KAMU KURUMLARINDA YAZILIM YAŞAM DÖNGÜLERİNİN UYGULANABİLİRLİĞİ VE ÇEVRE VE ŞEHİRCİLİK BAKANLIĞI İÇİN ÖNERİLER

UZMANLIK TEZİ

HAZIRLAYAN: SALİH SOYLU

ANKARA-2017

T.C

ÇEVRE VE ŞEHİRCİLİK BAKANLIĞI

KAMU KURUMLARINDA YAZILIM YAŞAM DÖNGÜLERİNİN UYGULANABİLİRLİĞİ VE ÇEVRE VE ŞEHİRCİLİK BAKANLIĞI İÇİN ÖNERİLER

Tez Hazırlayanın Adı Soyadı: Salih SOYLU
Tez Danışmanın Adı Soyadı: Nihan Şahin HAMAMCI
Birim Amirinin Adı Soyadı: Ömer ALAN

Salih SOYLU tarafından hazırlanan Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği ve Çevre ve Şehircilik Bakanlığı İçin Öneriler adlı bu tezin Çevre ve Şehircilik Uzmanlık tezi olarak uygun olduğunu onaylarım.

Çevre ve Şehircilik Uzmanı,
Nihan Şahin HAMAMCI
Tez Danışmanı

Bu çalışma, tez savunma komisyonumuz tarafından Çevre ve Şehircilik Uzmanlık tezi olarak kabul edilmiştir.

Başkan : (Genel Müdür V., Ömer ALAN)

Üye : (Genel Müdür Yardımcısı V., M.Yavuz TORUN)

Üye : (Daire Başkanı V., İskender ERMİŞ)

Üye : (Daire Başkanı V., Sibel ASLAN)

Üye : (Çevre ve Şehircilik Uzmanı, Nihan Şahin HAMAMCI)

Bu tez, Çevre ve Şehircilik Uzmanlığı Tez Hazırlama Yönergesi' ne uygundur.

İÇİNDEKİLER

ÖZET

ÇEVRE ve ŞEHİRCİLİK BAKANLIĞI	
Tezin Adı	Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği ve Çevre ve Şehircilik Bakanlığı İçin Öneriler
Türü	Çevre ve Şehircilik Bakanlığı Uzmanlık Tezi
Yazar	Salih SOYLU
Teslim Tarihi	01.04.2017
Anahtar Kelimeler	Yazılım, Yazılım Yaşam Döngüsü
Tez Danışmanı	Nihan Şahin HAMAMCI
Sayfa Adedi	100

Bilişim dünyasının gelişmesiyle birlikte bu değişime ayak uydurabilmek için Ülkemizde E-Devlet çalışmaları başlamıştır. Bu çalışmalarla ilgili 2016-2019 Ulusal e-Devlet Stratejisi ve Eylem Planı'nı hazırlanmıştır. Bu plan dikkate alınarak geliştirilecek yazılımların kalitesi arttırılmalı ve süreçleri iyileştirilmelidir. Bu yazılımlar talepleri karşılamalı ve kamu işlevlerini dijital ortamda sağlıklı bir şekilde yürütülmesini sağlamalıdır.

Bu kapsamda yapılan tez çalışmasında yazılım yaşam döngüsü modelleri, çevik yöntemler, Bütünleşik Olgunluk Modeli (CMMI) ve yazılım yaşam döngüsü standardı açıklanmıştır. Bunun yanında "Yazılım Yaşam Döngüsü" konusunda yapılan anket çalışmasıyla yazılım geliştirme yapan birimlerden ilgili teknik personelin düşünceleri öğrenilmeye çalışılmıştır. Bu çalışmaların sonucunda mevcut durum göz önünde bulundurularak Çevre ve Şehircilik Bakanlığı için yazılım süreçleri ile ilgili çeşitli önerilerde bulunulmuştur.

ABSTRACT

MINISTRY OF ENVIRONMENT AND URBANIZATION	
Thesis	Applicability of Software Life Cycles in Public Institutions and Recommendations for Ministry of Environment and Urbanization
Type	Ministry of Environment and Urbanization Expertise Thesis
Author	Salih SOYLU
Submission Date	01.04.2017
Key Words	Software, Software Life Cycle
Advisor	Nihan Şahin HAMAMCI
Total Page	100

With the development of the world of information, this change has begun e-government studies in our country to follow. The 2016-2019 National e-Government Strategy and Action Plan on these studies was prepared. Considering this plan, the quality of the software to be developed and the processes should be improved. These software must meet the demands and ensure that public functions are carried out in a healthy manner in digital environment.

In this thesis study, software life cycle models, agile methods, Integrated Maturity Model (CMMI) and software life cycle standard are explained. In addition to this, a survey study on "Software Life Cycle" has been tried to learn the thoughts of the related technical personnel from the software development units. As a result of these studies, various proposals have been made about the software processes for the Ministry of Environment and Urbanization considering the current situation.

TEŞEKKÜR

Tez çalışmam boyunca desteğini esirgemeyen başta nişanlım Elif YENİAY'a, Daire Başkanımız İskender ERMİŞ'e, tez danışmanım Nihan ŞAHİN HAMAMCI ve mesai arkadaşlarıma teşekkür ederim.

Salih SOYLU

TABLO LİSTESİ

ŞEKİL LİSTESİ

KISALTMALAR

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

ISO/IEC 12207: Yazılım Yaşam Döngüsü Süreci (Systems and Software Engineering - Software Life Cycle Processes)

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

ISO: Uluslararası Standardizasyon Teşkilatı (International Organization for Standardization)

IEC: Uluslararası Elektroteknik Komisyonu (International Electrotechnical Commission)

UML: Birleşik Modelleme Dili (Unified Modelling Language)

XP: Sınırsal(Uç) Programlama (Extreme Programming)

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

CMMI-SW: Yazılım Yetenek Olgunluk Modeli (Capability Maturity Model Integration Software)

CMMI-SE: Sistem Mühendisliği Olgunluk Yetenek Modeli (Capability Maturity Model Integration Systems Engineering)

CMMI-IPD: Entegre Ürün Geliştirme Yetenek Olgunluk Modeli (Capability Maturity Model Integration Integrated Product Development)

CMMI-SS: Tedarikçi Temini Yetenek Olgunluk Modeli (Capability Maturity Model Integration Supplier Sourcing)

CBS: Coğrafi Bilgi Sistemler

GİRİŞ

Yazılım; kamu ve özel sektör kurumlarının her alanda kullandığı sistemlerin ana bileşenlerinden birisidir. Yazılım geliştirme ve yönetimi için standartlar, yöntemler (prosedürler), metotlar, araçlar ve ortamlar hızla çoğalmaktadır. Bu çoğalma, yazılım geliştirme ve mühendisliğinde, özellikle de ürün ve hizmetlerin birleştirilmesinde büyük zorluklara sebep olmuştur. Yazılım disiplininin, bu çoğalma biçiminden, yazılımcıların yazılım oluşturma ve yönetmede "aynı dili konuşmaları" için kullanabilecekleri ortak bir çerçeveye taşınması gereklidir.(TS ISO/IEC 12207/ 2007)

Yazılım, bugünün dünyasında çok geniş bir uygulama alanına sahip ve her türlü iş için bir gereksinim konumundadır. Dolayısıyla, yüksek kalitede ve güvenli yazılım üretmek her türlü iş başarısı için vazgeçilmez bir öneme sahiptir. Yazılım kalitesini sağlamak için, yazılım mühendisliği proje yönetimi yazılım yaşam döngüsünün her aşamasında yer almalıdır. Yazılımın hem üretim, hem de kullanım süreci boyunca geçirdiği tüm aşamalar yazılım geliştirme yaşam döngüsü olarak tanımlanır. Yazılım geliştirme süreci, zamanlamaya dayalı ve içerik olarak bölünmüş aşamalardan oluşmaktadır. Bu sayede yazılım planlı bir şekilde geliştirilmektedir. Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu aşamalar sürekli bir döngü biçiminde ele alınır. Bu tür metodolojiler süreç bazlı olup teknik süreçlerin yanı sıra kurumsal süreçleri de içereceğinden Bakanlığımızın kurumsallaşmasına katkı sağlayacaktır.

Yazılım mühendisliği proje yönetiminin olmaması ya da yeterli olmaması, projelerin zaman, bütçe ve gerekli özellikleri yerine getirememekten dolayı başarısız olmalarına sebep olmaktadır. Bu yüzden yazılım yaşam döngülerinin uygulanabilirliği incelemesinde etkin proje yönetiminin önemli ana gereksinimlerden biri olduğu göz önünde bulundurulmalıdır. Diğer yandan, etkin ve verimli yazılım projesi yönetimi halen, yazılım organizasyonları için bir zorluk olarak karşımıza çıkmaktadır. Yazılım mühendisliği proje yönetimi planlama, koordinasyon ve geliştirme aşamalarının kontrolünü gerektirdiğinden, paydaşların memnuniyeti, gereksinimleri ve hedefleri garantileyecek, yazılım mühendislerine ürün ve geliştirmelerin uygulanmasında önemli kolaylık sağlayacak kararlar verilmelidir. Buna göre karar verme, yazılım

geliştirme sürecinde uygulanması gereken süreçlerden biri olmalıdır. Yazılım mühendisliği proje yönetimindeki kritik konulardan birisi de, projenin başarısını etkileyebilecek öneme sahip olan yazılım yaşam döngüsü modeli seçimidir. Yazılım geliştirme sürecinin tamamı seçilen model üzerine kurulduğundan, yazılım yaşam döngüsü modelinin seçimi projenin tüm aşamalarında işgücünün verimli bir şekilde kullanılması açısından vazgeçilmez bir unsurdur.

