir zamanda karşılık gelen iş akışı durumunda kaç kartın olabileceğini gösteren bir Devam Eden İş sınırı vardır. Bir kart tamamlandığında sıradaki sütuna ilerler ve doğal olarak kendi yerinde bir boşluk bırakır. Bu da geliştirme ekibinin bir önceki sütunda bitmiş olan bir işi o açılan boşluğa getirmesine imkan kılar. Herhangi bir sebepten ötürü bir sütundaki işler tamamlanıp ilerleyemezse bu sütun eninde sonunda WIP sınırına ulaşır. Bu durumda geliştirme ekibi bitmeyen işleri üst üste yığmak yerine, oluşan darboğazı çözmelidir.

Kanban yazılım geliştirme ekibinin üretkenliğini ve projedeki ilerleyişini ölçmek için belirlenmiş teslim süreleri içerisinde işleri bitirmeyi planlar. Teslim süresi, teslimatı tahmin etmek ve hizmet seviyesi taahhütlerinde bulunmak için kullanışlıdır. Belirlenen teslim süresi içerisinde kullanıcıya verilen işler hem projenin ilerleyişine katkıda bulunur hem de kullanıcı memnuniyetini arttırır. Kullanıcı hikayeleri benzer boyuttaki iş artımlarına bölünürse, çaba tahminine gerek kalmaz ve ekibin teslimat kapasitesini ölçmek için gereken tek ölçü teslim süresi olur.[105]

3.3.1. Kanban Değerleri

İşbirliği yaparak bir arada çalışan bireylerin ortaya iyi bir yazılım sistemi çıkarması adına Kanban'ın ortaya koyduğu bazı değerler vardır. Bunlar şeffaflık, denge, işbirliği, müşteri odaklı olmak, akış, liderlik, anlayış, ortak bir karara varma ve saygıdır.[106]

1. **Şeffaflık:** Açıkça bilgi paylaşımında bulunmak projenin ilerlemesini geliştirir.

2. **Denge:** Farklı bakış açıları, düşünceler ve yeteneklerin dengeli olması gerektiğini anlamak verimlilik için gereklidir.

[105]Viljan Mahnič, "Applying Kanban Principles To Software Development", *Information Technology for Practice*, 2013, *s.* 89.
[106]D. J. Anderson ve A. Carmichael, 2016, s. 3-5.

3. **İşbirliği:** Kanban yöntemi insanların birlikte çalışmasını merkez alır bu yüzden işbirliği oldukça önemlidir.

4. **Müşteri Odaklı Olmak:** İyi bir proje ortaya çıkarmak için müşteri katkısı esastır.

5. **Akış:** Sistem bir akış halinde gelişmektedir.

6. **Liderlik:** Ekip üyelerinin birbirini hareket geçirmesi ve projeye odaklanmasını sağlamak Kanban için önemlidir.

7. **Anlayış:** Müşterinin ne istediği, ne yapılması gerektiğini, ne istendiğini kavramak projenin sağlıklı bir şekilde ilerlemesi için önemlidir.

8. **Ortak Bir Karara Varma:** Birbirlerinin düşüncelerine saygı duyarak ortak bir amaca odaklanmak gereklidir.

9. **Saygı:** Birlikte çalışmak Kanban'ın merkezinde yer alır ve birlikte çalışan kişiler birbirlerinin rollerine, sorumluluklarına ve unvanlarına saygı duymalıdır. Bu şekilde iyi bir ortamda sağlıklı bir şekilde proje geliştirebilirler.

3.3.2. Kanban Prensipleri

Anderson Kanban yöntemini artımlı, evrimsel bir süreç ve organizasyonların bilgisine göre değişen sistemler olarak formüle etmiştir. Kanban'ın işleri bitirmeye odaklanan bir yapısı vardır ve temel olarak dört prensiple ifade edilir.

1. **Şu an yaptığın şeyle başla (start with what you do now):** Kanban'ın esnekliği, halihazırda başarıyla yapılmakta olan işi kesintiye uğratmadan mevcut iş akışları, sistemler ve süreçler üzerine yerleştirilmesine olanak tanır. Bir kişiden çalışma şeklini ve tarzını değiştirmesi istenmez, ne şekilde biliyorsa o şekilde çalışmaya devam etmesi beklenir.

2. **Artımlı, evrimsel değişiklikler olmasını onayla (Agree to pursue incremental, evolutionary change):** Ekip ve müşteri proje geliştirme süreci içerisinde mevcut koşulların iyileştirme için evrimsel bir yaklaşımı gerektirdiğini kabul etmelidir. Kanban yöntemi minimum direnç ile karşılaşacak şekilde tasarlanmıştır. Proje süresince küçük artımlı ve evrimsel değişiklikleri teşvik eder.

3. **Mevcut sürece, rollere, sorumluluklara ve iş unvanlarına saygılı ol (Respect the current process, roles, responsibilities and job titles):** Kanban, mevcut süreçlerin, rollerin, sorumlulukların ve unvanların bir değeri olduğunu ve genellikle korunmaya değer olduğunu kabul eder. Kanban yöntemi değişimi yasaklamaz, ancak onu "her derde deva" olarak da reçete etmez. Değişim korkusunu tetiklemeden artımlı, mantıksal değişiklikleri teşvik etmek için tasarlanmıştır.

4. **Her seviyede liderlik eylemlerini destekle (Encourage acts of leadership at all levels):** Kanban liderlik değerine oldukça önem verir. Liderlik değişimi katalize ederek projeyi daha ileri götürebilir. Öneriler, geri bildirimler ekip içerisinden ya da müşteriden gelebilir. İstenilen değişiklikler önemsiz görülmemeli, öneren kişiler cesaretlendirilmelidir.

3.3.3. Kanban Teknikleri

Anderson'a göre Kanban teknikleri genel olarak kanban sistemlerini yöneten kişiler için gerekli olan aktiviteleri içerir.[107]

1. **Görselleştir (Visualize):** Kanban yönteminde mümkün olduğunca bütün süreç görselleştirilmeye çalışılır. Böylece projenin ilerleyişi, olası hatalar daha rahat görülebilir. Sistem küçük parçalara bölünerek geliştirilir. Süreci görselleştirmek iş için gelen talepleri ve istekleri daha iyi anlamayı mümkün kılar. İş akışının görselleştirilmesi riskleri göz önüne serdiğinden gerekli işin tüm ekip üyeleri

[107]D. J. Anderson ve A. Carmichael, 2016, s. 17-26.

tarafından görünür kılınması proje için bir artıdır. Kanban panosu aracılığıyla iş akışı net bir şekilde görselleştirilebilir.

2. **Devam Eden İşi Sınırla (Limit WIP):** Aynı anda kaç iş yapılacağı belirlenmeli ve ona göre bir sınır çizilmelidir. İşler devam etmeden yeni işlere başlamak projeyi darboğaza sokabilir ve ilerlemesine engel olabilir. Ekip ne kadar sürede ne kadar işi nasıl yapabileceğini bilmeli ve ona göre kendini sınırlamalıdır.

3. **Akışı Yönet (Manage Flow):**İş akışındaki her durumun akış ölçümü olarak bilinen şekilde izlenip raporlanarak devam eden işin bitirilmesi amaçlanır. Kanban panosundaki iş akışı takip edilmeli ve geliştirilmelidir. Geliştirme ekibi işlerin durumlarını ve ilerlemelerini analiz etmeli, varsa sorunları tespit etmeli, yapılacak değişikliklere karar vermeli ve bunları uygulamalıdır.

4. **Süreç Politikalarını Açık Hâle Getir (Make policies explicit):** Kanban'ın yazılım geliştirirken önem verdiği değerlerden birisi şeffaflıktır. Hem müşteri hem de yazılımcılar işlerin nasıl yürüdüğünü, ne yaptığını gerçekten anlamış olmalı ve ona göre hareket etmelidir. Bir projenin nasıl yürüdüğünü bilmeden o proje hakkında tam bir bilgi sahibi olunamaz ve dolayısıyla projeye yeterli seviyede katkıda bulunulamaz.

5. **Geribildirimi Yerleştir (Implement feedback):** Kanban çalışmak için geribildirimlere ihtiyaç duyar. Ayakta toplantı, hizmet incelemesi, operasyon incelemesi ve risk analizi yaparak elde edilen sonuç ile beklenen gerçek sonucu karşılaştırır.

6. **Sürekli Geliştirme (Continuous Improvement):** Açık bir şekilde belirtilen ve ekip arasında paylaşılan iş teorileri, iş akışı, süreç ve hata analizleri ekip üyelerinin probleme dair ortak bir anlayış oluşturmasını sağlar. Bu sayede yapılabilecek geliştirmelere birlikte karar verebilirler.

3.3.4. Kanban Avantajları

- Kanban gelişmeye açık bir modeldir. Akışkan bir yapı benimsemiştir. Bu da istenen değişiklikleri rahat bir şekilde sisteme dahil etmeyi kolaylaştırır.
- Sadece gereken seviyede ve miktarda iş yapılmasını destekler. Ekip üyelerinin gereksiz iş yapmadıklarından ve zaman harcamadıklarından emin olur.
- Kanban proje geliştirme sürecini görsel olarak sergilemeyi savunduğu için anlaşılması kolaydır. Projenin neresinde olunduğu özellikle Kanban panosundan açık bir şekilde görülebilir.
- İşi bitirmek için süreç boyunca bütün ekip hızlı ve senkronize bir şekilde çalışır.

3.3.5. Kanban Dezavantajları

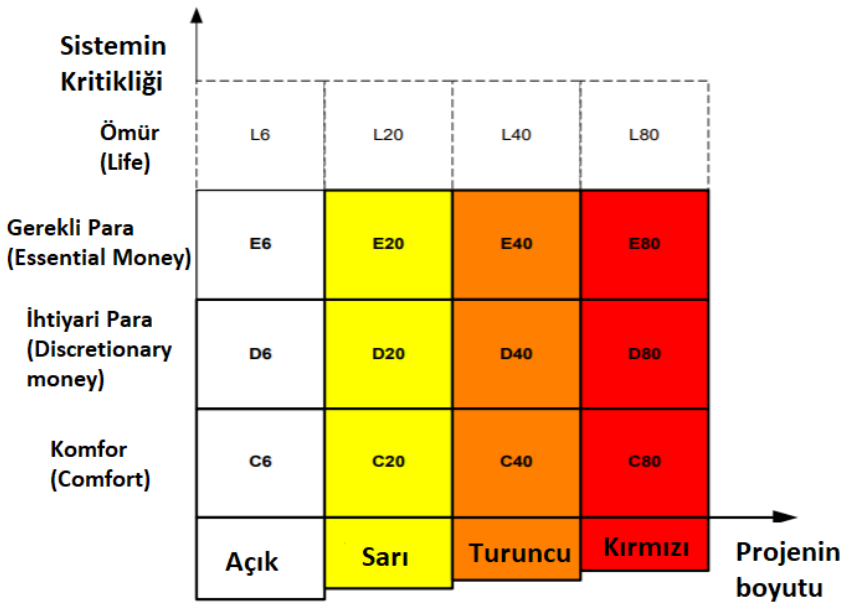
- Güncel olmayan bir Kanban panosu projenin ilerlemesi konusunda problem çıkarabilir. Ekip Kanban panosunu güncel tutmalıdır, aksi takdirde bir proje henüz bitmemiş bir panoya dayanarak geliştirilebilir ve istenilen sistem ortaya çıkmaz.
- Kanban panosu açık ve anlaşılır olmalıdır. Ekip üyeleri işlerle ilgili yapılacakları kısa ve öz olarak eklemediği takdirde pano zamanla karmaşıklaşır. Bu da projenin gidişatında zorluk oluşturabilir.
- Kanban yönteminde zaman belirli değildir. Kanban panosundaki işler devam ediyor, tamamlandı şeklinde tanımlanır ama hangi işin ne zaman tamamlandığı ya da ne kadar sürede tamamlanması gerektiğine dair zaman dilimleri yoktur. Bu yüzden geliştirmenin ne kadar süreceği kestirilemez.

3.4. KRİSTAL GELİŞTİRME YÖNTEMLERİ

Kristal yöntemler Alastair Cockburn tarafında geliştirilmiş, her bir proje için en uygun yöntemi seçmek üzere bir dizi farklı yöntem içeren bir yazılım geliştirme şeklidir. Kristal yaklaşım sadece çeşitli yöntemler değil aynı zamanda o yöntemlerin projeden projeye değişecek şekilde uyarlanabilecek ilkelerini de içerir. Cockburn Kristal yöntemleri, genetik bir koda dayalı, aile gibi benzerlik taşıyan bir yazılım

geliştirme yaklaşımı olarak tanımlar.[108]Proje geliştirme boyunca süreç önemlidir fakat kişiler arası uyum ve iletişim daha da önemli olarak kabul edilir.

Kristal yöntemlerin her birinin temel aldığı prensipler ve değerler aynıdır fakat projenin ve kullanılacak yöntemin ağırlığına göre yöntem değişir ve farklı bir renkle ifade edilir. Yöntemler Şekil 3.4.1'de görülebilir. Her geliştirme ekibi kendi yetenek ve becerilerini bilir ve projeyi geliştirirken kendilerine uygun renkte bir yöntem seçer. Renk ne kadar koyu ise yöntem o kadar ağırdır. Kristal geliştirme yöntemleri oluşturulacak projenin boyutuna ve kritikliğine bağlı olarak uygun rengi seçmeyi önerir. Büyük projeler küçük projelere kıyasla daha fazla kontrol ve ağır yöntemler gerektirecektir.



Şekil 3.4.1 Kristal Yöntemi[109]

[108]Alastair Cockburn, *Crystal clear: A human-powered methodology for small teams: A human-powered methodology for small teams*, Pearson Education, 2004, s. 238.

[109] P. Abrahamsson, O. Salo, J. Ronkainen ve J. Warsta, 2002, s. 39.