Bu uzmanlık tezi kapsamında; yazılım yaşam döngülerinden ve kullanılan süreç yöntemlerinden, Türkiye ve dünyadaki kullanım örneklerinden ve kamu kurumlarındaki yazılım süreçlerinden ve Bakanlık için yazılım geliştirme süreci için uygun süreç seçimi ve uygulanabilirliği için çözüm önerileri yer alacaktır.

1. YAZILIM YAŞAM DÖNGÜSÜ

Yazılım Geliştirme Yaşam Döngüsü, yazılım endüstrisi tarafından yüksek kalitede yazılımlar tasarlamak, geliştirmek ve test etmek için kullanılan bir süreçtir. Geliştirilen yazılımın, üretim aşaması ve kullanım süreci boyunca geçirdiği tüm aşamalar "Yazılım Geliştirme Yaşam Döngüsü" olarak tanımlanır. Yazılımın üretildiği aşamadan itibaren işlevsellik ile gereksinimleri sürekli değişecek ve bu değişiklikler de yazılımın genişlemesine neden olacaktır. Dolayısı ile bu aşamalar bir döngü olarak ele alınmak zorundadır. Bu döngü doğrusal ve tek bir yöne ilerleyen bir döngü değildir çünkü herhangi bir aşamada geriye dönmek, geliştirmeyi yapmak ve tekrar ilerlemek mümkündür.

Yazılım Geliştirme Yaşam Döngüsü (SDLC), yazılım geliştirme sürecinin her adımında gerçekleştirilen görevleri tanımlayan bir çerçevedir. Bu kapsamda ISO / IEC 12207, yazılım yaşam döngüsü süreçleri için uluslararası bir standarttır. Yazılım geliştirmede ve bakımında gerekli olan tüm görevleri tanımlayan standarttır.

1.1. Yazılım Yaşam Döngüsü Aşamaları

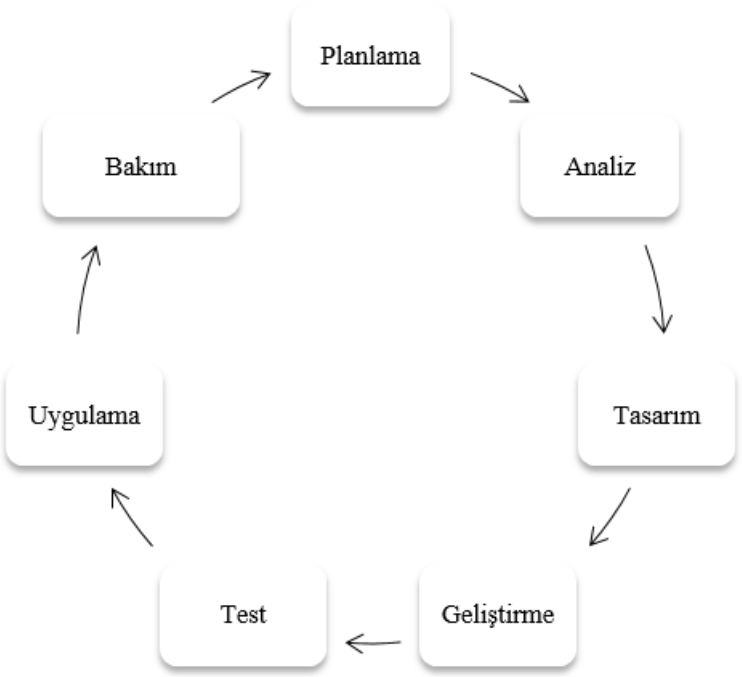
Yazılım yaşam döngüsü aşamaları literatür de farklı sayılarda ifade edilmektedir. Bu aşamalar 4 aşama ile 10 aşama arasında değişmektedir. Bu tez çalışmasında yazılım yaşam döngüsü aşamaları 7 aşama olarak değerlendirilmiştir.

Bu aşamalar:

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

aşamalarından oluşur.

Şekil 1: Yazılım Yaşam Döngüleri Adımları



1.1.1. Planlama

Planlama, proje hedeflerine ulaşmak için, belirlenmiş süre ve mali kriterleri göz önünde tutarak, izlenecek yöntem, yapılacaklar listesi, kullanılacak kaynaklar ve süre takviminin belirlenmesidir. Yazılım temel bir ihtiyacı esas alarak projenin kapsamı belirlenir ve planlama süreci başlamış olur.

Talepler ve proje sonucu çıkması istenen ürün özellikleri ayrıntılarıyla belirlenir ve yapılacaklar listesi ortaya çıkar. Ardından ihtiyaç duyulan kaynaklar belirlenir, tedarik ve görevlendirme süreci bunu takip eder. Bu işlerin yanında projenin hedef kitlesi olan kullanıcılar da belirlenmelidir. Projedeki asıl amaç müşteri ya da üst yönetimin istediği süre içinde projeyi tamamlamaktır.

1.1.2. Analiz

İhtiyaç analizi, paydaş, ihtiyaç ve beklentilerin ayrıntılı bir şekilde tanımlanmasıdır. İhtiyaç veya beklenti müşteri tarafından sistemin gerçekleştirmesi istenen temel görev veya fonksiyondur. Bir ihtiyaç aşağıdaki özellikleri taşımalıdır.

- **Gerçekleştirilebilme:** Mevcut insan ve mali vb. kaynaklarla yapılabilmesi
- **Doğrulanabilme:** Tarafsız ve ölçülebilir kriterlerle yapıldığını anlayabilme
- **Mananın tam anlaşılması:** Belirsiz ve birden fazla manaya gelebilecek ifadelerden kaçınılması
- **Çözümleri değil istekleri ifade etme:** Bu özellik ihtiyaç analizinin hedefini de açıkladığı için önemlidir.
- **Tutarlılık:** Diğer ihtiyaçlarla tezat teşkil etmemelidir.
- **Seviyesi veya yeri doğru olmalı:** Bir ihtiyaç sistem hiyerarşisinde doğru yerde tanımlanmalıdır. Basit bir ihtiyacı çok öne çıkartılmak veya karmaşık bir ihtiyacın en alt seviyede tanımlamak ürün yapısına uygun değildir. (System Engineering Fundamentals, 2001)

Yazılım tasarımını yapmadan önce proje sahibinin isteği büyük ölçüde anlaşılmalıdır. Ancak bunu ilk anda olması zordur. Nasıl bir yazılım geliştirileceği önceden proje sahibine demo halinde gösterilmesi, proje sahibine isteği sonucu elde edilecek ürün hakkında fikir verir.

1.1.3. Tasarım

Tasarım, planlama aşamasında istenen, analiz sonucunda belirlenen gereksinimleri yerine getirecek ve yazılım geliştirme araçlarının sınırları içerisinde geliştirilecek yazılımın üst seviye modelinin oluşturulması işlemidir. Tasarım aşamasında UML (Birleşik Modelleme Dili) şeması, ilişkisel veri modeli, nesne modeli, iş akış şeması gibi farklı yöntemler kullanılabilir.

1.1.4. Geliştirme

Proje sahibinin talepleri doğrusunda yapılmış olan tasarıma uygun yazılım geliştirme araçlarıyla vasıtasıyla ürüne dönüştürme işlemi işlemidir. Bu aşamada veri tabanının oluşturulması, kodlamanın yapılması, kullanıcı ara yüzlerinin oluşturulması ve raporların yazılması gibi işlemler yapılır. Projeye ait ilk çıktılar ortaya çıkmaya başlar. Geliştirilen modüllerle ilgili birim testleri de bu aşamada yapılabilir.

1.1.5. Test

Bu aşamada ise geliştirme aşamasında yapılanların kontrol edilmesi ve projeye entegre edilmesi sağlanır. Yapılan kontroller projenin önceki süreçlerinde belirlenen taleplerin karşılanması ve talepler karşılanmış ise doğru çalışıp çalışmadığı kontrol

edilir. Doğrulama işlemi için metot veya nesne seviyesinde birim testleri, bileşenler arasında entegrasyon testleri, bütünleşik sistem testi, performans ve yük testleri yapılır. Test sonucuna göre ihtiyaç duyulursa tasarım ya da kodlamada değişiklikler yapılır.

1.1.6. Uygulama

Bu aşamada; proje sahibine ürünün teslim edilmesi ve ürünün kullanılması aşamasına geçilir. Ürünün paydaşları tarafından ürünün anlaşılabilmesi önemlidir. Ürün teslim edildikten sonra ürünün son kullanıcıları doğru eğitilmezlerse sistemi anlayamayacaklar sistem verimli bir şekilde kullanılamayacaktır. Sistemin kullanıcıya başarılı bir şekilde teslim edilmesi için eğitim ve dokümantasyon çok önemlidir. Kullanıcılar sistemi kullanmaya başladıktan sonra artık sistem ilgili yeni talepler ve ihtiyaçlar ortaya çıkacaktır ve sistem olgunlaşmaya başlayacaktır.

1.1.7. Bakım

Bakım süreci, yazılım kullanılmaya başlandıktan sonra sistem üzerinde değişiklik yapılması sürecidir. Yazılım proje sahibi ve paydaşları tarafından yapılan isteklere göre sürekli değişir. Bu aşamada sisteme yeni eklentiler eklenir ya da hatalar giderilir.

Teslimden sonra değişiklik yapmanın temelde 3 nedeni vardır (Schach, 1993):

- Sistemde bulunan herhangi bir hatayı düzeltmek için (gereksinim, tasarım, kodlama, dokümantasyon hatası gibi) yapılan bakımlara doğrulayıcı bakım denir.
- Kusursuzlaştırma bakımı ise, değişiklikler ile ürünün etkinliğinin iyileştirilmesidir.
- Ürünün çevresinde yapılan değişikliklere cevap verebilmesi için yapılan değişikliklerdir.