Kristal yöntemler ailesinde ortak olan öncelikler proje sonucu, geliştirmedeki verimlilik ve sözleşmelerin gerçekleştirilebilmesidir. Kristal, proje ekiplerin ilk üç özelliği Kristal yöntemlerin özü olan yedi güvenlik özelliğini benimsemesini önemser. Cockburn'e göre bu özellikler şunlar;

- Sık ürün teslim etme (frequent delivery)
- Yakın bir iletişime sahip olma (close communication)
- Kişisel güvenlik (personal safety)
- Odaklanma (Focus)
- Uzman kullanıcılara kolay erişim (easy access to expert users)
- Yansıtıcı gelişme (reflective improvement)
- Otomatik test, konfigürasyon yönetimi ve sık entegrasyon ile teknik bir ortam (technical environment with automated testing, configuration management and frequent integration)

Kristal yöntemlerden özellikle açık ve turuncu olan yöntem çok sık kullanılmaktadır. Açık Kristal Yöntemi genelde 6 kişilik ekiplerden oluşan küçük boyutlu projeler için tasarlanmıştır. Bu yöntemi kullanan bir ekip aralarındaki iletişimi arttırmak için açık ofis içerisinde çalışmalıdır. Geliştiriciler proje sırasında istedikleri aracı kullanabilirler. Açık Kristal Yöntemi yazılım geliştiricilerin uygun araçları ve gereçleri seçmesi için seçenekleri açık tutar ve kararı onlara bırakır. Dokümantasyon oldukça azdır. Gereksinimler UML ile ifade edilir. İlk artımlı döngü genelde üç ayı geçmez ve her artım sonunda ürün teslim edildiğinde bir değerlendirme toplantısı yapılır. Turuncu Kristal yöntemi 10-40 kişilik ekiplerden oluşan orta boyutlu projeler için tasarlanmıştır. Genelde proje süreleri bir ile iki yıl arasında değişir. Açık Kristal gibi Turuncu Kristal de araç-gereç konusunda geliştiricileri özgür bırakır fakat Açık Kristal'den daha çok dokümantasyona ihtiyaç duyar. İlk artım süresi genelde dört aydır ve yazılımı test etmek desteklenen bir gerekliliktir.[110]

[110]Malik Qasaimeh, Mehrfard, Hossein ve Hamou-Lhadj, Abdelwahab, "Comparing Agile Software Processes Based On The Software Development Project Requirements", *2008 International Conference on Computational Intelligence for Modelling Control & Automation*, 2008, s. 51.

3.4.1. Kristal Yöntemlerin Avantajları

- Genel olarak bütün yazılım projelerinde etkili ekip iletişimi projeyi başarıya götüren anahtar etmendir. Kristal yöntemler değişik sayıda üye içeren ekipler için ekip iletişimi ile ilgili anlamlı ve düzgün bir rehberlik sağlar.
- Kristal yöntemler her projenin kendi ihtiyaçları doğrultusunda kendi kurallarına, tekniklerine ve süreçlere ihtiyacı olduğu düşüncesini taşır. Projenin özelliklerine göre yapı olarak esneklik sağlar ve uygun yöntemin seçilmesini mümkün kılar.
- Kristal Yöntemler iyi bir risk kontrolü ve teknik yönetimi sağlar.
- Kristal Yöntemler çeşitli takım boyutlarına uyum sağlayabilir, bu sayede farklı boyuttaki ve önemdeki projeleri geliştirmek için farklı kristal yöntemler seçilebilir.[111]

3.4.2. Kristal Yöntemlerin Dezavantajları

- Kristal yöntemlerde sadece açık ve turuncu kristal yani iki yöntem detaylı tanımlanmıştır ve bu yöntemler tasarım ile kod doğrulama etkinliklerinden yoksundur.
- Kristal yöntemler iş girişimciliği ile ilgili rehberlik etmez.
- Kristal yöntemler, sistem doğrulama uygulamalarından yoksundur bu da önemli sistemlerin geliştirilmesi için uygunsuz kılar.
- Kristal yöntemlerde iyi tanımlanmış bir ekip yapısı bulunmamaktadır.[112]

3.5. DİNAMİK SİSTEM GELİŞTİRME YÖNTEMİ (DYNAMIC SYSTEM DEVELOPMENT METHOD)

Dinamik Sistem Geliştirme Yöntemi 1990lı yılların ortasında İngiltere'de ortaya çıkmış bir yöntemdir. Hızla gelişen yazılım projelerine yardımcı olup daha etkili bir

[111]Harleen Flora ve Chande, Swati "A Systematic Study On Agile Software Development Methodologies And Practices" *International Journal of Computer Science and Information Technologies*, Sayı:5, No:3, 2014, s. 3632.

[112]P. Abrahamsson, O. Salo, J. Ronkainen ve J. Warsta, 2002, s. 42.

süreç geçirmek için tasarlanmıştır. Yazılımda yüksek kaliteye önem verir, değişikliklere açık bir yaklaşım benimser. Belirlenen bir süre ve maliyet içerisinde değişen gereksinimleri de göz önüne alarak hızlı teslimata odaklanan ve kullanıcıyı proje boyunca dâhil eden yinelemeli ve artımlı bir yaklaşımdır.[113] Zaman ve kaynakları tanımlar, daha sonra geliştirilecek işlevselliği ayarlar.

3.5.1. Dinamik Sistem Geliştirme Yöntemi Prensipleri

Dinamik Sistem Geliştirme Yöntemi ile geliştirilen her proje için gerekli olan birtakım prensipler vardır. Bu prensiplerin herhangi birinin göz ardı edilmesi felsefenin dışına çıkılmasına sebep olur ve projede oluşabilecek risk ihtimalini önemli ölçüde arttıracaktır. Aslında bu, bir yazılım geliştirilirken bazı adımların göz ardı edilmesini ya da projeye göre ayarlanmasını tolere edebilen Dinamik Sistem Geliştirme Yönteminin yapısına ters düşen bir durumdur. Bahsi geçen prensipler şunlardır:[114]

1. "**Aktif Kullanıcı Katılımı Zorunludur"**

İlk prensip en önemli prensip olarak kabul edilir zira proje geliştirme süresi boyunca kullanıcının katılımı olması, kullanıcı açısından hataları etkili bir şekilde azaltır. Daha az hata olması hata maliyetlerini de düşürür. Dinamik yöntem büyük kullanıcı gruplarıyla çalışmak yerine küçük, belli bir grup kullanıcı ile periyodik olarak çalışma ve değerlendirme görüşmeleri yapmayı önerir. Projenin sağlıklı bir şekilde ilerlemesi için devamlılık gereklidir.

[113]Iqra Zafar, Khan Nazir, Aiman ve Abbas, Muhammad "The Impact Of Agile Methodology (DSDM) On Software Project Management", In *Circulation in Computer Science: International Conference on Engineering, Computing and Information Technology (ICECIT 2017)*, 2017, s. 2.

[114]Benjamin Voigt, "Dynamic system development method", *no. January*,2004, s. 2-4.

2. **“Ekipler Karar Vermek İçin Yetkilendirilmelidir”**

Projede çabuk ilerleyebilmek için ekipler kendi başlarına karar verebilmelidir. En küçük değişiklik ya da kaynak için izin istemek bürokrasiyi arttıracak ve arada geçen süre projenin ilerleyişine negatif bir şekilde yansıyacaktır.

3. **“Sık Sık Teslimat Yapmaya Odaklan”**

Sık ürün teslimi hataların daha çabuk tespit edilmesini sağlar ve kolay bir şekilde bu hatalar ortadan kaldırılıp sorunun kaynağına inilebilir.

4. **“İşe Uygunluk, Kabul Edilen Ürünler İçin Bir Kriterdir**”

Dinamik Sistem Geliştirme Yönteminin esas amacı iş ihtiyaçlarını karşılayacak kadar iyi bir ürün ortaya çıkarmaktır. Daha sonra bir sonraki artımda bu ürün üzerinde geliştirme yapılabilir.

5. **“Yinelemeli ve Artımlı Geliştirme Zorunludur”**

Projenin karmaşıklığını yönetilebilir durumda tutmak için küçük parçalara ayırmak gerekir. Yazılım geliştirme yinelenir ve her artımda ürüne yeni bir özellik eklenir. Bu durum geliştirilen ürün, iş gereksinimlerini tamamen karşılayana kadar devam eder. Bu prensip her yazılım sisteminin değişebileceği olgusunu kabul etmeyi gerektirir. Artımlar ne kadar küçük olursa değişen gereksinimlere yanıt vermesi o kadar kolay olur.

6. **“Geliştirme Sırasında Yapılan Bütün Değişiklikler Geri Döndürülebilir Olmalıdır”**

Yazılım projelerinde bir değişiklik meydana gelmesi bütün sistemi etkileyen bir durumdur. Bir geliştirme sürecini tersine çevirmenin önceki değerli işlerin kaybedilmesine neden olacağından sık sık bu durumdan korkulur, ancak Dinamik

Sistem Geliştirme Yöntemi küçük artımlı yineleme ile yazılım geliştirmeyi önerdiğinden, toplam iş kaybı oldukça sınırlıdır.

7. "Gereksinimler Yüksek Seviyede Belirlenir"

Geliştirme süreci boyunca hangi gereksinimlerin değişebileceğini sınırlamak için bazı yüksek seviyeli gereksinimler belirlenmelidir ve bu gereksinimler ölçüsünde değişime izin verilmelidir.

8. "Test Yaşam Döngüsü Boyunca Entegre Edilir"

Dinamik Sistem Geliştirme Yöntemi sistemin en başından itibaren test edilmesini destekler. Gereksinim ile ilgili yapılan görüşmeleri bile bir kontrol grubuyla test etmeyi önerir.

9. "İş birliği ve Ortak Çalışma Yaklaşımı"

Bir proje süresince ekip ve müşteri arasında iş birliği olmalıdır. Her ekip üyesi birbirine güvenmeli ve birlik içinde çalışarak projeyi en iyi şekilde yürütmek için elinden geleni yapmalıdır.

3.5.2. Dinamik Sistem Geliştirme Yöntemi Aşamaları

Dinamik Sistem Geliştirme Yöntemi süreci 7 aşamadan oluşur. Aşamalardan bazıları projeye göre göz ardı edilebilir ya da uyarlanabilir.[115]

[115]I. Zafar, A. K. Nazir ve M. Abbas, 2017, s. 1-6

1. **Ön Proje**

Ekip proje önerilerinde bulunur ve aday projeler arasından bir tanesi seçilir. Olası projeler değerlendirilip incelenir. Bu aşama bir projenin yapılıp yapılmaması gerektiğinin belirlendiği aşamadır.

2. **Fizibilite Çalışması**

Problemin ne olduğunun anlaşılmasının, projeye dair maliyet hesaplamalarının, bir iş problemini çözmek için gereken teknik fizibilitenin değerlendirilmesinin yapıldığı aşamadır.

3. **İş Çalışması**

Gereksinimlerin kararlaştırıldığı ve probleme dair alt işlerin yapıldığı aşamadır.

4. **İşlevsel Ürün Döngüsü**

Müşteri için işlevsel halde olan bir ürünün ortaya çıkarıldığı aşamadır. Bu aşamanın amacı müşteriden alınan geribildirim ile daha iyi bir geliştirme yapabilmektir.

5. **Tasarım ve Yapı Döngüsü**

İşlevsel ürün döngüsünde ortaya çıkarılan prototipler tatmin edici bir standarda ulaşana kadar hem tasarım hem de kod açısından daha da geliştirilir. Bu aşamanın ürünü, uygulamaya hazır test edilmiş bir sistemdir.

6. **Uygulama**

Ürünün son kullanıcının kullanacağı ortama alındığı aşamadır.

7. **Proje Sonrası**

Ortaya çıkarılan projenin kullanıcıya verildikten sonra değerlendirildiği aşamadır. Projenin nasıl bir performans gösterdiği, herhangi bir gelişme yapılıp yapılmaması gerektiği dikkate alınır.

3.5.3. Dinamik Sistem Geliştirme Yöntemi Rolleri

DSDM kullanılırken genelde bir ya da iki ekip ile yazılım geliştirilir. İkinci ekip yazılım geliştirme ekibinin döngülerde ürettiği ürünleri test eden bir ekip olarak çalışabilir. Projenin büyüklüğüne göre ekip üye sayısı 150'ye kadar çıkabilmektedir. Dinamik Sistem Geliştirme Yönteminde ekip içerisinde belirlenmiş birtakım roller vardır ve her rol bir sorumluluk üstlenmiştir. Rollerin sayısı 15'e kadar çıkabilmektedir. Başlıca birkaç rol şunlardır:

1. **Geliştirici:** Yazılım geliştirme ile sorumlu ekip üyeleridir. Geliştirme ekibinde analizci, test elemanı, tasarımcı ve programcı yer alır.

2. **Teknik Koordinatör:** Projenin iş ve teknik yanlarıyla ilgilenir. Sistem mimarisini tasarlar ve sistemin teknik kalitesiyle ilgilenir.

3. **Kullanıcı Elçisi:** Geliştirilen sistemi kullanacak olan kullanıcılardan biridir. Kullanıcı tarafının isteklerini geliştirme ekibine, sistemin ilerleyişini kullanıcıya iletme yeteneğine sahip olmalıdır.

4. **Danışman Kullanıcı:** Kullanıcı Elçisi tek başına bütün kullanıcıları temsil edemez bu yüzden ek olarak Danışman Kullanıcı rolü de tanımlanmıştır. Danışman Kullanıcı projeyle ilgili spesifik bilgiler vererek katılım sağlayabilir.

5. **Vizyoner:** Projenin doğru yolda ilerlemesi için rehberlik eder.

6. **Yönetici Sponsor:** Projenin geliştirilmesi için gereken desteği ve kaynağı bulan roldür. Bu rol karar verme konusunda nihai bir güce sahiptir.

7. **Proje Müdürü:** Kullanıcı ya da ekip tarafından biri olabilir. Projenin ilerleyişini denetler.

3.5.4. Dinamik Sistem Geliştirme Yöntemi Temel Araçları ve Teknikleri

Dinamik Yöntem geliştirme ekibine yazılım geliştirirken kullanacakları araçlara ve tekniklere karar verme konusunda özgürlük sağlar. Bunlardan en yaygın kullanılan araçlar ve teknikler şunlardır:[116]

1. **Zaman kutusu**

Zaman kutusu, belirli bir görev dizisinin gerçekleştirilmesi gereken, genellikle 6 haftayı geçmeyen bir aralıktır. Zaman kutularında birçok görev vardır ve her zaman kutusunun sonunda bir ürün teslim edilmelidir.

2. **Moscow Önceliklendirmesi**

Yazılım projeleri belli bir zaman ve maliyet kısıtlaması içinde koştuğu için projede neyin öncelikli olduğunu anlamak gerekir. Kullanıcı proje ilerleyişi sırasında bazı hususlar hakkında fikrini değiştirebilir. Geliştirici ekibi buna hazır olmalı ve değişikliklere adapte olabilmelidir. Moscow Önceliklendirmesi gereksinimleri şu şekilde sınırlandırır:

- Mutlaka olmalı: Bu başlık altında tanımlanan bütün özellikler üründe bulunmalıdır.
- Olmalı: Bu başlık altındaki özellikler proje için önemli olan işlerdir.
- Olabilir: Sistemi geliştiren özelliklerdir ama bir başka zaman kutusunda ele alınabilirler.
- Olması isteniyor: Genelde belli bir kesim tarafından istenen, pek önemli sayılmayan özelliklerdir.

[116]B. Voigt, 2004, s. 10-12

3. **Prototipleme**

Dinamik Sistem Geliştirme Yönteminin özelliklerinden biri yinelemeli ve artımlı bir şekilde yazılım geliştirmeyi desteklemesidir. Her döngü sonunda bir ürün\prototip ortaya çıkar ve kullanıcıya sunulur. Prototipler yazılım geliştirme sürecinin başlarında kullanıcıdan geribildirim alınarak olası hataları tespit etmek, geliştirmenin gidişatına katkıda bulunmak için kullanılır. Dinamik Sistem Geliştirme Yöntemi prototip tiplerini şu şekilde ayırır:

- İş prototipi: Geliştirilen sistemin gözlemlenmesine müsaade eder.
- Kullanım Prototipi: Kullanıcı ara yüzü kontrol edilir.
- Performans/Kapasite Prototipi: Çözümün performansı karşılayacağına ya da projenin hacmini idame edebileceğine emin olur.
- Yetenek/Teknik Prototipi: Muhtemel seçenekleri değerlendirir.

4. **Kolaylaştırılmış Atölye**

Ekip üyeleri ve kullanıcıların ürünle ilgili çalışma yapmasını sağlayan bir atölye çalışmasıdır. Kolaylaştırılmış Atölyenin amacı işbirlikçi çalışmayı teşvik etmek ve insanların grup olarak bir araya gelip daha etkili iletişim kurması ve daha yaratıcı çözümler üretmesi için daha kısa bir zaman diliminde yüksek kaliteli ekip bazlı kararların alınmasını sağlamaktır.

3.5.5. Dinamik Sistem Geliştirme Yöntemi Avantajları

- Çevik prensipleri benimsenerek yazılım geliştirme yapıldığı için müşteri memnuniyeti ön plandadır.
- Yinelemeli ve artımlı şekilde ürün ortaya çıkarıp müşteriye sunarak müşteriyi sürece dahil eder. Müşteriden gelen geri bildirimler sayesinde daha sağlıklı bir proje geliştirilir.
- Diğer yöntemlerle bir arada kullanılabilir.
- Keskin prensipleri vardır ve bu sayede proje yönetimi, risk kontrolü ve geliştirme teknikleri konularında rehberlik sağlar.
- Kaliteyi ön planda tuttuğu için müşteriye yüksek kalitede bir ürün sunabilir.

- Projeler belli bir zaman aralığı içinde oluşturulur.

3.5.6. Dinamik Sistem Geliştirme Yöntemi Dezavantajları

- Bu yöntemde proje içerisinde bir sürü rol tanımlanmıştır fakat tanımlanan bu çok sayıda rol, geliştirme sürecinde yönetim sorunları yaratabilir.
- Projenin kritikliği ön planda değildir.
- Diğerlerine kıyasla yeni bir yöntem olarak kabul edildiği için henüz pek yaygın değildir.

DÖRDÜNCÜ BÖLÜM

BÜTÜNLEŞİK YETENEK OLGUNLUK MODELİ

Türkçe olarak 'Bütünleşik Yetenek Olgunluk Modeli' İngilizce haliyle "Capability Maturity Model Integration" olarak belirtilen CMMI, en iyi uygulamaların bir araya getirilmesiyle kurumların süreçlerini iyileştirmeyi ve süreçlerin olgunlaştırılmasına yardımcı olmayı hedefleyen bir süreç modelidir.[117] SEI (Software Engineering Institute), endüstri ve kamu kurumlarından oluşmuş katılımcılar tarafından geliştirilmiştir. İlk olarak yalnızca yazılım üzerine, SW-CMM olarak 1991 yılında yayınlanan model, 2002 yılında sektör bağımsız bir şekil alarak "CMMI" olmuş, 2006 yılında ise v1.2 olan son sürümüne ulaşmıştır. CMMI v1.2 ile beraber, belirli bir ilgi alanında süreç gelişimine yönelik faaliyetlerde bulunmak adına CMMI bileşenleri bir araya toplanarak "Takımyıldızı" adı verilmiştir. Şekil 4.1.1'de görülebilen bu takımyıldızları şu şekilde ifade edilir:

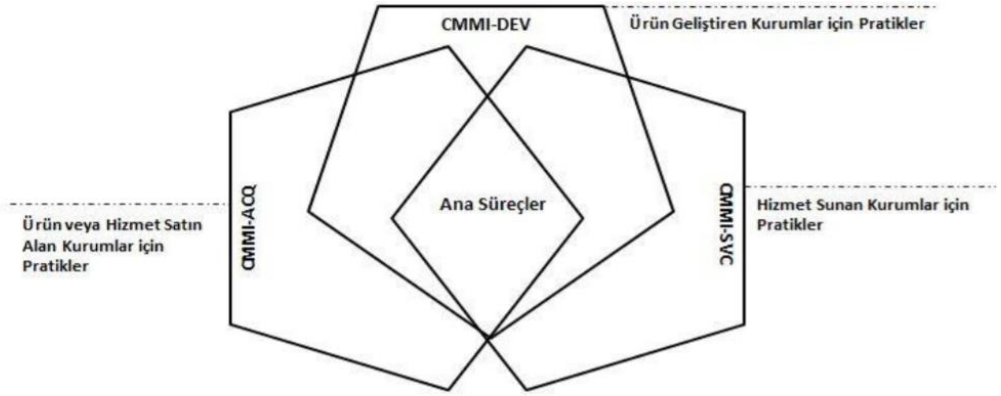
- CMMI-DEV (Development): Ürün geliştiren, proje yapan ve bu proje sonucu yeni bir ürün veya hizmet ortaya koyan kurumlar için uygundur.

- CMMI-SVC (Services): Geliştirilmesi tamamlanmış bir ürün veya kullanıcıya hizmet sunan kurumlar için uygundur. Ürünün işletilmesi, bakımı ya da yürütülmesi ile ilgili hizmetlerde kullanılabilmektedir.

- CMMI-ACQ (Acquisition): Satın alma süreçleri ile ilgilenen kurumlar için yararlıdır. CMMI-ACQ satın alma süreçlerinin olgunluk ölçümünün yapılmasını ve iyileştirilmesini amaçlar. [118]

[117] Ayça Alparslan Mollaoğlu ve Sönmeztürk Bolatan, Gülin İdil, "Yazılım Süreçlerinde Kalite Ve Yetenek Olgunluk Modeli." *Organizasyon ve Yönetim Bilimleri Dergisi,* Sayı: 12, No:1, 2020, s. 82.

[118] Halil İbrahim Saraçoğlu, "*CMMI Modeli Ve Çevik Yazılım Geliştirme Yöntemlerinin Entegrasyonu*, Erciyes Üniversitesi, Yüksek Lisans Tezi, 2009, s. 10.



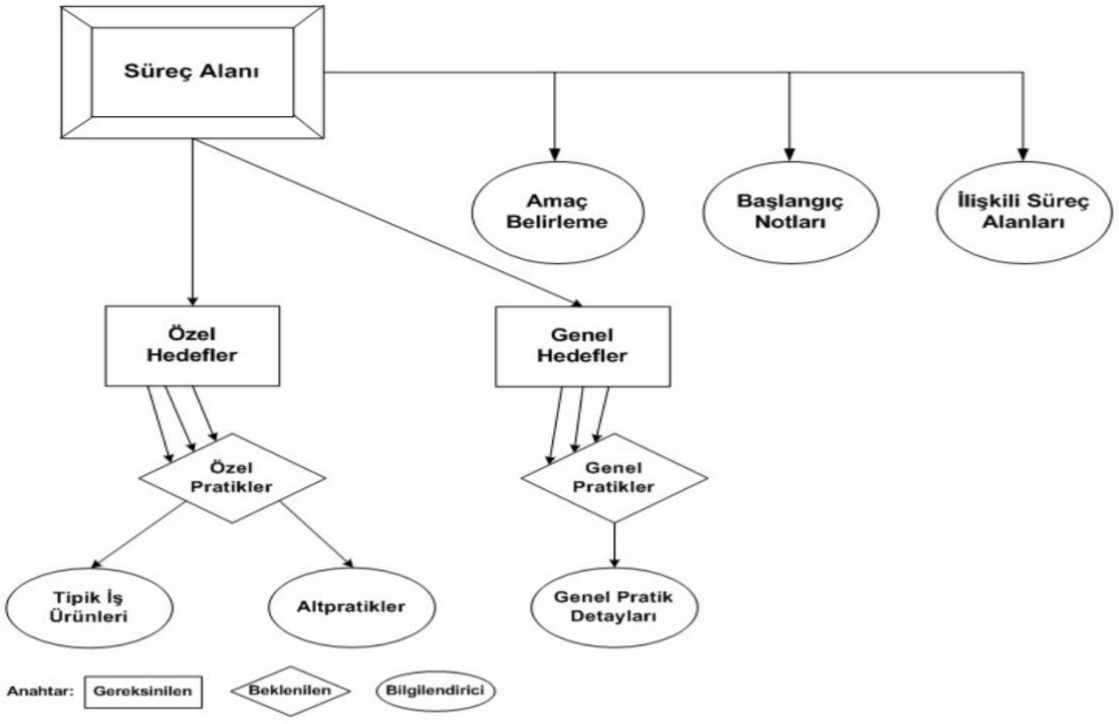
Şekil 4.1 CMMI Takımyıldızı[119]

4.1.CMMI Yapısı

Bütünleşik Yetenek Olgunluk Modeli kurumların ortaya çıkardıkları ürünlerin ve verdiği hizmetlerin kullanıcı beklentilerini en iyi şekilde karşılaması amacı ile aktivitelere odaklanmaları için bir yol haritası sunar. Bütünleşik Yetenek Olgunluk Modeli bir süreç veya standart değildir, kurum süreçlerinin olgunluk seviyesini belirleyen bir modeldir.[120] Bu modelde temel yapıtaşı Süreç Alanı'dır. Gereksinilen Bileşenler, Beklenilen ve Bilgilendirici Bileşenler olmak üzere üç başlıkta ele alınır. Gereksinilen Bileşenler bir organizasyonun, bir süreç alanında başarılı olması için üstesinden gelmesi gereken bileşenleri tanımlar. CMMI'daki gereksinilen bileşenler özel hedefler ve genel hedeflerden oluşur. Beklenilen Bileşenler bir organizasyonun gereksinilen bileşenlerde başarılı olması için yapması gerekenleri tanımlar. Beklenilen Bileşenler özel pratikler ve genel pratikler içerir. Bilgilendirici Bileşenler organizasyonun beklenilen ve gereksinilen bileşenlere ulaşmasıyla ilgili detay sağlayan bileşenlerdir. Tipik iş ürünleri, alt pratikler ve amaç genel pratik detayları Bilgilendirici Bileşenlerdir.[121] Gereksinilen bileşenlerin mutlaka karşılanması gerekmektedir. Beklenilen bileşenler direkt karşılanabilir ya da alternatif bir çözüm de

[119] A. Alparslan Mollaoğlu ve G. İdil Sönmeztürk Bolatan, 2020, s. 83.
[120] A.g.m., s. 82.
[121] Orhan KALAYCI, CMMI: Yöneticiler için Doğru Sorular, 2007, s: 16.

uygulanarak karşılanabilir. Bilgilendirici bileşenin ise direkt karşılanmasına gerek yoktur, ilgili süreç alanında referans olarak kullanılabilir. CMMI model yapısı şekilde görülebilir.



Şekil 4.1.1 CMMI Yapısı

Bu modelde 22 adet süreç alanı bulunur ve bu süreç alanları şu şekilde ifade edilir:[122]

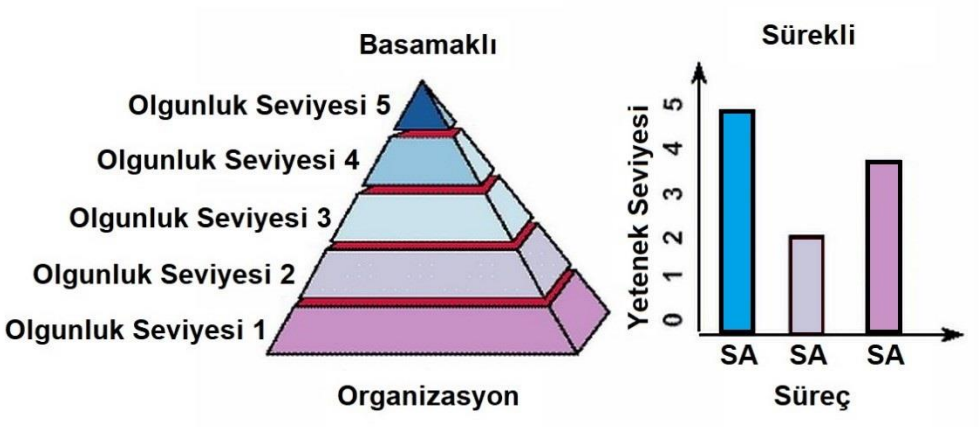
- Nedensel Analiz Çözümleme (Causal Analysis and Resolution - CAR)
- Konfigürasyon Yönetimi (Configuration Management - CM)
- Kara Analizi ve Çözümleme (Decision Analysis and Resolution - DAR)
- Tümleşik Proje Yönetimi (Integrated Project Management - IPM)
- Ölçüm ve Analiz (Measurement and Analysis - MA)
- Organizasyonel Yenilik ve Yayılma (Organizational Innovation and Deployment - OID)
- Organizasyonel Süreç Tanımlama (Organizational Process Definition – OPD)

[122] CMMI Product Team, "CMMI for Development, version 1.2.", 2006, s. 18.