Tompkins,Lientz ve Swanson 1978 yılında bir organizasyonda yaptıkları çalışmada, bakım yapan programcıların zamanlarının %17,5‘ini doğrulayıcı bakıma ayırdıklarını gözlemlemiştir. İlgili personelin zamanlarının %60,5’ini 2.tip bakım olan kusursuzlaştırma bakımına harcadıkları, %18 ’ini ise 3. tip bakıma ve %4’ünü bunların dışında bakıma ayırdıkları gözlenmiştir (Schach, 1993).

1.2. Yazılım Yaşam Döngüsü Yaklaşımına Farklı Bakış Açıları

Yazılım yaşam döngüsü literatürde değişik sayılır da ele alınmıştır. Bu yaklaşımlarından yazılım yaşam döngüsünü 4 adımda ele alan yaşam döngüsünü aşağıdaki gibi verebiliriz.

- **Planlama:** Bu aşamada planlamada düşünülen bütün işlevleri kapsar. Personel sayısı, zaman planlaması, hangi kaynakların kullanılacağı, proje sahibinin bilişim seviyesi ve sistemin kullanılabilirliği ve sürdürülebilirliği birçok parametre bu aşamada incelenir. Bu aşamanın tamamlanmasının ardından bir uygunluk raporu hazırlanarak yönetimine sunulur ve bilişim sisteminin yapacağı olumlu katkıların maliyet analizine göre katma değeri olduğu kesinleştirilir.

- **Sistem Tahlili:** Bu aşamada sistem için ayrıntılı analiz yapılır ve beklentiler, talepler listelenir. Proje paydaşlarıyla geliştirilmek istenen sistemin mevcut işleyiş içinde analizi yapılır.

- **Sistem Tasarımı:** Bu aşamada sistem analiz sonucu göre bütünsel olarak sistem tasarlanır. Tasarım yapılırken sistemin hedefleri ve etkileşimde olduğu alt bileşenleri arasındaki uyum gözetilir. Daha sonra bu aşamada programlama dilinden kullanılacak olan teknolojilere ve sistemin bütün parçalarına kadar olan tasarımı tamamlanır. (veri tabanı tasarımı, kullanıcı ekranları, veri akış yolları gibi).

- **Uyarlama ve işletme aşaması:** Bu aşama önceki aşamalarda belirlenen ve tasarım sonucunda ortaya çıkacak yazılımın bir ürün şeklini aldığı aşamadır. Kod geliştirilmesi, geliştirilen yazılım kodlarının incelenmesi, yapılan birim ve sistem testleri, yazılımın kurulması ve yazılımın yönetilmesi gibi süreçlerin tamamı bu süreçte gerçekleştirilir.
  (Şeker S.E, 2014)

Yukarıdaki 4 aşama kapsamında yazılım yaşam döngüsü adımlarının çıktıları aşağıdaki şekilde özetlenebilir:

Tablo 1: Yazılım Yaşam Döngüsü Aşamaları ve Çıktıları

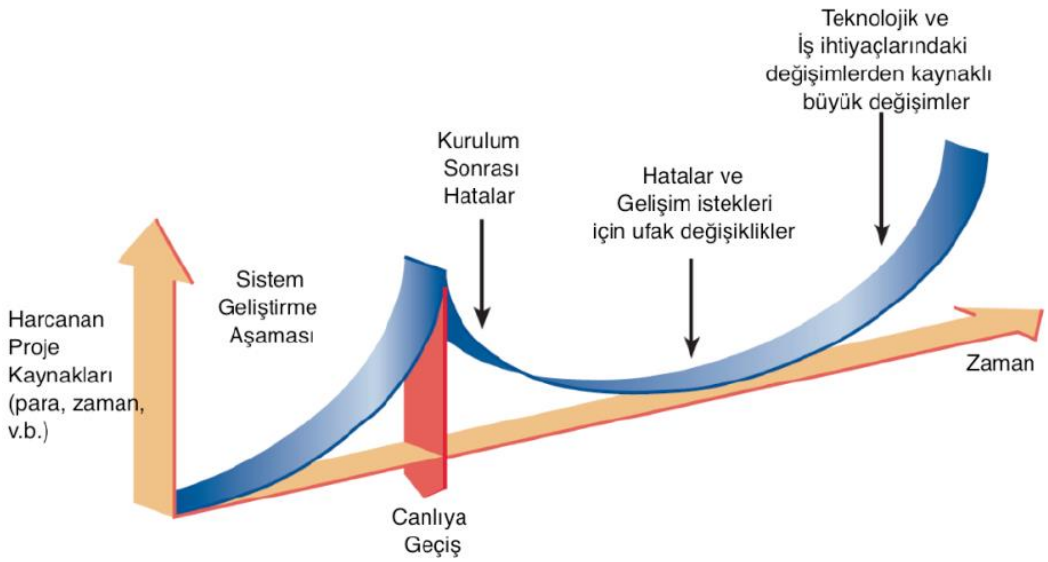
Aşama	**Çıktı**
Planlama	• Sistemin çalışacağı alan için özellikleri • Sistemin gerekleri • Bilgi sisteminin mimarisi • Seçilmiş olan proje için detaylı çalışma planı • Sistemin iş akışı açısından önemi
Sistem Tahlili	• Mevcut sistemin detaylı tanımı, • Mevcut sistemin iyileştirecek öneriler • Alternatif sistemlerin değerlendirilmesi • Yeni teknoloji gereksinimlerinin belirlenmesi ve geçiş için öneriler
Sistem Tasarımı	• Bütün sistemin detaylı bir tasarımı
Uyarlama ve İşletme	• Kodlama, dokümantasyon eğitim ve destek kaynakları • Yeni sürümler için kod, eğitim ve destek aşamalarındaki değişiklikler

(Şeker S.E, 2014)

Yukarıdaki 4 adımlı yaklaşımın yanında Kendall tarafından sunulan 7 adımlı yaklaşımdan söz etmek de mümkündür. Bu yaklaşım aşağıdaki adımlardan oluşur.

- Problem, fırsatlar ve hedeflerin belirlenmesi
- İnsan seviyesi enformasyon ihtiyaçlarının belirlenmesi
- Sistem ihtiyaçlarının belirlenmesi
- Şimdiye kadar olan adımlar ışığında elde edilen isteklere cevap veren bir sistemin tasarlanması
- Yazılımın geliştirilmesi ve dokümantasyonu
- Sistemin testleri ve bakım adımlarının belirlenmesi
- Sistemin gerçeğe alınması ve değerlendirilmesi (Kendall, 1992)

Şekil 2: Yazılım Yaşam Döngüsü Proje Maliyet Grafiği



(Şeker S.E, 2014)

Şekil 2'ye göre projenin geliştirilmesi sürecinde yukarı yönlü artan bir maliyet vardır. Proje canlı sisteme aktarıldıktan sonra maliyet grafiği aşağı yönlü hareket ederek düşmektedir. Proje paydaşlarından gelen taleplerin artması sonucu zaman içerisinde maliyet grafiğinde yine yukarı yönle bir artış gözükür. Hatta projenin hayata geçirilmesi için ilk aşamadaki maliyetini bile geçebilir. Bu yüzden proje geliştirilirken analiz ve tasarım aşamalarının çok iyi ileriyi düşünerek tasarlanması gerekir.

1.2.1. Yazılım Yaşam Döngüsü Yaklaşımının Avantajları ve Dezavantajları

Avantajları

Yazılım Geliştirme Yaşam Döngüsünde her bir aşamadan sonra gözden geçirme olduğu için proje sahiplerinin ve paydaşlarının sistemin geliştirilmesinde kontrolü vardır. Bu sayede planlanan hedeflere gidilip gidilmediği kontrol edilir. Aynı zamanda bu gözden geçirmeler yardımıyla hataların düzeltilmesi de mümkün olur. Bu yaklaşımda sistemin planlanmasından başlayarak analiz, tasarım kodlama gibi her bir aşama için detaylı dokümantasyon sağlar. Bu süreçler sayesinde kullanıcının ihtiyaçlarının karşılanıp karşılanmadığı ve standartlara uyulup uyulmadığının kontrol edilir. Yine bu kontrol için birçok ara aşamada mevcuttur.

Dezavantajlar

Yazılım yaşam döngüsü kullanmanın dezavantajlarından biri küçük ve sade projelere uygulanması durumunda maliyeti ve karmaşıklığı arttırır bu yüzden küçük projeler için uygun değildir. Projenin başlangıç aşamasında her şeyin ayrıntılı olarak bilinmesi mümkün olmadığından üretilen sonuçlarında sağlıklı bir ürün çıkmayabilir. Sisteme yaşam döngüsünde kullanıcılardan gelen bütün taleplerin eklenmesi sonucu proje içinden çıkılamayacak karmaşık bir hal alabilir. Bunun sonuncu olarak proje başlangıçta istenen şekilde sonuçlanmayabilir. Her aşamada dokümantasyon olması ve bu dokümantasyonların fazla olması zaman kayıplarına yol açabilir.

2. YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ

Literatürde birçok Yazılım Yaşam Döngüsü modeli vardır. Her modelin avantajları olduğu gibi dezavantajları da vardır. Fakat bir modelin her proje için en iyi olduğu konusunda belirli bir kural yoktur.

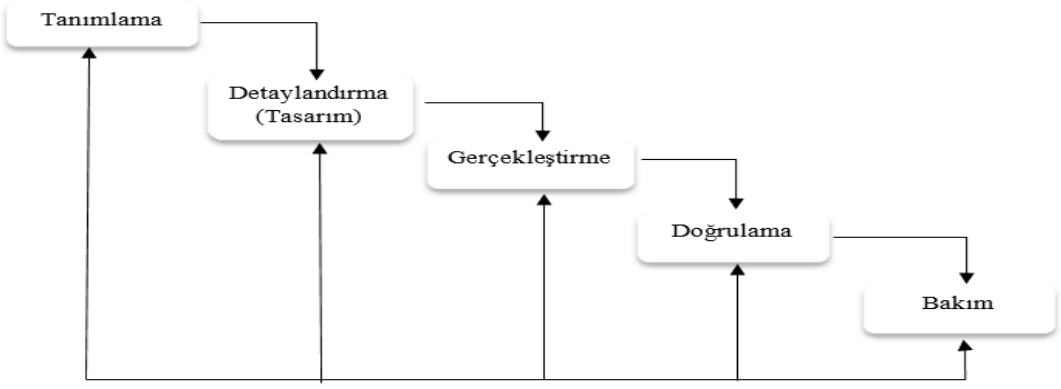
2.1. Şelale Modeli (Waterfall Model)

Şelale Modeli 1970 yılında Royce tarafından sistem mühendisliği sürecinden türetilen yazılım geliştirme sürecinin ilk yayınlanmış modelidir. (Sommerville,2004)

Bu model genellikle geleneksel model olarak adlandırılan klasik bir geliştirme yazılım modelidir. Şelale modeli ardışık bir yazılım yaşam döngüsü modelidir ve bu modeldeki gelişim ihtiyaç analizinin aşamaları olan tanımlama, tasarım, uygulama, test, entegrasyon ve bakım boyunca istikrarlı bir şekilde aşağıya doğru akar(Hiçdurmaz,2011).

Bu model statik bir yapıya sahiptir. Yazılım projeleri çok esnektir ve çok sık bir şekilde değiştiğinden şelale modeli yazılım projeleri için kötü yaklaşımlardan birisidir denilebilir.

Şekil 3: Şelale Modeli



(ISTQB Foundation Level, 2016)

Bu model her aşamada yazılım gereksinimleri, tasarım belgeleri, kod ve benzeri olaylar ile aşama aşama ilerlemektedir. Her aşama başlamadan önce gerçekleştirilmesi gereken faaliyetler vardır.

Bu modelin kilit noktası bir etkinlik tamamlandıktan sonra geri dönmemektedir. Dolayısıyla her etkinliği ve aşamayı bir gözden geçirme ile izlemek zorunludur. Kısaca bu model yazılım geliştirme sürecinin kullanıcı ihtiyaçlarını koda

dönüştüren adım adım bir süreç olarak planlanabileceğini varsayar(Christensen, Thayer, 2001).

2.1.1. Şelale Modelinin Avantajları ve Dezavantajları

Avantajları

- Proje ilerlemesinin daha doğru bir şekilde izlenmesini, muhtemel kaymaların erken tespitini ve ölçülebilir yazılım geliştirmeyi sağlar.
- Projeler daha yönetilebilir ve maliyet aşımı olmadan zamanında teslim edilir.
- Bütçeyi tahmin etme kolaylığı sağlar.
- Her aşamanın sonunda yapılan incelemeler kullanıcı katılımını sağlar.
- Sistemi test etmek ve korumak için kullanılan belgeleri üretir.
- Teknik açıdan zayıf veya deneyimsiz personel için iyi çalışır.

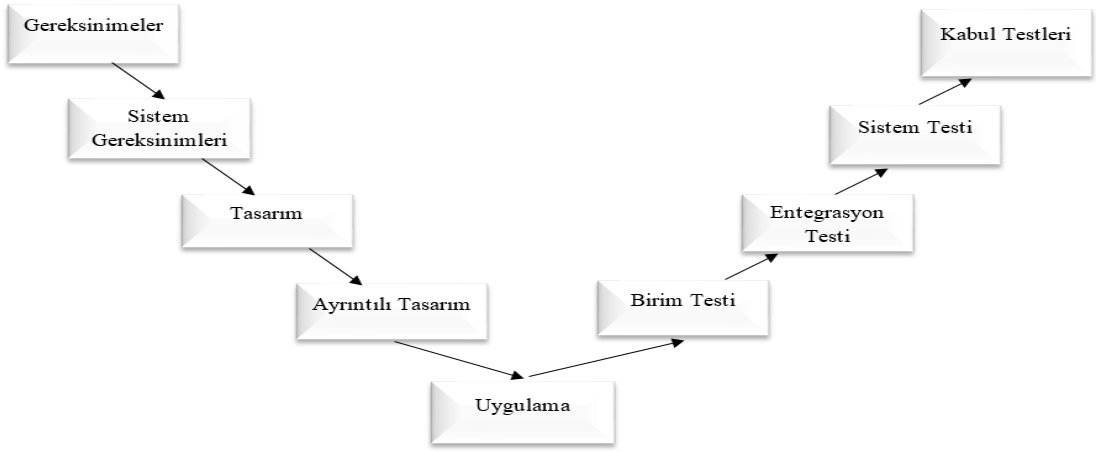
Dezavantajları

- Müşteriler, gereklilikleri tamamen, doğru ve net bir şekilde ifade etmelidir.
- Planlama ve belgeleme için çok fazla zaman harcanmaktadır.
- Projenin sonunda entegrasyon ve test için kapsamlı bir çaba gerektirir.
- Projenin sona ermesinden önce hiçbir gösterge yoktur.
- Gereksinimlerdeki değişiklikler ve hataları düzeltmek zor ve masraflıdır.
- Esnek değildir.
- Her türlü ihtiyacı önceden tahmin etmek zordur.
- Test aşamasına kadar fark edilmemiş tasarım hataları olabilir.

2.2. V-Şeklinde Model (V-Shaped Model)

V modeli, süreçlerin yürütülmesinin ardışık bir şekilde gerçekleştiği Yazılım Geliştirme Yaşam Döngüsü modelidir. Bu modelde onaylama ve doğrulama vardır.

Şekil 4: V Model



Bu modelde "V" yapısında bir yol izlenir ve adımlar bu şekildeki gibi gerçekleştirilir. Bu yol üzerinde sol taraf üretimi sağ taraf ise test işlemini ifade eder. Bu modelin çıktıları "Kullanıcı Modeli", "Mimari Model" ve "Gerçekleştirim Modeli" dir. Kullanıcı modelinde geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır. Mimari modelde sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevler ele alınmaktadır. Gerçekleştirim modelinde de yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar ele alınmaktadır. Bu model belirsizliklerin az iş tanımlarının belirgin olduğu bilişim teknolojileri projeleri için uygun bir modeldir. Ayrıca model kullanıcının projeye katkısını artırmaktadır(Yazılım Geliştirme Süreç Modelleri, 2016).

V model kullanımı için uygun olan senaryolar şu şekildedir;

- İş ihtiyaçları çok iyi tanımlanmış olmalıdır ve doküman haline getirilmiş olmalıdır.
- Ürün tanımı sabit olmalıdır, değişkenlik göstermemelidir.
- Teknoloji değişken olmamalıdır ve proje ekibi tarafından iyi bilinmelidir.
- Tüm ihtiyaçlar benzersiz ve eksiksiz biçimde tanımlanmış olmalıdır.
- Kısa süreli proje olmalıdır. (SDLC-V-Model, 2016)

2.2.1. V-Şeklinde Model Avantajları ve Dezavantajları

Avantajları

- Bu model oldukça disiplinli bir modeldir ve aşamalar birer birer tamamlanmıştır.

- Gerekliliklerin çok iyi anlaşıldığı küçük projeler için iyi çalışır.
- Basit ve kolay anlaşılır ve kullanılır.
- Her aşamadaki modelin sağlamlığı nedeniyle yönetilmesi kolaydır, belirli aşamalar ve inceleme süreci vardır.

Dezavantajları

- Karmaşık ve nesne yönelimli projeler için iyi bir model değildir.
- Uzun ve devam eden projeler için zayıf bir modeldir.
- İhtiyaçların orta ve yüksek risk altında olduğu projeler için uygun değildir.
- Bir uygulama test aşamasındayken, geriye dönüp bir işlevselliği değiştirmek zordur.
- Yaşam döngüsü sonuna kadar çalışma yazılımları üretilmez.
  (SDLC-V-Model, 2016)

2.3. Arttırımlı Model (Incremental Model)

Arttırımlı model yazılımın basit bir ihtiyacını karşılamakla başlayıp, ilerledikçe daha fazla ihtiyacı karşılayan ve sonunda müşterinin istediklerini karşılar halde yazılımı hazırlar. Döngüler daha küçük, daha kolay yönetilen modüllere bölünür. Her bir modül gereksinim, tasarım, uygulama ve test aşamasından geçer. Model, her bir yenilemede ürüne yeni özellikler katar. Yani her bir döngü, yeni bir sürüm anlamına gelebilir. Yazılım geliştirme süreci içerisinde birde fazla döngü, aynı anda gerçekleşebilir(Software Development Life Cycle, 2016).