- Organizasyonel Süreç Odaklama (Organizational Process Focus - OPF)
- Organizasyonel Süreç Performansı (Organizational Process Performance - OPP)
- Organizasyonel Eğitim (Organizational Training - OT)
- Ürün Entegrasyonu (Product Integration - PI)
- Proje İzleme ve Kontrol (Project Monitoring and Control - PMC)
- Proje Planlama (Project Planning - PP)
- Süreç ve Ürün Kalite Güvencesi (Process and Product Quality Assurance - PPQA)
- Nicel Proje Yönetimi (Quantitative Project Management - QPM)
- Gereksinimlerin Gelişimi (Requirements Development - RD)
- Gereksinimlerin Yönetimi (Requirements Management - REQM)
- Risk Yönetimi (Risk Management - RSKM)
- Tedarikçi Sözleşme Yönetimi (Supplier Agreement Management - SAM)
- Teknik Çözüm (Technical Solution - TS)
- Onaylama (Validation - VAL)
- Doğrulama (Verification - VER)

4.2.CMMI Model Gösterimleri

CMMI Modeli Sürekli Gösterim ve Basamaklı Gösterim olarak iki şekilde gösterilir. Her iki gösterimde içerik aynıdır fakat verinin gösterilmesi ve düzenlenmesi farklılık gösterir. Sürekli Gösterim süreçlerin yetenek düzeylerini tanımlarken, basamaklı gösterim ise kuruluşun olgunluk düzeyini tanımlar ve her ikisi de bir organizasyonun süreçlerinin ne kadar iyi geliştirebildiğinin ölçülmesine olanak sağlar. Şekil 4.1.2.1'de CMMI Gösterim Çeşitleri görülebilir.



Şekil 4.2.1 CMMI Gösterim Çeşitleri

4.2.1.1.Sürekli Gösterim

Bu gösterim süreçleri iyileştirmede daha esnek bir bakış açısı öne sürer. Kurumların hedefleri çerçevesinde belirledikleri sürece veya süreçlere odaklanarak iyileştirmeler yapılmasına ve faklı süreçlerin çeşitli oranlarda iyileştirilmesine imkân verir. Sürekli Gösterimde her süreç alanı için tamamlanmamış süreçten iyileşen sürecine kadar olan iyileştirmeyi ölçmek için çeşitli yetenek seviyeleri kullanılır. Bu yetenek seviyeleri şu şekilde ifade edilir: [123]

Seviye 0 – Tamamlanmamış (Incomplete): Bu seviye gerçekleştirilmeyen veya kısmen gerçekleştirilmiş bir süreçtir. Süreç alanının belirli hedeflerinden bir veya daha fazlası karşılanmamıştır ve kısmen gerçekleştirilmiş bir hedefi kurumsallaştırmak için bir neden olmadığından genel hedefler uygulanmaz.

Seviye 1 – Gerçekleştirilen (Performed): Bu seviye bir süreç alanının belirli hedeflerini karşılayan bir süreç olarak ele alınır. Tamamlanmamış ve gerçekleştirilen

[123]Shahid Mahmood, "*Capability Maturity Model Integration CMMI*", Virtual University of Pakistan, 2015, s. 7-8.

süreçler arasındaki fark, gerçekleştirilen seviyede süreç alanının tüm özel hedeflerini karşılamasıdır. Gerçekleştirilen süreç, ortaya bir ürün çıkarmak için süreç alanının belirli hedeflerine ulaşılmasını destekler ve sağlar.

Seviye 2 – Yönetilen (Managed): Bu seviyede yönetilecek olan amaç uygun bir şekilde planlanır, yürütülür, gözden geçirilir, kontrol edilir ve süreç tanımına uygunluk açısından değerlendirilir yani bir süreç planlanır ve sürecin performansı plana göre yönetilir.

Seviye 3 – Tanımlı (Defined): Bu seviyede yönetilen süreç, organizasyonun politikalarına göre organizasyonun standart süreçlerine uyarlanır. Organizasyondaki diğer süreç uygulamalarına ve iyileştirmelerine katkıda bulunur.

Seviye 4 – Nicel Yönetilen (Quantitatively Managed): Bu seviyenin temel amacı, projenin daha önceden belirlenmiş kalite ve süreç performans hedeflerine ulaşmaktır. Bunun belirlenmesinde sayısal teknikler kullanılır, kalite ve süreç performansı istatistiksel olarak hesaplanır ve projenin yaşam döngüsü boyunca yönetilir.

Seviye 5 – İyileşen (Optimizing): Bu seviyede artan, yenilikçi süreç ve teknolojik yöntemler yoluyla süreç performansını sürekli iyileştirmeye odaklanılır. Organizasyon iş hedeflerine ve performans gereksinimlerine ilişkin nicel bir anlayışa dayalı olarak süreçlerini sürekli olarak geliştirir. Yapılan süreç iyileştirmeleri organizasyonun iş hedeflerinden türetilen kuruluşun kalite ve süreç performans hedeflerini destekler. Süreçler sürekli revize edilir.

4.2.1.2.Basamaklı Gösterim

Basamaklı gösterim her aşamada sistematik bir yol ortaya koyarak süreç iyileştirmesinin gerçekleştirilmesini hedefler ve organizasyon seviyesinde bütün organizasyonu kapsayacak şekilde bir bakış açısı sunar. Olgunluk seviyesi adı altında beş süreç alanından oluşur. Her seviyenin başarılması, bir sonraki aşamanın temeli olarak yeterli bir süreç altyapısının oluşturulmasını sağlar. Bu olgunluk seviyeleri şu şekildedir:[124]

Seviye 1 – Başlangıç (Initial): Bu seviyede süreçler geçici ve karmaşıktır. Organizasyon istikrarlı bir ortam sağlayamaz ve süreçler kriz anında terk edilebilir. Başarı kişisel becerilere bağlıdır.

Seviye 2 – Yönetilen (Managed): Süreçlerin planlamasının, gerçekleştirilmesinin, ölçülmesi ve kontrol edilmesinin takibi yapılır. Bu seviyede süreçler ve hizmetler planlanan dokümanlara göre yönetilmektedir.

Seviye 3 – Tanımlı (Defined): Bu seviyede süreçler iyi tanımlanmış ve anlaşılmıştır. Prosedürler, araçlar ve standartlar olarak detaylıca tanımlanmıştır. Süreçler Seviye 2'ye göre daha detaylı ve kapsamlı ele alınır.

Seviye 4 – Nicel Yönetilen (Quantitatively Managed): Bu seviyede süreçler nicel ölçümler ile kontrol edilir. Kalite ve süreç performansları için nicel hedefler belirlenir ve bu hedefler süreçleri yönetmek için kullanılır. Bir organizasyon için nicel hedef, müşterinin, son kullanıcıların, organizasyonun ve süreç ihtiyaçlarına ve görüşlerine göre belirlenir.

[124] S. Mahmood, 2015, s. 4-5.

Seviye 5 – İyileşen (Optimizing) : Bu seviyede süreçlere ve süreç iyileştirmeye odaklanılır. Süreçlerde meydana gelen ortak sorunlar araştırılır ve çözüm aranır, süreç performansı nicel ve teknolojik yöntemler kullanılarak sürekli iyileştirilir. Uygulanan süreç iyileştirme faaliyetlerinin etkisi nicel olarak ölçülür ve süreç iyileştirme hedeflerine göre değerlendirilir. Yani bu seviyenin amacı süreçlerdeki ortak sorunları bulmak, süreci nicel olarak inceleyip süreç performansını iyileştirmek ve önceden tanımlanmış süreç iyileştirme hedefine ulaşmaktır.

4.3.Türkiye'de CMMI Kullanımı

Yazılım geliştiren kuruluşların performanslarını iyileştirmek amacıyla tasarlanan CMMI çalışmaları ülkemizde gün geçtikçe artmaktadır. Bu hususta Savunma Sanayii Müsteşarlığının askeri yazılım alımlarında ön gereklilik olarak CMMI Seviye 3 sertifikasyon şartı arayacağını belirtmesi etkili olmuştur. Ülkemize ait bazı CMMI bilgileri şu şekildedir: [125]

- İlk CMMI sertifikasını (CMMI Level 3) Milsoft 2002 yılında almıştır.
- İlk CMMI Level 5 sertifikasını 2007 yılında yine Milsoft firması almıştır.
- Türkiye'de 2011 yılı itibari ile toplam 20 CMMI değerlendirilmesi yapılmıştır.
- 2011 yılı itibari ile toplam 23 CMMI sertifikası alınmıştır. 20 CMMI Level 3 sertifikası, 3 CMMI Level 5 sertifikası alınmıştır
- Türkiye'de yazılım şirketleri arasında savunma, finans, telekom ve tıp sektörüne yönelik yazılım üreten şirketlerde CMMI çalışmaları yapılmaktadır.
- Finans sektöründe çeşitli çalışmalar yapılmaktadır. Fakat çalışmaların asıl amacı süreç iyileştirmek olduğu için henüz herhangi bir finans kuruluşu CMMI sertifikası sahibi değildir.
- 2008 yılında TÜBİTAK-UEKAE CMMI Level 3 sertifikası almıştır.
- 2008 yılında TÜBİTAK-BTE CMMI Level 3 sertifikası almıştır.

[125]Orhan Tanrıkulu, "*CMMI'ın Bir Türk Bankacılık Sektöründe Uygulanması Ve Analizi*", Haliç Üniversitesi, Yüksek Lisans Tezi, 2011, s. 10

- 2011 yılında SOFTECH Level 3 sertifikası almıştır.
- 2017 yılında TÜBİTAK CMMI Level 5 sertifikası almıştır.[126]

[126] Salih Soylu (2017), "Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği Ve Çevre Ve Şehircilik Bakanlığı İçin Öneriler", Uzmanlık Tezi, 2017, s. 73.

BEŞİNCİ BÖLÜM

YAZILIMLARIN BAŞARISIZ OLMA SEBEPLERİ

Bir yazılım ürünü ortaya çıkarılacağı zaman yazılıma dair bilgiler derlenip toplanır, hazırlıklar yapılır, uygun bir yazılım geliştirme yaşam döngüsü modeli seçilir ve yazılım geliştirilir. Her yazılım en iyi şekilde sunulmak istenir, her yazılımcı ürettiği emek verdiği projenin iyi bir şekilde sonlanmasını ister fakat her geliştirilen yazılım sonuca ulaşamayabilir. Bazen geliştirme ekibi problemleri sebebiyle bazen kullanıcı kaynaklı problemler yüzünden bazen yazılımın güçlüğü nedeniyle bazen bütçe kısıtı bazen de zaman sınırlamasından kaynaklı olarak yazılım ürünleri istenilen şekilde sonuç vermez. Bir yazılımın başarısız olması için çeşitli sebepler vardır. Başlıca şu durumları sebep olarak ele alabiliriz: [127]

a. Gereksinimlerin Net ve Doğru Belirlenmemesi

Geliştirilmek istenen yazılım ürününün gereksinimlerini belirlemek yazılım ürününü oluşturmanın ilk basamağıdır. Bazen bir projenin amacı, bir projenin tanımlama aşamasında toplanan yetersiz gereksinim bilgisi nedeniyle yalnızca kısmen net olabilir. Birçok projenin yüksek seviyeli, belirsiz ve genellikle yardımcı olmayan gereksinimleri vardır. Bu durum yazılım geliştiricilerin kullanıcıdan hiçbir girdiye sahip olmamasına ve projenin geliştirildiği iş hakkında gerçek bir bilgiye sahip olmadan gerekli olduğuna inandıkları şeyi oluşturmalarına yol açar. Haliyle kullanıcının talep ettiğinden farklı bir ürün ortaya çıkmış olur ve sistem teslim edildiğinde kullanıcı elindeki ürünün ihtiyaç duyduğu şeyi yapmadığını beyan edebilir. Bu yüzden gereksinimler oldukça tam belirlenmelidir. Bir proje için açık ve net gereksinimlerin tanımlanması zaman alabilir ve çok fazla iletişim gerektirebilir bu yüzden kullanıcı ne istediğini bilmelidir. Ancak bazen kullanıcılar gerçekte neye ihtiyaç duyduklarını açıklayacak deneyime sahip olmadığı için hedefler ve amaçlar net

[127]Rupinder Kaur ve Sengupta, Jyotsna. "Software Process Models and Analysis on Failure of Software Development Projects", *International Journal of Scientific & Engineering Research,* Sayı: 2, No: 2, 2013, s. 3-4.

olmayabilir. Kullanıcı, projede neye ihtiyaç duyduğunu bilmeli ve bunu açıkça belirtebilmelidir. Bununla birlikte, kullanıcı BT uzmanı olmadığından, geliştirici yazılım mühendisliğindeki beceri ve deneyimleriyle kullanıcıdan gereksinimleri çıkarmalıdır, kullanıcının ne istediğini belirtme ve belirleme konusunda gerektiğinde yardımcı olmalıdır.

b. Kullanıcının Projeye Dâhil Olmaması

Araştırma şirketleri ve akademik kurumlar, yönetici desteğinin olmaması ve kullanıcı katılımının olmamasını bilişim projelerini yönetmede iki ana zorluk olarak ele almaktadır. Kullanıcı katılımının olmamasının birçok proje için önemli bir etmendir. Kullanıcının proje sürecine katılımının az ya da eksik olması projenin kullanıcı beklentilerini karşılayamamasına sebep olur.[128] Bu yüzden yazılım projesinin başarılı olmasının kriterlerinden biri, projenin başlangıcından itibaren ve geliştirmesi boyunca devamlı olarak kullanıcının sürece dâhil olmasıdır. Bu durum kullanıcının zamanını alır ve onlar tarafından çaba gerektirir fakat çoğunlukla yeni bir proje için zaman bulmak kullanıcılar için öncelik listesinde yüksek bir sırada yer almadığı için genellikle göz ardı edilir. Hâlbuki kullanıcının projeyi sürekli desteklemesi, geri bildirimlerde bulunması, projeye katılması gerekir. Geliştirici, gereksinimleri ortaya çıkarmaya ve projenin tüm işlevlerini sunmaya yardımcı olduğu için kullanıcıyı geliştirme sürecine dâhil etmelidir.