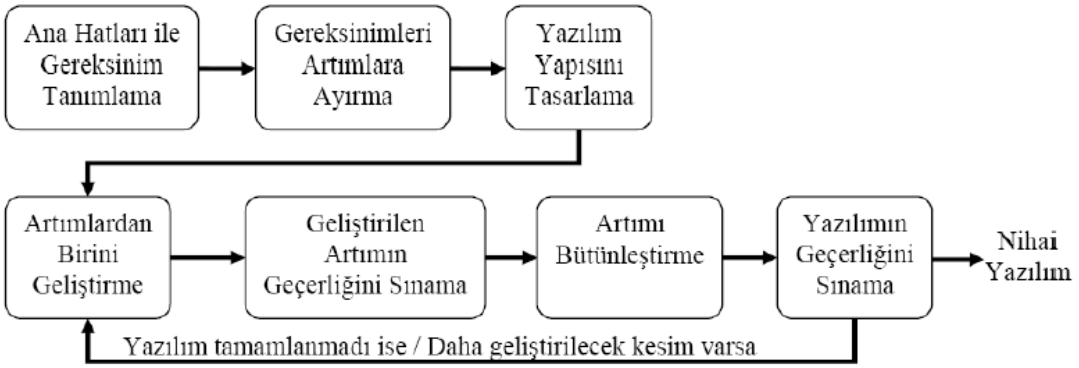
Şekil 5: Arttırımsal Örnek

(What is Incremental model- advantages and disadvantages, 2016)

Yukarıdaki şemada art arda çalışılırken şekiller parça parça eklenir ancak her parçanın tamamen bitmesi beklenir. Bu nedenle, parçaları tamamlanıncaya kadar eklemeye devam edilir. Yukarıdaki resimde olduğu gibi bir kişi uygulamayı düşünür. Ardından onu inşa etmeye başlar ve ilk adım da uygulamanın veya ürünün ilk modülü

tamamen hazır hale gelir ve müşterilere demo yapılabilir. Aynı şekilde, ikinci adım da diğer modül hazırlanır ve birinci modül ile bütünleştirilir. Benzer şekilde, üçüncü adım da tüm ürün hazır olur ve entegrasyon yapılır. Dolayısıyla ürün adım adım hazır hale gelir.

Şekil 6: Arttırımlı Model



(Anaral, 2011)

Arttırımsal bir yazılım geliştirme yaşam döngüsünün başarılı bir şekilde kullanılmasının anahtarı, koşulların titizlikle doğrulanması ve yazılımın her sürümünün, modelin her döngüsü içerisinde bu gereksinimlerle karşılaştırılmasıyla doğrulanmasıdır. Modelin her çevrimi, birim seviyesinde test, yazılım entegrasyonu, sistem entegrasyonu ve kabul için gerekli yazılımları üretir. Yazılım arka arkaya döngüler vasıtasıyla geliştikçe, yazılımların her bir sürümünü doğrulamak için testler tekrarlanmalıdır ve genişletilmelidir(Baban, 2016).

2.3.1. Arttırımlı Model Avantajları ve Dezavantajları

Avantajları

- Bazı çalışma fonksiyonları yaşam döngüsünün başlangıç aşamasında kolayca geliştirilebilir.
- Sonuçlar erken ve periyodik olarak elde edilir.
- Paralel gelişme planlanabilir.
- İlerleme ölçülebilir.
- Kapsamı / gereksinimi değiştirmek için daha az maliyetlidir.
- Daha küçük yineleme sırasında sınama ve hata ayıklama kolaydır.

- Riskler yineleme sırasında tanımlanır ve çözülür.
- Risk yönetimi kolaylığı vardır. Önce yüksek riskli kısım yapılır.
- Her artış ile operasyonel ürün teslim edilir.
- Her artıştan tespit edilen sorunlar, zorluklar ve riskler bir sonraki adımda çözülebilir.
- Risk analizi daha iyidir.
- Değişen gereksinimleri destekler.
- Başlangıçtaki çalışma süresi daha azdır.
- Büyük ve kritik önem taşıyan projeler için daha uygundur.
- Yaşam döngüsü sırasında, müşterinin değerlendirmesini ve geri bildirimini kolaylaştıran yazılımlar erken üretilir.

Dezavantajları

- Daha fazla kaynak gerekebilir.
- Değişim maliyeti daha düşük olmakla birlikte, değişen gereksinimler için çok uygun değildir.
- Sistem mimarisi veya tasarım sorunları ortaya çıkabilir, çünkü tüm gereksinimler tüm yaşam döngüsü başında toplanmamaktadır.
- Artımların tanımlanması, komple sistemin tanımlanmasını gerektirebilir.
- Daha küçük projeler için uygun değildir.
- Yönetim karmaşıklığı daha fazladır.
- Projenin sonu bilinmeyebilir bu sebeple bir risk oluşturur.
- Risk analizi için yüksek vasıflı kaynaklara ihtiyaç vardır.
- Projelerin ilerlemesi, risk analizi aşamasına oldukça bağımlıdır.

(SDLC - Iterative Model, 2016)

2.4. Spiral Model

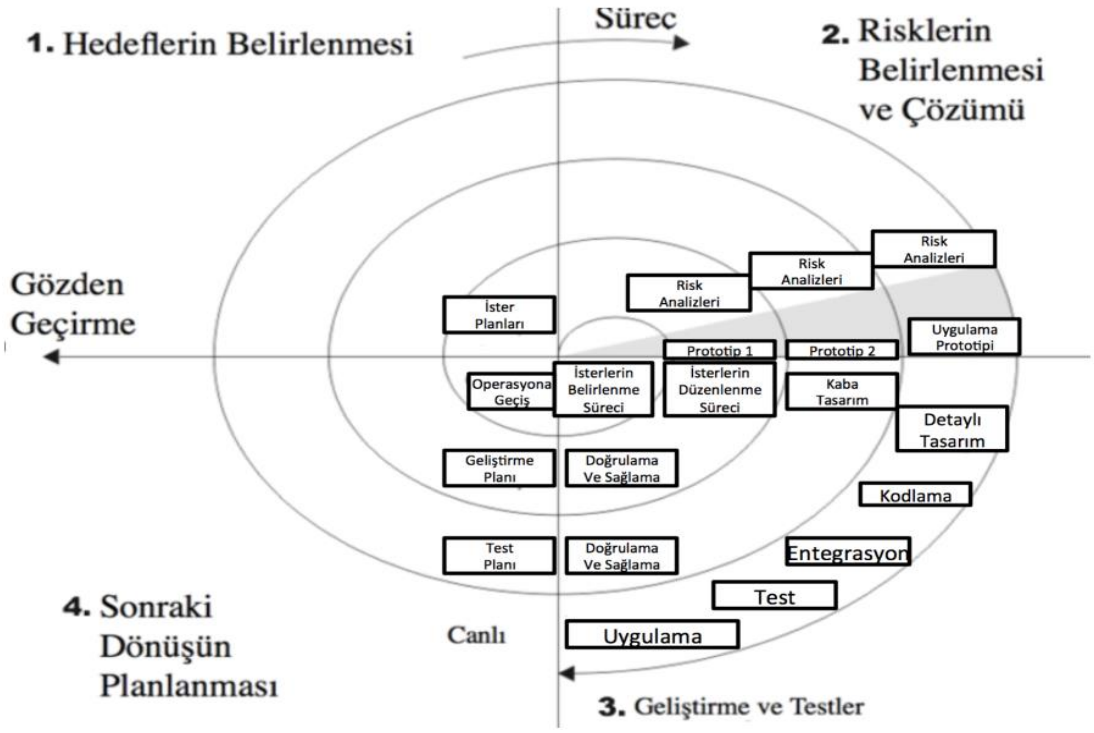
Spiral modeli ilk olarak 1988 yılında Barry Boehm tarafından tanımlanmıştır. Spiral modeli, Şelale modelinden farklı olarak belli ve düz bir akıştan ziyade iteratif bir yazılım modelidir. Tabi ki Spiral ilk iteratif model değildir; ancak neden iteratif bir süreç yürütüldüğünü açıkça gösteren ilk modeldir. Spiral modelinde iteratiflikten kasıt ise, her bir iterasyon bir tasarım hedefi ile başlar, standart yazılım süreçlerinin bazı aşamalarını geçip müşteri yorumu ve görüşü ile tamamlanır. Her bir iterasyonda amaç

iş değerini yakalamak olup, bir önceki iterasyonun sonuçlarından faydalanarak gelişen bir yapıda devam eder(Rossberg, 2008).

Temel prensipler;

- Risk değerlendirmesi, projeyi küçük parçalara ayırarak riskin minimize edilmesi, geliştirme süresinde kolay değiştirilebilir olmasının sağlanması gibi noktalara odaklanılmıştır.
- Proje yaşam döngüsündeki her aşama, proje kapsamında hazırlanan genel bir operasyonel kavram dokümanından her uygulamanın kendi kodlarını içeren spesifik dokümanına kadarki her aşama ve ürünün ve detaylarının her bir adımı boyunca kendi içinde bir ilerleme sağlar.
- Her döngüye paydaşların tespiti ile başlanır ve her döngüyü son bir kez gözden geçirerek ve taahhütle sonlandırılır. (Anaral, 2011)

Şekil 7: Spiral Model



(Şeker, 2015)

Basitçe Spiral modelinin aşamaları aşağıdaki gibidir.

1. **Gereksinim analizi:** Bu süreç müşteriden ihtiyaçların alınıp analizin yapılması sürecidir.