c. Ekip Boyutu

Uygun ekip boyutu yazılım geliştirme projelerinde oldukça önemli bir faktördür. Genel olarak üç farklı ekip boyutları vardır: küçük ölçekli projeler için oluşturulan 10 ya da daha az kişiden oluşan küçük çaplı bir ekip, orta ölçekli projeler için oluşturulan 11 ila 25 kişiden oluşan orta çaplı bir ekip, büyük ölçekli projeler için oluşturulan 26 ya da daha fazla kişiden oluşan büyük çaplı bir ekip. Küçük bir ekip grubu iyi bir

[128]Ardhendu Mandal ve Pal, S. "Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models", *International Journal of Computer Sciences and Engineering,* Sayı: 3, 2015, s. 118-126.

iletişime sahip olmakla birlikte orta ve büyük çaplı ekiplerden daha esnek olma eğilimine sahiptir. Toplantı ayarlamak ve anında geribildirim almak kolaydır. Bu da projeler için oldukça önemli bir durumdur zira bazı projelerin başarısız olma sebeplerinden birisi ekip içerisindeki eksik iletişimdir.[129]

d. Zaman Aralığı

Yazılım geliştirmede zaman aralığı önemlidir, projenin belirli bir zaman çizelgesine göre teslim edilmesi faydalıdır. Bu yüzden projeye verilen zaman iyi düşünülmelidir. Görevde geçen süre, görevin kesintisiz olarak tamamlanması için gereken zaman aralığıdır, proje süresi ise görevin kesintiler dâhil olmak üzere fiilen tamamlanması için geçen zamandır. Projenin başlangıç ve bitiş süresi arasında geçen süreyi tahmin etmek için görevde geçen süreyi kullanmak, proje yöneticileri tarafından yapılan yaygın hatalardan biridir. Bir proje için gereğinden uzun bir zaman çizelgesi ya da projenin tamamlanmasından öncesine denk gelen kısalıkta bir zaman çizelgesi projenin başarısız olmasına neden olur. Gerçekçi bir şekilde projenin süresi belirlenmeli ve proje ona göre yönetilmelidir.

e. Kontrol

Bir yazılım sistemi kurulurken gereksinimlerde bir değişiklik beklememek gerçekçi bir yaklaşım değildir. Proje geliştirilirken ortam faktörleri ve müşteri gereksinimleri değişmeye devam edebileceğinden, geliştirici sistemi oluştururken bileşen odaklı bir yaklaşım benimsemelidir. Yeni gereksinimler veya modifikasyonlar, bu bileşenler mevcut geliştirme süreciyle eşleşene kadar ayrı ayrı ele alınabilir. Ancak kontrolsüz değişiklikler, geliştirilmekte olan sisteme zarar verir ve birçok projenin başarısızlığına neden olur.

[129] Gonçalo Costa, "Spring 6-10-2017 A Survey Of Faıilures In The Software Development Process", *ECIS 2017 Proceedings,* 2017, s. 2450-2451.

f. Test

Testin esas amacı yazılım hatalarını tespit etmek ve böylece kusurların keşfedilip düzeltilebilmesidir. Geliştirici, yazılım geliştirme sırasında yazılım ürününü test eder, ancak sonunda kullanıcının, oluşturulan sistemin kendi iş gereksinimlerini karşılayıp karşılamadığını görmek için kabul testi yapması gerekir. Kabul testi çoğu zaman sistem canlı ortamda devreye girmeden önce birçok arızayı yakalayamaz; bunun nedeni ise planlanmadan yapılan testler, testin amacını bilmeyen kullanıcı ve proje geciktiği için testi yapmak için yetersiz kalan zaman olabilir.

g. Kötü Kalite Yönetimi

İyi bir yazılımda aranan önemli özelliklerden biri kaliteli olmasıdır. Yazılım geliştirme süreci boyunca çeşitli şekillerde kalite yönetimi yapılır. Proje bitiminde elde edilen ürünün arzu edildiği gibi olması için periyodik kalite değerlendirmeleri yapılması, proje geliştirildiği sırada herhangi bir problem ortaya çıkması durumunda probleme uygun bir şekilde müdahale edilmesi ve problem olmaması için önlemler alınması zorunludur. Hata giderme faaliyetleri arasında gereksinimlerin gözden geçirilmesi, tasarımın gözden geçirilmesi, kodlamanın gözden geçirilmesi ve çeşitli şekillerde testler yapılması bulunur. Proje geliştirme süreci boyunca iyi kalitede bir ürün geliştirilmeye önem verilmediği takdirde ortaya kullanıcının talep ettiğinden uzak bir sonuç çıkması oldukça muhtemeldir.

BULGULAR VE ANALİZ

Bu tez çalışması boyunca yazılım proje yönetimi açısından oldukça önemli bir yere sahip olan Yazılım Geliştirme Yaşam Döngüsü kavramı incelenmiştir. Kavramın, yazılım geliştirme modelleri ve yöntemleri ile olan ilişkisi değerlendirilmiştir.

Tez çalışmasının ikinci ve üçüncü bölümlerinde herhangi bir yazılım ürünü geliştirilirken uygulanabilecek yazılım geliştirme yaşam döngüsü modelleri ve yöntemleri araştırılmıştır. Karşılaştırmalı analiz yöntemi ile incelenen yazılım geliştirme yaşam döngüsü modelleri ve yöntemlerinin avantajları dezavantajları belirlenmiştir.

Tez çalışmasının dördüncü bölümünde yazılım projelerinin başarısız olma sebepleri incelenmiş, bu çalışmanın içerikleri, yapılma yöntemleri ve etkinlikleri hakkında literatür taraması yapılmıştır.

Her yazılım geliştirme yaşam döngüsü modeli ve yönteminin kendine has özellikleri ve ortak özellikleri bulunmaktadır. Bir yazılım geliştirilirken dikkat edilmesi gereken ve yazılımın başarısız olmasını engellemek için önemsenmesi gereken faktörler vardır. Dolayısıyla bir yazılım ürününü geliştirmek için yazılım geliştirme yaşam döngüsü modeli veya yöntemi seçilirken bu faktörler göz önüne alınmalıdır. Tezin önceki başlıklarından elde edilen bilgiler sonucunda şu çıkarımlar yapılmıştır:

1. **Geliştirme Yaklaşımı ve Esneklik:**

Şelale Modeli doğrusal bir geliştirme yaklaşımı benimsemektedir. Katı bir yapısı vardır. Yaşam döngüsü aşamaları doğrusal olarak ilerler ve daha önceki aşamalara dönmek çok zordur.

V-Şekli Modeli Şelale Modelinden türemiştir fakat yapısal olarak farklılıklar içermektedir. Bir yere kadar doğrusal olarak ilerler fakat daha sonra bükülerek V şeklini oluşturur. Değişikliklere Şelale Modelinden daha esnek bir şekilde yanıt verir.

Helezonik Model ve Prototip Modeli evrimsel bir geliştirme yaklaşımı, DSDM yinelemeli bir yazılım geliştirme yaklaşımı benimsemiştir. Helezonik Model ve Prototip Modelinde SDLC içerisinde daha önceki aşamalara dönmek daha kolaydır bu sebeple esnek bir yapıları vardır. DSDM, Çevik Manifestoyu benimsediği için değişime her an açıktır bu yüzden çok esnek bir yapısı vardır.

Kanban, Kristal Yöntemler ve Artımlı Model artımlı bir yazılım geliştirme yaklaşımı benimsemiştir, projede küçük artımlar oluşturularak ilerlenir. Kanban ve Kristal Yöntemler Çevik Manifestoyu benimsedikleri için değişime her an açıktır, bu yüzden yapıları çok esnektir, Artımlı Model geliştirme süresince değişime izin verir fakat artım sırasında katı bir yapıya sahiptir bu yüzden bu diğer iki modelden daha az esnektir.

Scrum ve XP yinelemeli artımlı bir yapıya sahiptir. Yazılım geliştirme süresi boyunca artımlar oluşturup döngüyü yineleyerek ortaya son ürünü çıkarma amacı güder. Çevik Manifestoyu benimsedikleri için değişime her an açıktır bu yüzden yapıları çok esnektir.

2. Kullanıcı Katkısı

Şelale Modeli ve V-Şekli Modeli'nde kullanıcı gereksinim analizi sırasında ve test sırasında projeye dahil olur. Bu yüzden kullanıcı katılımı oldukça düşüktür.

Helezonik Model, Prototip Modeli ve Artımlı Model'de proje kullanıcıdan geri dönüş alarak ilerler. Proje sonunda ortaya çıkması öngörülen sistemin oluşturulabilmesi için her döngü sonucu ortaya çıkan çıktılar kullanıcıya sunulur, kullanıcıdan alınan geri bildirim ile proje şekillenir fakat kullanıcı her aşamaya dahil edilmez.

Scrum, XP, Kanban, Kristal Yöntemler ve DSDM Çevik Manifestoyu benimsedikleri için proje boyunca kullanıcıyı merkeze koyarak ilerler, kullanıcı katılımı yüksek ve önemlidir.

3. Ekip Boyutu

Şelale Modeli, V-Şekli Modeli, Helezonik Model, Artımlı Model ve Prototip Modeli ekip boyutunu büyük tutarak projeyi yürütür.

Kanban ekip boyutuna herhangi bir sınırlama getirmez, yazılım geliştireceklerin kararına bırakmıştır. Bu yüzden değişkendir. Kristal Yöntemlerde ise seçilen yönteme göre ekip boyutu değişir.

Scrum, XP ve DSDM çevikliği ön planda tuttuğu için projede en verimli sonucu alabilmek adına ekip boyutunun az olması gerektiğini savunur.

4. Maliyet

Şelale Modelinde proje son aşamaya gelene kadar ürün tam anlamıyla görülemez. Bu yüzden projede ne gibi değişiklikler olması gerektiği bilinmediği için maliyet tam olarak kestirilemez.

Artımlı Model artımlı bir şekilde ilerlediği için proje maliyeti diğerlerine oranla daha düşüktür.

Helezonik Model risk analizi, Prototip Modeli kullanıcının geribildirimleri üzerine yapılan değişiklikler, V-Şekli modeli ise yapısı esnek olmadığı için değişimlere zor tepki vermesi maliyeti arttıran faktörlerdir.

Scrum, XP, Kanban, Kristal Yöntemler ve DSDM kısa sürede çabuk bir şekilde sonuç vermeye odaklandığı için projede yapılacak değişiklikler maliyeti yükseltir.

5. Geliştirme Süresi

Şelale Modelinde bir aşama tamamlanmadan diğer aşamaya geçilmez. Dolayısıyla bir aşamada meydana gelen sorunlar gecikme oluşturabilir. Bu da proje geliştirme sürecini uzatır. Helezonik Model ve Prototip Modeli ortaya kullanıcının istediği şekilde bir ürün çıkana kadar proje geliştirme sürecine devam eder. Bu durum sürecin uzamasına sebep olan bir durumdur.

Artımlı Model her artım sonunda oluşturulan çıktıyı kullanıcıya iletir. Böylece uzun bir geliştirme süresinden kaçınır. V-Şekli Modeli Şelale Modeli gibi esnek olmayan bir yapıya sahiptir fakat doğrulama ve onaylama sayesinde geliştirme süreci Şelale Modelinden daha kısadır. Scrum'un geliştirme süreci Scrum Guide'da önerilmiştir.

Geliştirme süreci hangi Kristal Yöntemler seçildiğine göre değişiklik gösterir, DSDM'de ise geliştirme ekibine bırakılmıştır. XP ve Kanban kısa sürede projeyi bitirmeye odaklanarak geliştirme yapar.

6. Başarı Oranı

Şelale Modelinin yapısı çok katıdır ve değişimlere çok zor cevap verir. Bu yüzden başarı oranı düşüktür.

V-Şekli modelinde test planlaması erken aşamalarda yapıldığı için başarı oranı Şelale Modelinden yüksektir. Helezonik Modelde projenin başarı oranı büyük oranda risk analizine bağlıdır. Risk analizi iyi bir şekilde yapıldığı zaman başarı oranı yüksektir. Artımlı Model'de artımlı bir geliştirme yaklaşımı benimsendiği için çalışan ara ürünler üzerinden geliştirme yapılır. Bu yüzden başarı oranı yüksektir. Prototip Modelinde proje prototip oluşturulup kullanıcıdan geribildirim alarak yürütülür. Bu da kullanıcının isteği doğrultusunda bir ürün ortaya çıkmasına zemin hazırlar.

Scrum, XP, Kanban, Kristal Yöntemler ve DSDM kullanıcının isteklerini ön planda tutarak hızlı bir geliştirme ile ortaya çalışan ürünler çıkarmaya odaklandıkları için başarı ihtimalleri yüksektir.

Bahsedilen faktörlere dair SDLC modellerinin ve yöntemlerinin karşılaştırması aşağıdaki Tablo 1'de görülebilir.

Tablo 1 Yöntemlerin Karşılaştırılması

Yöntem Adı\Özellikler	Geliştirme Süresi	Geliştirme Yaklaşımı	Esneklik	Kullanıcı Katkısı	Ekip Boyutu <10	Başarı İhtimali	Maliyet
Şelale Modeli	Uzun	Doğrusal	Katı	Düşük	Hayır	Düşük	Düşük
V-Şekli Modeli	Orta	Doğrusal+Artımlı	Düşük	Düşük	Hayır	Orta	Yüksek
Helezonik Model	Uzun	Evrimsel	Esnek	Orta	Hayır	Yüksek	Yüksek
Prototip Modeli	Uzun	Evrimsel	Esnek	Orta	Hayır	Yüksek	Yüksek
Artımlı Model	Orta	Artımlı	Az Esnek	Orta	Hayır	Yüksek	Düşük
Scrum	Orta	Yinelemeli+Artımlı	Çok Esnek	Yüksek	Evet	Çok Yüksek	Çok Yüksek
XP	Kısa	Yinelemeli+Artımlı	Çok Esnek	Yüksek	Evet	Çok Yüksek	Çok Yüksek
Kanban	Kısa	Artımlı	Çok Esnek	Yüksek	Değişken	Çok Yüksek	Çok Yüksek
Kristal Yöntemler	Değişken	Artımlı	Çok Esnek	Yüksek	Değişken	Çok Yüksek	Çok Yüksek
DSDM	Değişken	Yinelemeli	Çok Esnek	Yüksek	Evet	Çok Yüksek	Çok Yüksek

SDLC model ve yöntemi olarak Şelale Modeli, V-Şekli Modeli, Artımlı Model, Prototip Modeli, Helezonik Model, Scrum, XP, Kanban, Kristal Yöntemler ve DSDM incelenmiş ve çeşitli kıyas yöntemleriyle kıyaslanarak yukarıdaki tablo elde edilmiştir. Bu çalışmanın yorumlanması Tartışma ve Yorum bölümünde yapılmıştır.

TARTIŞMA VE YORUM

Teknoloji günden güne oldukça hızlı bir şekilde ilerlemekte ve çeşitli alanlarda bütün dünyayı etkilemektedir. Doğal olarak dünya da değişmekte ve yazılımcılar bu değişen dünyaya ayak uydurmak için gelişmekte, yazılım ürünlerini ortaya çıkarma konusunda değişik yöntemler ortaya atmaktadır. Yıllar içerisinde farklılaşan ihtiyaçlar ve görüşler doğrultusunda değişik yöntemler keşfedilmiş ve yazılım geliştirilirken projelere uygulanmıştır.

Bir yazılım ürünü geliştirilirken o ürünü en kaliteli ve faydalı şekilde ortaya çıkarabilmek için hangi yöntemin seçilmesi ve kullanılması gerektiği, bu tez çalışmasının önceki bölümlerinde de bahsedildiği gibi yazılım geliştirme konusunda oldukça önemli husustur. Çünkü her biri çeşitli özelliklere sahip olup çeşitli ihtiyaçları karşılayan birden fazla yazılım geliştirme yöntemi bulunmaktadır. Yazılım geliştirme yöntemlerinin her birinin kendine has bakış açıları, aşamaları bulunmaktadır ve her bir aşama kendi içinde bir karmaşıklığa sahiptir. En iyi performansa sahip yazılım ürününün ortaya çıkabilmesi için tüm bu yaklaşımların gözden geçirilmesi, deneysel çalışmalar ile incelenmesi ve en iyi yönteme karar verilmesi gerekmektedir. Bu sebeple bu tez çalışmasında karşılaştırmalı analiz yöntemiyle yazılım geliştirme yaşam döngüsü sırasında kullanılabilecek yöntemler değerlendirilmiş ve ileride bir yazılım ürünü geliştirilmek istenildiğinde, istenilen şekilde çalışabilir bir yazılımın ortaya koyulması için yol göstermek, katkı sağlamak amaçlanmıştır.

Dijital teknolojilerin oldukça hızlı bir şekilde gelişmesi sadece ekonomik ve sosyal hayatı değil bunların yanı sıra yazılım geliştirme tanım ve kavramlarının değişmesinde de büyük bir etkide bulunmuştur. Kamu kurumlarının değişen teknolojilere ayak uydurması için ülkemizde çeşitli adımlar atılmıştır. Türkiye Cumhuriyeti Ulaştırma ve Altyapı Bakanlığı bilgi ve iletişim teknolojilerinin güvenliğine dair 2020-2023 yılları

arası Ulusal Siber Güvenlik Stratejisi ve Eylem Planı'nı yayınlamıştır.[130] Temmuz 2020'de Türkiye Cumhuriyeti Cumhurbaşkanlığı Dijital Dönüşüm Ofisi'nin yayımladığı Bilgi ve İletişim Güvenliği Rehberi incelendiğinde güvenli yazılım geliştirme konusuna eğildiği görülmektedir.[131] Kurumsal yazılım geliştirmenin güvenli yazılım geliştirme yaşam döngüsü yöntemleri ve modellerinden faydalanılarak güncellenmesi gerektiği, güvenli yazılım geliştirme yaşam döngüsü uygulanmasının gerekli olduğu belirtilmektedir. Bilgi ve İletişim Güvenliği Rehberi'ndeki öneriler dikkate alınarak Radyo ve Televizyon Üst Kurulunda geliştirilecek yazılımların kalitesi artırılmalı, kullanıcıların taleplerini karşılayacak şekilde fonksiyonel yazılım ürünleri ortaya çıkarılmalıdır.

Satın alma yoluyla kullanılacak yazılım projeleri için hazırlanacak teknik şartnamede talep edilen projeye dair tercih edilen yöntem belirtilebilir. Karar yüklenici firmaya bırakıldığı takdirde Kurum personelinin proje geliştirme sürecine dâhil olabilmesi adına yüklenici firmanın kullanacağı yazılım geliştirme yönteminin personel tarafından bilinmesi, sürecin daha sorunsuz bir şekilde ilerlemesini sağlayacaktır. Bu sayede hem personel Kurum bünyesine katılacak olan yazılımlar hakkında bir anlayış geliştirebilir ve sürece dair bir hâkimiyet sağlayabilir hem de proje sürecinde projenin gidişatını gözlemleyerek proje konusunda yardımcı olabilir. Her iki durumda da projenin kontrolünün en iyi şekilde yapılabilmesi için personelin yazılım geliştirme süreci ile ilgili bilgi sahibi olmasının gerekli olduğu değerlendirilmektedir.

Kurum içerisinde geliştirilecek olan yazılım projelerinde gelen taleplerin özellikleri oldukça önemlidir. Tezin Yazılım Projelerinin Başarısız Olma Sebeplerinden bahsedildiği kısımda belirtildiği gibi gereksinimlerin açık bir şekilde belirlenmesi bir yazılım projesini başarıya götüren ilk adımlardan biridir. İstenilen

[130]Türkiye Cumhuriyeti Ulaştırma ve Altyapı Bakanlığı (29.12.2020), https://hgm.uab.gov.tr/haberler/ulusal-siber-guvenlik-stratejisi-ve-eylem-plani-2020-2023-yayimlandi, (21.03.2021)

[131]Türkiye Cumhuriyeti Cumhurbaşkanlığı Dijital Dönüşüm Ofisi (2020), https://cbddo.gov.tr/bgrehber, (22.03.2021)

yazılım ürününe dair yazılımın ne yapması gerektiğine hâkim bir personel, yazılım geliştirmeyi yapacak olan personel ile gereksinim toplantıları yaptığı takdirde ürüne dair ihtiyaçları açık ve anlaşılır bir şekilde ifade edebilecektir fakat gelen taleplerdeki gereksinimlerin yeterince açık olmayabileceği ihtimali göz ardı edilmemelidir. Hem bu sebeple hem daha yeni teknoloji olması hem de Kurum'daki personel sayısı ve işlevi göz önüne alındığında Kurum içerisinde yazılım geliştirilirken Çevik Yöntemlerden birinin kullanılması daha çok fayda sağlayacaktır. An itibariyle Kurum'da kullanılan bazı mevcut yazılım ürünlerinin belli bir yönteme bağlı kalmaksızın geliştirildiği gözlemlenmiştir. Bu sebeple bu mevcut yazılımlar tam olarak tamamlanmamış ve sürekli olarak personelin müdahalesine ihtiyaç duymaktadır. Yazılım geliştirme yaşam döngüsü kullanarak proje geliştirmek yazılım ürünlerinin kalitesini ve kontrolünü arttıracaktır. Bu sebeple tezde incelenen yöntemler arasında hem yöntemin özellikleri hem de Kurum'daki proje geliştiren personel göz önüne alınarak orta ve büyük çaplı yazılımlar için Scrum yönteminin, küçük çaplı yazılımlar içinse geleneksel yöntemlerden Artımlı Model'in Kurum içi projelere uyarlanarak uygulanmasının uygun olduğu değerlendirilmektedir.

Kurumda ihtiyaç duyulan bir yazılımı meydana getirmek üzere seçilen bir yazılım geliştirme yöntemi ile proje geliştirilirken Bütünleşik Yetenek Olgunluk Modeli'ne de yer verilmesinin süreç yönetimi ve süreç iyileştirilmesi açısından projeye katkıda bulunacağı düşünülmektedir. Zira bir süreç değerlendirme yöntemi olan CMMI, kullanılan yazılım geliştirme yöntemine bağlı olmaksızın yapılan işlerin amacına ve elde ettiği sonuca bakarak değerlendirme yapabildiği için esnek bir yapı sağlayabilir. Yazılım geliştirme yöntemleriyle bir arada kullanıldığı takdirde yazılım geliştirme yöntemi süreç içeriği ile ilgilenirken CMMI kurumsal seviyede kurum süreçleri için etkin, verimli ve disiplinli bir evrimsel iyileştirme yolu ortaya koyabilir.

Bir yazılım ürünü geliştirilirken o ürünün ne olduğu ne istediği ne yapması beklenildiği gibi hususları kavramak, kaynakları idareli ve uygun kullanmak ortaya iyi ve kullanışlı bir yazılım çıkarmak için oldukça önemlidir. Zira Tezin dördüncü bölümünde anlatılan yazılımların başarısız olma sebeplerinde belirtildiği gibi

gereksinimlerin belirlenmesi, kullanıcı etkisi, yazılımı geliştirilecek ekibin boyutu, yazılımın geliştirileceği zaman aralığı, yazılım üzerindeki kontrol, yazılımın test edilmesi ve kalite yönetimi hususları bir yazılım için oldukça önemlidir. Geliştirilecek yazılımın istenilen şekilde sonuç vermesi için bu konulara dikkat edilmelidir.

Temel olarak bir yazılım ürünü geliştirilmeye başlamadan önce aşağıdaki sorulara cevap vererek, projenin sınırları belirlenmelidir.

1- Yazılım ürünün kapsamı, ne tür fonksiyonlar içereceği, ne yapması hedeflendiği ve nasıl bir işleve sahip olması gerektiği gibi gereksinimler iyi bir şekilde anlaşılmalıdır.

2- Geliştirilecek yazılım ürünü için önemli olan noktalar belirlenmelidir. Bu yazılımın iyi bir kalitede olması, kısa süre içerisinde geliştirilip teslim edilmesi, az maliyetli olması gibi kavramlar olabilir.

3- Geliştirilmek istenen yazılım ürününe dair bir anlayış geliştirildikten sonra;
   a. Projede çalışacak personel
   b. Projenin süresi
   c. Projenin kalitesinin işlevselliğine göre önem oranı
   d. Projenin boyutu başlıklarına göre problem daha özel bir hale getirilmelidir. Aksi halde her bir durum için ayrı ayrı çalışılması ve birbirlerini etkileyecek unsurlarında değerlendirilmesi gerekmektedir.

4- Tezin önceki başlıklarında incelendiği gibi yazılım ürününü geliştirirken hangi yazılım geliştirme yöntemlerine başvuracağı belirlenmelidir. Her yazılım geliştirme yönteminin kendine göre avantajları ve dezavantajları vardır. Projeye göre bu yöntemlerden biri veya birden fazlası seçilerek birlikte kullanılmalıdır.

5- Yazılım ürününe dair ihtiyaçlar ve yazılım geliştirme yöntemi belirlendikten sonra ortaya çıkacak yazılımın neyi, nasıl yapması gerektiği detaylıca belirlenerek bu yazılımın yapısı, parçaları, parçaların birbiriyle olan ilişkisi ortaya konulmalı, kısaca yazılımın nasıl kurulacağı tasarlanmalıdır. Daha sonra yöntemin adımlarına göre projeye yön verilmelidir.

Bu beş adım sayesinde sürece dair gereksinim belirlenmesi, kullanıcı etkisi, projeyi geliştirecek ekip, projenin geliştirileceği zaman aralığı, yazılımın kalitesi ve yazılım üzerindeki kontrol belirlenebilir. Test konusunda ise geliştirme ekibi gerek kendi testlerini gerek kullanıcı testlerini proje geliştirme sürecinin uygun yerlerinde işleterek projenin çalışmasını kontrol etmeli, güvenliğini gözlemleyerek Bilgi ve İletişim Güvenliği Rehberi'nde de önerildiği gibi güvenli bir yazılım ortaya çıkarmak için gayret etmelidir.

Radyo ve Televizyon Üst Kurulu Bilgi Teknolojileri Dairesi için yazılım geliştiren personel sayısı, mevcut yazılımların sayısı ve işlevi, personelin iş yükü göz önüne alındığında yapılabilecek öneriler şunlardır:

- Herhangi bir yazılım ürünü ortaya çıkarılmak istenildiğinde süreci yönetecek olan kişilerin yazılım geliştirme yöntemlerine dair bilgi sahibi olmasının yanı sıra yazılım mühendisliğinin temel paradigmaları konusunda da bilgi edinmeleri gerekmektedir.

- Yazılım ürünleri istenilen gereksinimleri en iyi şekilde karşılamayı da göz önüne alarak zamana duyarlı bir şekilde kalite anlayışı içerisinde geliştirilmelidir.

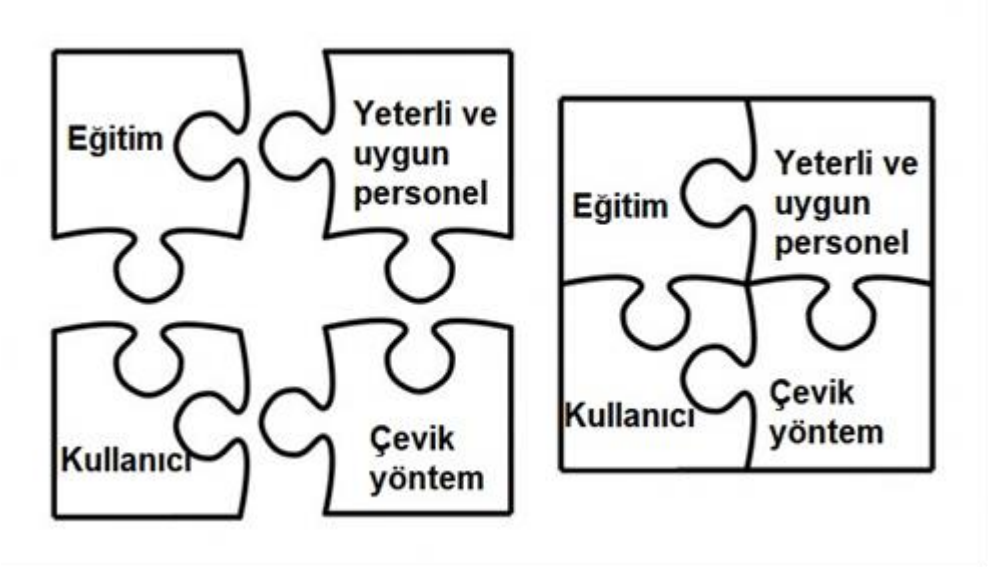
- Yazılımı geliştirecek olan personel, var olan yazılım geliştirme bilgilerinin yanı sıra yazılım mühendisliğinin temel konularından olan yazılım ve yazılım

geliştirme yöntemleri hakkında eğitimler almalıdırlar ve eğitim almak isteyen personel desteklenmelidir. Bu sayede personel Kurum'a daha faydalı olabilecektir. Proje geliştirilirken sürece dâhil olan personelin Kuruma fayda ve aidiyet duygusu da artacaktır.