2. **Taslak Tasarım:** Bu süreçte alınan analize göre taslak tasarım yapılır.
3. **İlk prototip:** Oluşturulan taslak tasarıma göre yazılımın ilk prototipi oluşturulur. Tabi bu prototip sadece müşteriye yazılım ve çözüme yaklaşımı konusunda fikir vermek için geliştirilen basit bir prototiptir.
4. **İkinci prototip:** İkinci prototip aşağıdaki dört adımın sonunda oluşturulur.
   a) İlk prototipin güçlü ve zayıf yanları ve risklerinin değerlendirilmesidir.
   b) İkinci prototip için tekrar gereksinim analizinin yürütülmesi gerekir.
   c) Tekrarlanan gereksinim analizi sonucuna göre tekrar teknik tasarım yapılmalıdır.
   d) Oluşan teknik tasarıma göre yazılımın geliştirilmesi ve test edilmesi gerekir.
   e) Müşteri ile oluşturulan prototipin değerlendirilmesi gerekir. Bu aşamada eğer müşteri gidişattan memnun değilse daha en başta proje iptal edilebilir, böylece boşa gitme ihtimali olan birçok maliyet başlangıçta fark edilir. Şayet müşteri gidişattan memnunsa sürece devam edilir.
5. **n. Prototip:** Spiral modelde süreç bu şekilde iteratif adımlarla devam eder. Her bir adımda yukarıdaki beş adım uygulanır.
6. Her prototip sonunda yapılan müşteri değerlendirmeleri, müşteri oluşan yapıdan memnun olana kadar devam eder. Ne zaman müşteri ihtiyaçlarının karşılandığını düşünür ve sonuçtan tatmin olursa nihai ürün için çalışılmaya başlanır.
7. Nihai ürün, onaylanan son prototip üzerinde şekillenir ve canlı hayata alınır. (Anaral, 2011)

Görüldüğü gibi Spiral modelde açık bir değişiklik yönetimi, kapsamlı bir test süreci yoktur. Çünkü her iterasyonda müşteriden alınan analiz ile değişiklik istekleri toplanmış olur ve tekrar tekrar test edilmiş olur.

Spiral model, küçük kapsamlı projeler için uygun değildir ve büyük projelerde kullanılabilir.

2.4.1. Spiral Model Avantajları ve Dezavantajları

Avantajları

- Değiştirme gereksinimleri karşılanabilir.

- Prototiplerin kapsamlı bir şekilde kullanılmasına izin verir
- Gereksinimler daha doğru belirlenebilir.
- Kullanıcılar sistemi başlangıçta görürler.
- Geliştirme daha küçük parçalara bölünebilir ve daha riskli kısımları daha önce geliştirilebilir ve bu da daha iyi risk yönetimine yardımcı olur.

Dezavantajları

- Yönetimi daha karmaşıktır.
- Projenin sonu başlangıçta bilinmeyebilir.
- Küçük veya düşük riskli projeler için uygun değildir ve küçük projeler için pahalı olabilir.
- Süreç karmaşıktır.
- Spiral süresiz olarak sürebilir.
- Çok sayıda ara aşama aşırı dokümantasyon gerektirir.
  (SDLC-Spiral Model, 2016)

2.5. Prototip Model

Yazılımda prototipleme, istenilen yazılım kodlanırken, yazılımın tamamlanmamış bir prototipinin oluşturulma sürecindeki aktiviteleri içeren bir iskelet yapısıdır.

Prototip Model;

- Prototip, yazılımın limitli özelliklerine sahip modelidir.
- Prototip, gerçek yazılım uygulamasında kullanılan mantığı her zaman tutmaz ve tahmin için ek bir çaba gerektirir.
- Prototipleme, kullanıcıların geliştirici tekliflerini değerlendirip uygulamadan önce denemelerine izin vermek için kullanılır.
- Ayrıca, kullanıcıya özgü ve ürün tasarımı sırasında geliştirici tarafından dikkate alınmamış olabilecek gereksinimleri anlamaya yardımcı olur(SDLC-Software Prototype Model, 2016).

Bir yazılım prototipi tasarlamak için aşamalı bir yaklaşım örneği:

- **Temel Gereği Belirleme:** Bu adım, özellikle kullanıcı arayüzü açısından temel ürün gereksinimlerini anlamayı içerir. İç tasarımın karmaşık ayrıntıları, performans ve güvenlik gibi harici unsurlar bu aşamada göz ardı edilebilir.
- **İlk Prototip Geliştirme:** İlk prototip, çok temel gereksinimlerin gösterildiği ve kullanıcı arabirimlerinin sağlandığı aşamada geliştirilir. Bu özellikler tam olarak geliştirilen gerçek yazılımında olan şekilde çalışmayabilir ve geçici çözümler, geliştirilen prototipte müşteriye ürünün aynı görünümünü ve hissini vermek için kullanılır.
- **Prototip Gözden Geçirme:** Geliştirilen prototip müşteriye ve projenin diğer önemli paydaşlarına daha sonra sunulur. Geribildirim organize bir şekilde toplanır ve gelişme aşamasındaki ürünün iyileştirilmesi için kullanılır.
- **Gözden Geçirilmesi ve Prototip Geliştirmek:** Geri bildirim ve inceleme yorumları bu aşamada tartışılır. Müşteri tarafından zaman, bütçe kısıtlamaları, fiili uygulama ve teknik fizibilite gibi faktörlere dayalı olarak bazı görüşmeler yapılır. Kabul edilen değişiklikler yine geliştirilen yeni prototipe dahil edilir ve müşteri beklentilerine cevap verilene kadar döngü tekrarlanır(SDLC-Software Prototype Model, 2016).

Prototipler yatay veya dikey boyutlara sahip olabilir. Yatay prototip, ürünün kullanıcı arabirimini görüntüler ve dahili işlevlere konsantre olmadan tüm sistemin daha geniş bir görünümünü verir. Diğer taraftan düşey bir prototip, üründeki belirli bir işlevinin veya alt sisteminin detaylı bir şekilde hazırlanmasıdır.

Yatay ve dikey prototipin amacı farklıdır. Yatay prototipler, kullanıcı arabirimi seviyesi ve iş gereksinimleri hakkında daha fazla bilgi almak için kullanılır. Piyasada iş almak için satış demolarında bile sunulabilir. Dikey prototipler, alt sistemlerin tam işleyişinin ayrıntılarını almak için kullanılır ve doğal olarak tekniktir. Örneğin, belirli bir alt sistemdeki veritabanı gereksinimleri, etkileşim ve veri işleme yükleridir(SDLC-Software Prototype Model, 2016).

Yazılım Prototipleme Türleri

Endüstride kullanılan farklı yazılım prototipleri türleri vardır. Aşağıda, yaygın olarak kullanılan başlıca yazılım prototipleme türleri verilmektedir:

- **Fırlatılmış / Hızlı Prototipleme:** Fırlatılmış prototiplendirme, hızlı veya yakın uçlu prototipleme olarak da adlandırılır. Bu tip prototipleme, bir

prototip oluşturmak için minimum gereksinim analizi ile çok az çaba sarf eder. Gerçek gereksinimler anlaşıldıktan sonra, prototip oluşturulur ve gerçek sistem, kullanıcı gereksinimlerini çok net anlayarak geliştirilir.

- **Evrimsel Prototipleme:** Evrimsel prototipleme, aynı zamanda başlangıçta minimal işlevsellik ile gerçek fonksiyonel prototipler oluşturma üzerine kurulu olarak da bulunur. Geliştirilen prototip üst sistemin inşa edildiği gelecekteki prototiplerin kalbini oluşturuyor. Evrim prototipini kullanmak, yalnızca iyi anlaşılmış şartları prototip içerisine ekler ve gereklilikler anlaşıldıklarında eklenirler.
- **Artımlı Prototipleme:** Artımsal prototipleme, çeşitli alt sistemlerin çok işlevli prototiplerini oluşturmak ve daha sonra bütün mevcut prototipleri bütün bir sistem oluşturmak üzere bütünleştirmektir.
- **Aşırı Prototipleme:** Aşırı prototipleme, web geliştirme alanlarında kullanılır. Üç ardışık safhadan oluşur. İlk olarak, var olan sayfaların hepsiyle birlikte temel bir prototip html formatında sunulmaktadır. Ardından, veri işleme bir prototip hizmetler katmanı kullanılarak simüle edilir. Sonunda hizmetler uygulanmakta ve nihai prototipe entegre olmaktadır. Bu işlem aşırı prototipleme olarak adlandırılır ve işlemin ikinci aşamasına dikkat çekilir; burada tamamen işlevsel bir kullanıcı arabirimi, gerçek hizmetleri çok az dikkate alarak geliştirilir(SDLC-Software Prototype Model, 2016).

2.5.1. Prototip Model Avantajları ve Dezavantajları

Avantajları

- Uygulamadan önce bile ürüne kullanıcı katılımının artmasını sağlar.
- Sistemin çalışan bir modeli görüntülendiğinden, kullanıcılar geliştirilmekte olan sistemi daha iyi anlamış olurlar.
- Kusurlar çok daha erken tespit edilebildiğinden zamanı ve maliyeti düşürür.
- Daha iyi çözümler üretmek için daha hızlı kullanıcı geribildirimi mevcuttur.
- Eksik işlevsellik kolayca tespit edilebilir.
- Kafa karıştırıcı veya zor fonksiyonlar tespit edilebilir.

Dezavantajları

- Prototipe aşırı bağımlılık nedeniyle yetersiz ihtiyaç analizi riski vardır.