- Yazılım geliştirme yaşam döngüsü hakkında farkındalık oluşturacak bilgiler edinilmeli ve mümkün olduğunca tekrarlanmalıdır.

- Dokümantasyon, kullanıcılara yazılımın nasıl kullanılacağını ve problem olursa ne yapmaları gerektiğini anlatır. Bu sebeple yazılım geliştirilirken uygun seviyede dokümantasyon yapılmalıdır.

Ayrıca Kullanıcı tarafında olan personelin de yazılım geliştirme döngüsü ve işleyişi hakkında bilgilendirilmesi önemlidir. Birçok yazılımın başarısız olma sebebi kullanıcı katılımının olmaması dolayısıyla yazılım geliştiricinin ürünü kendi düşüncesine göre geliştirmesi veya kullanıcı katılımının yeterli olmamasından kaynaklanan geribildirim eksikliği sonucu projenin hedeften uzak bir şekilde sonlanmasıdır. Radyo ve Televizyon Üst Kurulu yüksek kullanıcı katılımına elverişli bir kurumdur. Radyo ve Televizyon Üst Kurulu kurum içi geliştirilecek projelerde kullanıcı personel ile geliştirme personeli aynı ortamda bulunabileceği için birbirlerine kolay bir şekilde ulaşıp iletişime geçebileceklerdir. Yani hem kullanıcı personel hem de yazılım geliştirme personeli için eğitim oldukça önemlidir. Bahsedilen öneriler Şekil 5.1'deki şema ile temsil edilebilir.



Şekil 5.1 RTÜK için önerilen şema

Şekil 5.1'de de görülebileceği ve daha önce bahsedildiği üzere eğitim hem geliştirmeyi yapacak personel için hem de kullanıcı personel için oldukça önemlidir. Çünkü bir projeyi geliştirmek için ilk adım düşünme ile ortaya çıkar ve proje çerçevesinde ortaya atılan fikirler sürecin ilerleyişini etkiler. Gerekli eğitimleri almış olan yeterli ve uygun sayıda personel ile yazılım geliştirme döngüsü hakkında bilgi sahibi olan kullanıcıların katkılarıyla, projeye uygun bir şekilde seçilen çevik yöntem sayesinde tıpkı yapboz parçalarının birbirine uyup ortaya bir bütün çıkarması gibi ortaya kaliteli ve güzel bir ürün çıkacaktır.

Yazılımı yapacak ekipteki kişi sayısı ve projenin bitirilme süresi personelin mevcut iş yükü mutlaka göz önünde bulundurularak en uygun şekilde belirlendiğinde ve yazılım geliştirme süreci sıkı tutulduğunda proje pürüzsüz bir şekilde ilerleyebilir. Böylece yapılan çalışmanın Radyo ve Televizyon Üst Kurulu projelerine katkı sağlaması mümkün olacaktır.

SONUÇ

Hayatımızın her noktasında kullandığımız elektronik cihazların içinde bulunan en küçük yazılımın bile bir hizmet sunduğu günümüzde yazılım geliştirmek oldukça titizlikle yapılması gereken bir faaliyettir. Hem uygun maliyette hem istenilen süre zarfında hem talep edilen fonksiyonları yerine getiren kaliteli bir yazılım oluşturmak için çabalamak her yazılımcının kendine düstur edinmesi gereken bir hedef olmalıdır. Bu çalışmada yazılım geliştirme yaşam döngüsü ve Bütünleşik Yerleşik Olgunluk Modeli ele alınarak iyi ve kaliteli bir yazılım geliştirmek için kullanılması önerilen mevcut yazılım geliştirme yöntemleri incelenmiş ve karşılaştırılmış, Bulgular ve Analiz bölümünde tabloda gösterilmiş, Tartışma ve Yorum bölümünde de bulgular yorumlanmıştır. Buna göre, literatürde önerilen SDLC model ve yöntemleri geliştirilen projeye göre uygun seçildiği ve izlendiği takdirde etkilidir ve ortaya kaliteli, iyi bir yazılım ürünü ortaya çıkmasını sağlamaktadır. Yazılım ürünü geliştiren geliştiricilerin ve kuruluşların yazılım geliştirme yaşam döngüsünü izlemesi gerektiği değerlendirilmektedir. Bu konuda Türkiye Cumhuriyeti Ulaştırma ve Altyapı Bakanlığı ve Türkiye Cumhuriyeti Cumhurbaşkanlığı Dijital Dönüşüm Ofisi'nin yayımları yazılım geliştiriciler için bir rehber görevi görmektedir.

Bu kapsamda bir proje geliştirirken izlenmesi gereken geliştirme yönteminin nasıl olacağı, projenin maliyetinin ne kadar olacağı, ne kadar süre içerisinde tamamlanmasının hedeflendiği, kullanıcının projeye ne kadar dâhil edilmesi gerektiği, projede yazılım ürününü geliştirecek olan personelin yazılım geliştirme yaşam döngüsüne dair ne kadar bilgisi olduğu ve yazılım geliştirilirken kurulacak ekipteki personel sayısı oldukça önemli faktörlerdir. Projenin en iyi şekilde ilerlemesini sağlayacak bir süreç yönetiminin benimsenmesi, uygun yazılım geliştirme yaşam döngüsü modeli veya yönteminin seçilmesi, yazılımı geliştirecek olan personelin ve kullanıcının yazılım geliştirme yaşam döngüsü hakkında bilgi sahibi olması bir yazılımın başarılı olmasında büyük rol oynamaktadır.

Radyo ve Televizyon Üst Kurulu Uzmanlık Tezi olarak hazırlanan bu çalışmada, iyi ve kaliteli yazılımlar geliştirilmesinin sağlanması için literatürdeki modeller ve yöntemler taranmış, ilgili modeller ve yöntemlerin avantajları ve dezavantajları araştırılmıştır. Araştırılan SDLC modelleri ve yöntemleri karşılaştırılmış ve hangisinin tercih edilebileceği hususunda bir karşılaştırma tablosu ortaya çıkarılmıştır.

Bu tez çalışması, bir yazılım projesi geliştirilirken Yazılım Geliştirme Yaşam Döngüsü'nün ve Bütünleşik Yerleşik Olgunluk Modeli'nin ne denli önemli olduğunu ortaya koymuştur. Bu çalışmanın, yazılım ürünü geliştirilirken uygun yazılım geliştirme yöntemini seçerek kullanıcı taleplerini karşılayan, uygun maliyetle ve zaman süresi içerisinde en iyi ve kaliteli yazılım ürününü ortaya çıkarmayı sağlamaya katkı sunma amacına ulaşması temenni edilmektedir.

KAYNAKÇA

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen ve Juhani Warsta (2002), “Agile Software Development Methods: Review and Analysis”, *Proc. Espoo 2002*, s. 21-42.

Ahmad, Muhammad Ovais, Markkula, Jouni ve Oivo, Markku (2013), "Kanban In Software Development: A Systematic Literature Review." *2013 39th Euromicro conference on software engineering and advanced application IEEE*, s. 9-16.

Alexander, Linda, ve Davis, Alan (1991), "Criteria for the Selection of a Software Process Model," *Fifteenth IEEE International Conference on Computer Software and Applications (COMPSAC)*, s. 521-528.

Ali, Kazim (2017), "A Study of Software Development Life Cycle Process Models", *International Journal of Advanced Research in Computer Science*, Sayı: 8, s. 15-23

Alsahli, Abed ve Ramadan, Nagy (2019), “A Proposed Approach for Tracing the Progress of XP Projects”, *Indian Journal of Science and Technology,* Sayı: 12, s.3.

Alshamrani, Adel ve Bahattab, Abdullah (2015), “A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, And Incremental/Iterative Model”, *International Journal of Computer Science Issues (IJCSI)*, Sayı: 12, No:1, s. 106-107.

Anwer, Faiza, Aftab, Shabib, Shah, Syed ve Waheed, Usman (2017), “Comparative Analysis Of Two Popular Agile Process Models: Extreme Programming And Scrum”, *International Journal of Computer Science and Telecommunications,* Sayı:8 No:2, 2017, s. 5.

Aydan, Ufuk, Yılmaz, Murat ve O'Connor, Rory (2015), “Towards a Serious Game to Teach ISO/IEC 12207 Software Lifecycle Process: An Interactive Learning Approach.”, *15th International Conference SPICE*, Sayı: 526, s. 6.

Baban, Nalkar Sanjivani (2016), "Processing Models of SDLC", s. 1-2.

Barjtya, Sahil, Sharma, Ankur ve Rani, Usha (2017), “A Detailed Study Of Software Development Life Cycle (SDLC) Models”, *International Journal of Engineering and Computer Science*, Sayı:6, No: 7, s. 22099.

Baysan, Serdar ve Durmuşoğlu. Bülent (2013), "*Ürün Geliştirmede Yalın Süreçlere Dönüşüm*", İstanbul Teknik Üniversitesi

Beck, Kent (1999), "Embracing Change With Extreme Programming", *Computer*, Sayı: 32, No: 10, s. 70-77.

Beck, Kent (2000), *Extreme Programming Explained: Embrace Change*, Addison-wesley professional, Boston.

Bersoff, Edward ve Davis, Alan (1987), "A New Look at the C3I Software Life Cycle," *SIGNAL*, Sayı: 41, No:8, s. 85-93

Blancard, Benjamin Seaver ve Fabrycky, Wolter Joseph (2006*), Systems Engineering And Analysis 4th Ed.*, Pearson Prentice Hall, New Jersey.

Boehm, Barry (1998), "A Spiral Model Of Software Development And Enhancement", *Computer*, Sayı: 21, No. 5, s. 61-72.

Cervone, Frank, (2011), "Understanding Agile Project Management Methods Using Scrum", *OCLC Systems & Services: International Digital Library Perspectives*, Sayı: 27, s. 20.

CMMI Product Team, *"CMMI for Development, version 1.2."*, 2006, s. 18.

Cockburn, Alastair (2004), *Crystal Clear: A Human-Powered Methodology For Small Teams: A Human-Powered Methodology For Small Teams*, Pearson Education.

Cohn, Mike (2004), *User Stories Applied: For Agile Software Development*, Addison-Wesley Professional, Boston.

Costa, Gonçalo (2017), "Spring 6-10-2017 A Survey Of Faıilures In The Software Development Process", *ECIS 2017 Proceedings*, s. 2450-2451.

Datyal, Deeksha, "Proposed Model to Overcome the Problems in Waterfall Model", *International Journal of Computer Science & Information Technology*, Sayı: 2, No. 2, 2015, s. 29-32.

Davis, Alan (1992), “Operational Prototyping: A New Development Approach”, *IEEE Software*, s. 70-78.

Dudziak, Thomas (1999), “Extreme programming an overview”, *Methoden und Werkzeuge der Software: produktion* WS, s.15-16.

Erten, Murat (2016), *Bilgisayar Mühendisliği'ne Giriş*, Papatya Yayıncılık, İstanbul.

Everett, Gerald ve McLeod, Raymond (2007), *Software Testing: Testing Across the Entire Software Development Life Cycle*, Wiley-IEEE Press, New Jersey.

Flora, Harleen ve Chande, Swati (2014), "A Systematic Study On Agile Software Development Methodologies And Practices" *International Journal of Computer Science and Information Technologies*, Sayı:5, No:3, s. 3632.

Forsberg, Kevin ve Mooz, Harold (1991), “The Relationship of System Engineering to the Project Cycle”, *The 12th INTERNET World Congress on Project Management*, s. 4.

Huang, Chun-Che ve Andrew Kusiak (1996), “Overview of kanban systems”, *Int. J. Computer Integrated Manufacturing*, s. 169.

IEEE (2000), "IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*,” IEEE Std 1471-2000*, s. 1-30

Ivanovskaya, Anastasiya, Aksyonova, Olga ve Ziomkovskaya, Polina (2020), "Software Life Cycle Model Development at BPsim.DSS,", *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT),* s. 483-487.

Javanmard, Mahdi ve Alian, Maryam (2015), "Comparison Between Agile And Traditional Software Development Methodologies", *Cumhuriyet Üniversitesi Fen-Edebiyat Fakültesi Fen Bilimleri Dergisi*, Sayı: 36, No:3, s. 1386-1391.

Juric, Radmila (2000), “Extreme Programming And Its Development Practices”, *Proceedings of the 22nd International Conference on Information Technology Interfaces*, s. 97 - 104.

Kara, Helen (2015), *Creative Research Methods In The Social Sciences: A Practical Guide,* Policy Press, Bristol.

Kaur, Rupinder ve Sengupta, Jyotsna (2013), "Software Process Models and Analysis on Failure of Software Development Projects", *International Journal of Scientific & Engineering Research,* Sayı: 2, No: 2, s. 3-4.

Khurana, Gourav ve Gupta, Sachin (2012), "Study & Comparison of Software Development Life Cycle Models", *International Journal of Research in Engineering & Applied Sciences,* Sayı: 2, No: 2, s. 1516-152.

Kumar, Sujit ve Dubey, Pushkar (2013), "Software Development Lıfe Cycle (Sdlc) Analytical Comparison And Survey On Traditional And Agile Methodology", *Abhinav National Monthly Refereed Journal Of Research In Science & Technology,* Sayı: 2, s. 23.

Kute, Sheema Suresh ve Thorat, Surabhi Deependra (2014), "A Review on Various Software Development Life Cycle (SDLC) Models", *International Journal of Research in Computer and Communication Technology,* Sayı: 3, No: 7, s. 778-779.

Larman, Craig, ve Basili, Victor (2003), "Iterative And Incremental Developments. A Brief History", *Computer*, Sayı: 36, No:6, s. 47-56.

Lei, Howard, Ganjeizadeh, Farnaz, Jayachandran, Pradeep Kumar ve Özcan, Pınar (2017), "A Statistical Analysis Of The Effects Of Scrum And Kanban On Software Development Projects", *Robotics and Computer-Integrated Manufacturing,* Sayı: 43, s. 60.

Lindstrom, Lowell ve Jeffries, Ron (2004), "Extreme Programming And Agile Software Development Methodologies", *Information systems management*, Sayı: 20, No: 3, s. 43.

Mahmood, Shahid (2015), *Capability Maturity Model Integration CMMI,* Virtual University of Pakistan.

Mahnič, Viljan (2013), "Applying Kanban Principles To Software Development", *Information Technology for Practice,* s. 89.

Mandal, Ardhendu ve Pal, S. (2015), “Identifying the Reasons for Software Project Failure and Some of their Proposed Remedial through BRIDGE Process Models”, *International Journal of Computer Sciences and Engineering*, Sayı: 3, s. 118-126.

Matharu, Gurpeet Singh, Mishra, Anju, Singh, Harmet ve Upadhyay, Priyanka (2015), “Empirical Study Of Agile Software Development Methodologies: A Comparative Analysis”, *ACM SIGSOFT Software Engineering Notes*, Sayı: 40, No: 1, s. 2-3.

Mnkandla, Ernest ve Dwolatzky, Barry (2004), “A Survey Of Agile Methodologies”, *The Transactions Of The SA İnstitute Of Electrical Engineers,* Sayı:3, s. 236-247.

Mollaoğlu, Ayça Alparslan ve Sönmeztürk Bolatan, Gülin İdil (2020), "Yazılım Süreçlerinde Kalite Ve Yetenek Olgunluk Modeli." *Organizasyon ve Yönetim Bilimleri Dergisi,* Sayı: 12, No:1, s. 82-83.

Munassar, Nabil Mohammed Ali ve Govardhan, Dr. A. (2010), “A Comparison Between Five Models Of Software Engineering”, *International Journal of Computer Science Issues - IJCSI*, Sayı: 7, No: 5, s. 95.

Newkirk, James (2002), “Introduction To Agile Processes And Extreme Programming”, *Proceedings of the 24th International Conference on Software Engineering*, s. 695-696.

Nugroho, Suryanto, Waluyo, Sigit Hadi ve Hakim, Luqman (2017) “Comparative Analysis Of Software Development Methods Between Parallel, V-Shaped And Iterative”, *International Journal of Computer Applications*, Sayı: 169, No: 11, s. 8.

Paulk, Mark (2001), "Extreme programming from a CMM perspective." *IEEE Software*, Sayı:18, No:6, s. 19-26.

Palmquist, Steve, Lapham, Mary Ann, Miller, Suzanne, Chick, Timothy ve Ozkaya, Ipek (2013), “Parallel Worlds: Agile and Waterfall Differences and Similarities”, Carneige Mellon University Pensilvanya, s. 5-6.

Pressman, Roger (2005), *Software Engineering: A Practitioner's Approach 5th Edition*, McGraw-Hill, New York.

Pressman, Roger (2010), *Software Engineering: A Practitioner's Approach 7th Edition,* McGraw-Hill, New York.

Pressman, Roger ve Maxim, Bruce (2014), *Software Engineering: A Practitioner's Approach 8th Ed,* McGraw-Hill Education, New York.

Qasaimeh, Malik, Mehrfard, Hossein ve Hamou-Lhadj, Abdelwahab (2008), "Comparing Agile Software Processes Based On The Software Development Project Requirements", *2008 International Conference on Computational Intelligence for Modelling Control & Automation,* s. 51.

Rastogi, Vanshika (2014), "Software Development Life Cycle Models-Comparison, Consequences", *International Journal of Computer Science and Information Technologies (IJCSIT)*, Sayı: 6, s.168-171

Rohil, Harish ve Syan, Manisha (2012), "Analysis of Agile and Traditional Approach for Software development", *International Journal of Latest Trends in Engineering and Technology*, Sayı: 1, No: 4, s. 1-3.

Royce, Winston (1987), "Managing The Development Of Large Software Systems: Concepts And Techniques", *Proceedings of the 9th international conference on Software Engineering,* s. 328–338.

Ruparelia, Nayan (2010), "Software Development Lifecycle Models", *ACM SIGSOFT Software Engineering Notes*, Sayı: 35, No: 3, s. 8-13.

Sabale, Rajendra Ganpatrao ve Dani, Dr. A. R. (2012), "Comparative Study Of Prototype Model For Software Engineering With System Development Life Cycle", *IOSR Journal of Engineering*, Sayı: 2, No: 7, s. 21-24.

Saraçoğlu, Halil İbrahim (2009), *CMMI Modeli Ve Çevik Yazılım Geliştirme Yöntemlerinin Entegrasyonu, Erciyes Üniversitesi*, Yüksek Lisans Tezi.

Sarıdoğan, Mustafa Erhan (2014), *Yazılım Mühendisliği*, Papatya Yayıncılık, İstanbul.

Schwaber, Ken (1997), *Scrum Development Process,* Springer, London.

Schwalbe, Kathy (2015), *Information Technology Project Management*, Cengage Learning, Boston, 2015.

Sharma, Mohit Kumar (2017), "A Study Of SDLC To Develop Well Engineered Software*", International Journal of Advanced Research in Computer Science*, Sayı: 8, s. 521.

Singh, Raghu (1996), "International Standard ISO/IEC 12207 Software Life Cycle Processes", *Software Process: Improvement and Practice*, Sayı: 2, s. 36.

Sommerville, Ian (2004), *Software Engineering 7th edition*, Pearson Addison-Wesley, Boston.

Sommerville, Ian (2011), *Software Engineering 9th edition*, Pearson Addison-Wesley, Boston.

Soylu, Salih (2017), *Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği Ve Çevre Ve Şehircilik Bakanlığı İçin Öneriler,* Uzmanlık Tezi.

Sundararajan, Mukund ve Mahalakshmi, M. (2013), "Traditional SDLC Vs Scrum Methodology–A Comparative Study", *International Journal of Emerging Technology and Advanced Engineering,* Sayı:3, No:6, s. 193.

Sutherland, Jeff (2010), *Jeff Sutherland's Scrum Handbook,* Scrum Training Institute.

Sutherland, Jeff ve Schwaber, Ken (2007), "The Scrum Papers. Nuts, Bolts and Origins of an Agile Process", s. 27.

Tanrıkulu, Orhan (2011), C*MMI'ın Bir Türk Bankacılık Sektöründe Uygulanması ve Analizi*, Haliç Üniversitesi, Yüksek Lisans Tezi.

Usman, Atiku Victor ve Nonyelum, Ogwueleka Francisca (2018), "SDLC Models as Tools in the Development of MIS: A Study", *IUP Journal of Information Technology,* Sayı: 14, No: 4, s. 52-59.

Verner, June, Sampson, Jennifer ve Cerpa, Narciso (2008),"What Factors Lead To Software Project Failure?" *Proceedings of the 2nd International Conference on Research Challenges in Information Science RCIS*, s. 71 - 80.

Voigt, Benjamin (2004), “Dynamic system development method”, *no. January,* 2004, s. 2-12.

Zafar, Iqra, Khan Nazir, Aiman ve Abbas, Muhammad (2017), “The Impact Of Agile Methodology (DSDM) On Software Project Management”, *In Circulation in Computer Science: International Conference on Engineering, Computing and Information Technology (ICECIT 2017),* s. 1-6.

ELEKTRONİK:

8th Annual Agility Report, http://www.agileturkey.org/ (24.02.2021)

8 Stances Of Scrum Master, https://www.scrum.org/resources/8-stances-scrum-master, s. 2-3, (01.03.2021)

Büyük Türkçe Sözlük, http://www.tdk.gov.tr, (24.12.2020, 26.12.2020)

https://evansdata.com/press/viewRelease.php?pressID=278, (23.01.2021)

Principles behind the Agile Manifesto, https://agilemanifesto.org/principles.html, (27.01.2021)

Project Management Institude (1996), A Guide to the Project Management Body of Knowledge, http://www.cs.bilkent.edu.tr/~cagatay/cs413/PMBOK.pdf, (10.02.2021)

Sadi Evren Şeker, Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü (SDLC), YBS Ansiklopedisi, 2015, s. 18 – 29, http://ybsansiklopedi.com/cilt-2-sayi-2-eylul-2015/, (13.02.2021)

The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game, https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf, (27.02.2021)

Toyota Global, https://www.toyotaglobal.com/company/history_of_ toyota/75years /text/entering_the_automotive_business/chapter1/section4/item4.html, (21.02.2021)

Türkiye Cumhuriyeti Cumhurbaşkanlığı Dijital Dönüşüm Ofisi (2020), https://cbddo.gov.tr/bgrehber, (22.03.2021)

Türkiye Cumhuriyeti Ulaştırma ve Altyapı Bakanlığı (29.12.2020), https://hgm.uab.gov.tr/haberler/ulusal-siber-guvenlik-stratejisi-ve-eylem-plani-2020-2023-yayimlandi, (21.03.2021)

Türk Standartları Enstitüsü, https://www.tse.org.tr/IcerikDetay?ID=945&ParentID=1214 (25.02.2021)

ÖZET

Yazılım kavramı sağlık, finans, haberleşme gibi çeşitli alanlarda insan hayatına önemli bir şekilde dâhil olmuştur. Bir yazılım ürünü geliştirilirken hizmet kalitesinin yüksek olması, maliyetinin uygun olması, yazılımın kullanıcı için verimli ve etkili olması, belirtilen sürede tamamlanması, yazılımın kullanıcı talebini en iyi şekilde karşılaması gibi önem verilmesi gereken çeşitli faktör vardır. Bir yazılımı, o yazılımdan en iyi şekilde faydalanılacak şekilde geliştirmek gerekmektedir. Bu sebeple Yazılım Yaşam Döngüsü, Süreç Modelleri ve RTÜK'e Uyarlanabilirliği bu tezin konusu olarak seçilmiştir.

Tez çalışmasının birinci kısmında Yazılım ve Yazılım Geliştirme Yaşam Döngüsü ile ilgili kavramlar açıklanmış, bu kavramların birbiriyle olan ilişkisi, tanımlar ve çeşitli görseller aracılığıyla aktarılmıştır. İkinci kısımda yazılım geliştirilirken kullanılan Geleneksel Yöntemler teze konu edilmiş, Geleneksel Yöntemler araştırılarak avantajları ve dezavantajları belirlenmiştir. Üçüncü kısımda Çevik Yazılım Geliştirme Yöntemleri ele alınmış, bu yöntemler incelenerek avantajları ve dezavantajları belirlenmiştir. Tezin dördüncü kısmında süreç değerlendirme modeli olan Bütünleşik Yerleşik Olgunluk Modeli ele alınmıştır. Tezin beşinci kısmında. bir yazılımın başarısız olma sebepleri incelenmiştir. Bulgular ve Analiz başlığında tez çalışmasında toplanan veriler karşılaştırmalı analiz yöntemiyle incelenmiş, elde edilen bulgular bir tablo ile gösterilmiştir. Tartışma ve Yorum başlığında bu tez çalışmasında incelenen Yazılım Geliştirme Yaşam Döngüsü Yöntemleri ile Bulgular ve Analiz kısmında elde edilen sonuçlar doğrultusunda yazılım geliştirilirken dikkat edilmesi gereken hususlar hakkında önerilerde bulunulmuş ve Radyo ve Televizyon Üst Kurulu üzerinden kamu kurumlarının da izleyebilecekleri bir yol haritası ortaya konulmuştur. Sonuç kısmında tez çalışmasının ortaya çıkardığı netice değerlendirilmiştir.

Arşiv Kayıt Bilgileri

Tezin Adı: Yazılım Yaşam Döngüsü, Süreç Modelleri ve RTÜK'e Uyarlanabilirliği

Tezin Yazarı: Melike Gizem DAVUTOĞULLARI

Tezin Danışmanı: Cem ULUOĞLAKCI

Tezin Konusu: Uzmanlık Tezi

Tezin Tarihi: Haziran / 2021

Tezin Alanı: Yazılım Geliştirme Yaşam Döngüsü

Tezin Yeri: RTÜK / ANKARA

Anahtar Kelimeler: Yazılım, Yazılım Geliştirme Yaşam Döngüsü, SDLC, Geleneksel Yöntemler, Çevik Yöntemler, Agile, Süreç Modelleri, CMMI

ABSTRACT

The concept of ‘software’ has been significantly involved in human life in various fields such as health, finance and communication. When developing a software product, there are several factors that need to be given importance, such as high quality of service, affordable cost, efficient and effective software for the user, completion within the specified time, and the best way for the software to meet the user's demand. It is necessary to develop a software in such a way that the best use of that software is made. For this reason, Software Development Lifecycle, Process Models and Applicability to RTÜK were selected as the subject of this thesis.

In the first chapter of the thesis, concepts related to Software and Software Development Lifecycle were explained, the relationship of these concepts to each other is conveyed through definitions and various images. In the second chapter, the Traditional Methods used in software development were discussed, the advantages and disadvantages have been examined by investigating these Traditional Methods. In the third chapter, Agile Software Development Methods were discussed and their advantages and disadvantages have been examined. In the fourth chapter, Capability Maturity Model Integrated has been examined. In the fifth chapter the reasons for software failure has been examined. In the Findings and Analysis chapter, the data collected in the thesis are presented with comparative analysis method, and the results are shown with a table. In the Discussion and Comment chapter, through Software Development Lifecycle Methods examined in this thesis and results obtained in the Findings and Analysis chapter suggestions have been made to ensure for software development and a road map for RTÜK has been put forward. In the Conclusion part, the result of the thesis has been evaluated.

ÖZGEÇMİŞ

Melike Gizem DAVUTOĞULLARI 1994 yılında Ankara'da doğdu. Sırasıyla Ahmet Hamdi Tanpınar İlkokulu, Onuncu Yıl İlkokulu ve Gazi Çiftliği Anadolu Lisesi'nde okuduktan sonra 2016 yılında Ankara Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği bölümünden mezun olup lisans derecesi aldı. 2018 yılından itibaren Radyo ve Televizyon Üst Kurulunda görev yapmaktadır.