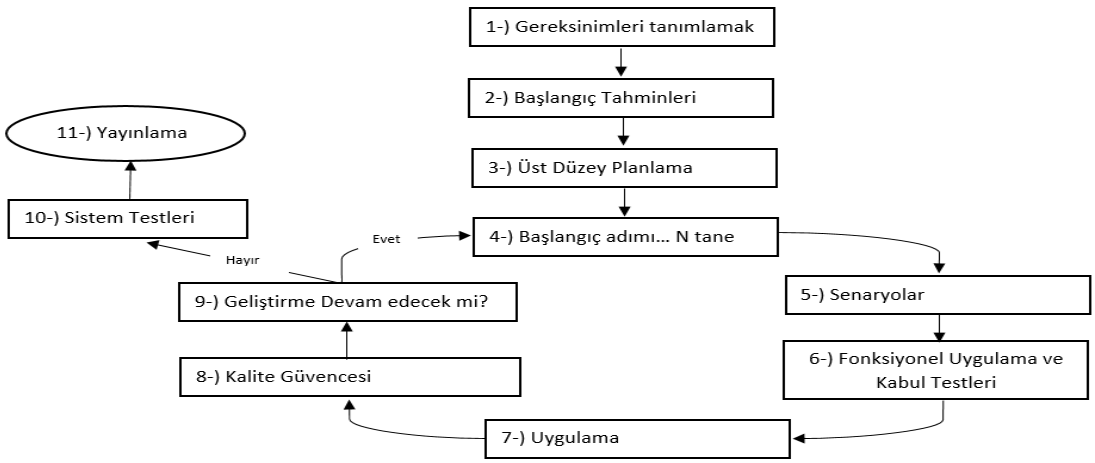
- Kullanıcılar, prototipler ve gerçek sistemler konusunda karışıklık yaşayabilirler.
- Pratik olarak, sistemin kapsamı orijinal planların ötesine geçebileceğinden, bu metodoloji sistemin karmaşıklığını artırabilir.
- Geliştiriciler, gerçek sistemi oluşturmak için teknik olarak mümkün olmasa bile mevcut prototipleri tekrar kullanmayı deneyebilirler.
- Prototip oluşturmak için harcanan çaba uygun şekilde izlenmezse çok fazla olabilir.(SDLC-Software Prototype Model, 2016)

2.6. Agile(Çevik) Model

Çevik modelleme, yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve dokümantasyonunu yapmaya yönelik pratiğe dayalı yöntemlere verilen genel addır.

Çevik modellemenin başlıca özelliği veri modelleri ve ara yüzü modelleri gibi modelleme tekniklerinin neler olduğunu ve bunların ayrıntılarını söylemek yerine bu tekniklerin nasıl uygulanması gerektiğini söylemesidir. Mesela; yapılan projelerin test edilmesi gerektiğini belirtse bile nasıl test hazırlanacağına değinmemesi gibi. O sadece bir yöntemler biçimidir ve bir projenin etkili, hızlı bir şekilde ortaya çıkarılmasında, müşterinin ihtiyaçlarını karşılamasında ve aynı zamanda da her türlü değişikliğe kolayca adapte olabilmesinde geliştiricilere yol gösterir(Ocak, 2012).

Şekil 8: Agile Model



(Çevik Modelleme ve Çevik Yazılım Geliştirme, 2017)

Çevik geliştirme teknikleri, süreç ve belgeleme yerine doğrudan yazılımın kendisine yoğunlaşan ve yinelemeli geliştirmeye dayanan tekniklerdir. Teknoloji ve kullanıcı beklentilerindeki sürekli değişime bir cevap olarak doğmuştur.

Çevik yazılım geliştiricileri tarafından ortak bir standart oluşturulması için 2001 yılında çevik yazılım geliştiriciler bir araya gelerek çevik yazılım geliştirme manifestosunu ve prensiplerini yayınlamışlardır. Bu manifesto ile çevik yöntemlerin özellikleri çevik yazılım geliştiricileri tarafından ortak olarak açıklanmıştır. Bu manifestoda;

- Süreçler ve araçlardan ziyade bireyler ve etkileşimlere
- Kapsamlı dokümantasyondan ziyade çalışan yazılıma
- Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye ve değişime değer vermeye
- Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine bağlı kalmak dile getirilmiştir(Çevik Yazılım Geliştirme Manifestosu, 2017).

Çevik yazılımın prensipleri 12 madde olarak aşağıda belirtilmiştir. Bunlar:

- En önemli önceliğimiz değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
- Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.
- Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
- İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar.
- Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir.
- Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.

- Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.
- En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

  (Çevik Yazılım Geliştirme Manifestosu, 2017)

2.6.1. Agile Model Avantajları ve Dezavantajları

Avantaları

- Değişim ve kolayca uyum sağlama esasına dayandığı için küçük ve orta çaplı projeler için uygundur.
- İşlevsellik hızla geliştirilebilir ve gösterilebilir.
- Kaynak gereksinimleri minimumdur.
- Sabit veya değişen gereksinimler için uygundur.
- Erken kısmi çalışma çözümleri sunar.
- Değişen ortamlar için iyi modeldir.
- Minimal kurallar olduğu için dokümantasyon kolayca uygulanır.
- Genel olarak planlanmış bağlamda eşzamanlı geliştirme ve teslimat yapılmasını sağlar.
- Yönetmesi kolaydır ve geliştiricilere esneklik kazandırır.

Dezavantajları

- Karmaşık bağımlılıkların kullanımı için uygun değildir.
- Sürdürülebilirlik için fazla risk vardır.
- Müşteri ile etkileşimine bağlı olarak, müşteri açık değilse proje ekibi yanlış yönde projeye devam edebilir.
- Dokümantasyon minimum seviyede oluşturulduğundan bireysel bağımlılık çok yüksektir.
- Dokümantasyon eksikliği sebebiyle yeni ekip üyelerinin projeye dahil edilmesi zor olabilir.

  (SDLC-Agile Model, 2017)

Agile Modelinin kullanılmasının en uygun olduğu durumlar şunlardır:

- Projenin yazılım evresinde müşteriden gelebilecek talep değişikliklerinin tahmin edilemez olması
- Projenin parçalarının önce tasarlanıp ardından hemen geliştirilmesinin gerekmesi ve önceden ne yapılacağını, detaylı yol haritasını ve tasarımını tahmin etmenin çok güç olması
- Analiz, tasarım ve test etme süreçlerinin ne kadar zaman alacağının önceden bilinememesi
- Yazılım ekibinin birlikte çalışmak, hiyerarşiye önem vermemek, sağlam iletişim kurmak gibi özelliklere sahip olması(Ocak, 2012)

3. AGİLE (ÇEVİK) GELİŞTİRME YÖNTEMLERİ

Çevik metotları tam bir yazılım süreci değildir ama kapsamlı yazılım geliştirme yöntemlerini tamamlayıcı niteliktedir. Başlıca Agile(çevik) geliştirme yöntemleri şunlardır:

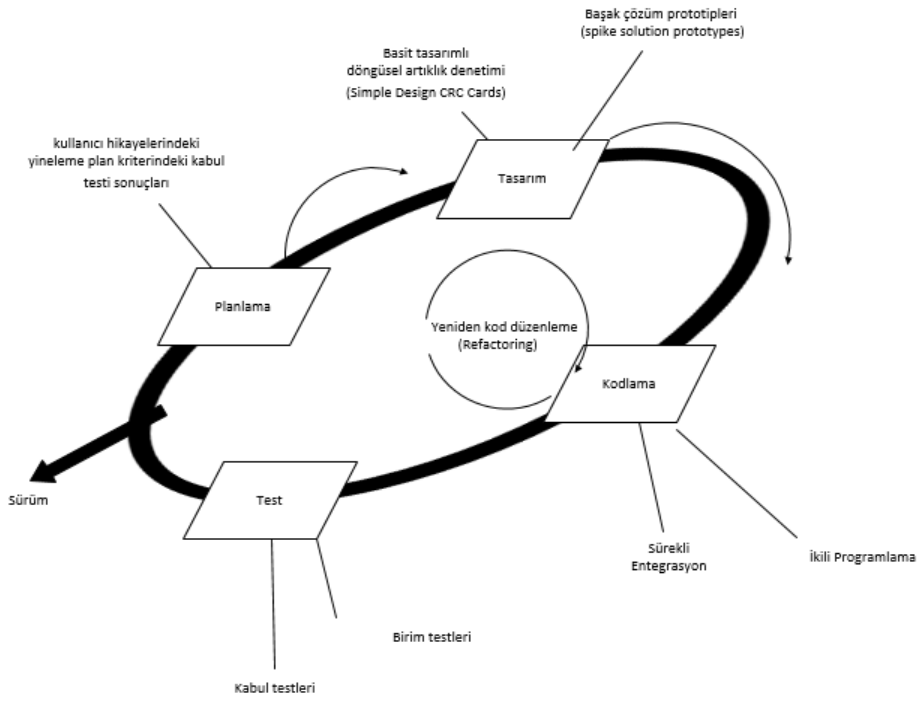
1. Sınırsal Programlama(Extreme Programming-XP)
2. Scrum
3. Kristal Yöntem
4. Dinamik Sistem Geliştirme Yöntemi(Dynamic Systems Development Method – DSDM)
5. Özellik Güdümlü Geliştirme (Feature-Driven Programming)

3.1. Sınırsal(Uç) Programlama (Extreme Programming-XP)

Sınırsal Programlama(XP), 3 kişi tarafından ortaya atılmıştır. Bu kişiler: Ward Cunningham, Kent Beck, Ron Jeffries'dir. Sınırsal programlama, Kent Beck tarafından 1999 yılında yayınladığı kitap ile tüm dünyaya duyurulmuştur ve zamanla kabul görmeye başlamıştır. Günümüzde geniş bir uygulama alanına sahiptir. Sınırsal programlamanın en önemli özelliklerinden bir tanesi, sahip olduğu pratikler hakkında uygulayıcılarından sürekli geri dönüşümlerinin olmasıdır. Bu sebeple zaman içerisinde sürekli gelişmekte ve iyileşebilmektedir. Böylelikle sınırsal programlamanın bu gelişiminde sadece bir kişi değil dünyanın çeşitli yerlerindeki yazılım organizasyonlarından birçok uygulayıcı katkı sahibi olmaktadır.

Sınırsal programlama basitlik, haberleşme, geribildirim, cesaret ve saygı temelleri üzerine kurulmuş olan bir yazılım geliştirme metodudur. Tüm takım üyeleri sınırsal programlama tarafından önerilen basit pratikleri, yani nelerin nasıl yapılması gerektiğini kullanırlar. Direkt müşteriden, takım üyelerinden ve testlerden alınan yeterli geribildirim ile takım işin neresinde olduğunu görür. Pratikleri kendisine uygun bir şekilde ayarlamaya çalışır. Buradan anlaşıldığı gibi sınırsal programlamanın önerdiği pratikler mutlaka uyulması gereken kurallar değil proje ekibinin kendinden bir şeyler katabileceği pratiklerdir.

Şekil 9 :Sınırsal Programlama Modeli



(Pressman, Maxim, 2015)

Sınırsal programlama diğer sistemlerden farklıdır. Küçük sürümler vardır ve bu parçalar bir araya getirildiğinde ana resme erişilir. Bu, geleneksel yaklaşımdan önemli bir farktır.

Sınırsal programlama aşamaları:

1. Planlama

• Kullanıcı hikâyeleri yazılmıştır.

• Yazılımın sürümünün yayınlanması için zaman planlaması yapılır.

• Küçük sürümler genellikle çıkarılır.

• Proje hızı sürekli ölçülür.

• Proje küçük parçalara bölünmüştür.

• İterasyon planlaması iterasyon ile başlar.

• Operasyon sürekli insanların etrafında meydana gelir.

• Bir sorun varsa, mutlaka düzeltilir.

2. Tasarlama

• Her şey basittir.

• Sistemi tanımlayan bir metafor seçilmiştir.
• Tasarım riskini azaltmak için hızlı çözümler bulunur.
• Doğru zaman olmadan herhangi bir işlev eklenmez.
• Herhangi bir zamanda ya da ihtiyaç duyulduğunda yeniden düzenleme yapılabilir.

3. Kodlama

• Müşteri her zaman kullanılabilir.
• Kodlama, kodlama standartlarında yapılır.
• Birim testleri kodlama sırasında sürekli kontrol edilir.
• Tüm projeler çift geliştiriciler tarafından kodlanır.(Çiftli Programlama)
• Aynı zamanda yalnızca bir kişi kod üzerinde değişiklikler yapabilir.
• Kodlar çoğu kez entegre edilmiştir.
• Sahiplik kodlamada herkese aittir. (Toplu kod sahipliği)
• Optimizasyon son noktaya bırakılmıştır.
• Kodlamada fazla mesai yapılmaz.

4. Test

• Tüm kodların birim testi vardır.
• Bir kod yayınlanmadan önce, tüm birim testlerini kesinlikle desteklemelidir.
• Bir hata ile karşılaşıldığında yeni testler hazırlanır.
• Kabul testleri genellikle müşteri tarafından hazırlanır ve takip edilir. Program parçacıklarına puan verilir ve bu program parçacıkları yayınlanır(Bakan, 2007).

Sınırsal programlama aşamaları yukarıdaki gibi oluşur ve bu tekniğin bize söyledikleri şu şekildedir:

- Test yazılımı, test edilen parçalardan önce yazılır ve mümkün olduğunca her şey test edilir. Bu testler üst, üste işletilebilen türden olmalıdır.
- Programcılar yalnız çalışmaz, bütün kod ikişerli takımlar tarafından yazılır.
- Sistem mimarisi için hiç zaman ayrılmaz. Yazılım mimari yapısı önceden tek başına yazılmaz. Bütün iş bölümü, müşteriye anlamlı özellikler üzerinden yapılır, mimariye, yavaş yavaş özellikler eklenirken arka planda inşa edilir.
- Dokümantasyon yazılmaz. Program yazmaya hiç yararı olmayan bu kâğıt parçaları için zaman harcanmaz.

- Müşteri her zaman programcılar ile aynı yerde durur, soru sormak için sürekli mevcut bulunmalıdır.
- Programcılar iş sürecini ilgilendiren kararlar alamazlar, müşteriler teknik kararlar alamazlar. Hangi özelliğin önce, hangisinin sonra geleceği iş sürecini ilgilendirir. Özelliklerin ne kadar sürede kodlanacağı, nasıl kodlanacağı programcılara kalmıştır.
- Özellikler, teker teker orta boy kartların üzerine yazılır, fakat bu kartlar atılabilir, yerine yenisi yazılabilir. Müşteri istediği zaman yeni bir kart ekleyip çıkartabilir. Tek kurala uymak kaydıyla: 3 haftalık dilimlerde yazılacak kart sayısı, takımın iş hacmi ötesinde olmamalıdır.(Koçer, 2007).

3.1.1. Sınırsal Programlama Değerleri

Sınırsal programlama beş temel öğeden oluşur. Bunlar: cesaret, iletişim, saygı, basitlik, geri bildirimdir.

1. Cesaret

Sınırsal programlamada proje daha iyi çalışabilmesi için kolaylıkla değiştirilebilir. Çünkü testler çok sık ve sağlam bir şekilde yapılır. Beğenilmeyen bir durum oluştuğunda o modül bırakılır tekrar yeniden yazılır. Bu şekilde yazılım üzerinde değişiklik korkmadan yapılır. Bu durumlar sistemin geliştirilmesi için bir tehdit içermez. Aynı zamanda müşteri ve yazılımcı aynı ortamda bulunur ve müşteri taleplerine hemen cevap verilir bu şekilde müşteriye de güven sağlanmış olur. Kodlamada bilgi eksikliği olması bir krize yol açabilir. Bu durumda çiftli programlama yöntemi uygulandığından yazılımcılar birbirlerinin kodlama hakkında bilgilerini paylaşırlar ve bu şekilde tüm sistem üzerinde bilgi sahibi olunur. Kişiye bağımlılık azalmış olur. Bu şekilde kod yazmak daha cesaretli olunmasına yol açar.

2. İletişim

Sınırsal programlama tüm proje ekibi üyelerinin bir araya gelerek çalışmasını sağlar. Bu şekilde ekip üyeleri birbirleri ile bilgilerini paylaşarak sistemin geliştirilmesine katkı sağlarlar. Yazılımcı ile birlikte müşteride ekibin bir parçasıdır. Müşteri taleplerinin hazırlanması, bu taleplerin önceliklerinin belirlenmesi geliştirilecek sistemde hangi özelliklerin planlandığı konusunda proje ekibi bilgilendirilir. Analizi yapan kişiler müşterilerin taleplerinin belirlenmesine yardım

eder. Yazılımın son kullanıcı testinin yapılabilmesi için ve son kullanıcı deneyimlerinden faydalanmak için proje ekibine son kullanıcı dahil edilebilir. Test edecek personel yine proje ekibinin bir parçasıdır. Bu şekilde müşteri taleplerinin test edilmesine yardımcı olur. Proje yöneticisi ve farklı yöneticilerde proje ekibinde olabilir. Bu şekilde farklı rollerde farklı deneyimlerde birden fazla kişinin bir arada çalışması için iletişim çok önemli bir role sahip olur.

3. Basitlik

Geliştirilen sistemlerde yazılımın her bir parçası mümkün olduğunca basit bir şekilde yapılmalıdır. Sınırsal programlamanın en temel özelliği sadeliktir. Basit bir tasarımla başlanır ve testler yapılarak bu sade tasarım üzerinde devam edilir. Talep edilen ihtiyaçlara göre tasarım şekillenir ek özellikler eklenerek tasarım karmaşıklaştırılmaz ve bu sayede risk azaltılmış olur. Yeni özellikler talep edilirse daha sonra bu talepler için tasarım gerçekleştirilir. Her adım sadece ihtiyaç olduğu için yapılır fazladan bir şey yapılmaz. Yazılım her zaman bir sonraki yapılandırılma ya da sürüm değişimi için hazırdır. Sınırsal programlama tasarımlar için uzun zaman beklenmez.

4. Saygı

Proje ekibindeki personeller yaptıkları işe yeterince önem vermez ve diğer ekip üyelerine saygı duymazlarsa bir projenin başarılı olması mümkün olmaz. Bu yüzden projedeki personeller birbirlerine ve yaptıkları işe saygı göstermeleri gerekir. Bu şekilde davranmakta projenin başarılı olmasına katkı sağlar.

5. Geribildirim

Testler, sınırsal programlama için en önemli geri bildirim aracıdır. Yazılan en küçük kod parçacığı bile test edilerek yazılım en başında kontrol altında tutulur. Bu işleme birim testi denir. Daha sonra birimler bir arada çalışacağı için onları birleşik olarak test etmek gerekir bu da birleştirme testidir. Birleştirilen birimler bir araya geldiğinde sistemi oluşturur ki buda sistem testi olarak adlandırılır. Özellikle birim testleri çok önemlidir ve sınırsal programlama bu testlerin kesinlikle yapılmasını önerir. Bir başka geribildirim aracı bizzat yazılan koddur. Sınırsal programlamada "CodeSmell" (Kod koklama) yöntemi, yazılan kod ile alakalı bir problem varsa, izleri takip edilerek, yani kod incelenerek, problemin bulunması için başvurulan bir yöntemdir (Koçer, 2007).

3.1.2. Sınırsal Programlama Prensipleri

XP değerlerinden yola çıkarak on beş XP prensibi oluşturulmuştur. Bunlar:

1. Basitliği Tercih Etmek (Assume Simplicity)

Sadelik projenin hızlı bir şekilde geliştirilmesini sağlar. Bu yapı geri bildirimin hız bir şekilde oluşmasını sağlar. Basit çözümler daha kolay anlatılır. Gereksinimler en sade şekli ile karşılanır. Yeni gelen talepler daha sonraki aşamalarda eklenir.

2. Öğrenmeyi Öğretmek (Teach Learning)

Sınırsal programlamada proje ekiplerindeki yazılımcılar arasında ast üst farkı yoktur. Ekipteki herkes eşittir. Tecrübe sahibi olan yazılımcılar deneyimlerini paylaşarak proje ekibindeki diğer kişilerin bilgi seviyelerini arttırırlar ve zaman içinde proje ekibindeki yazılımcılar yaklaşık aynı teknik seviyelere gelirler. Bu şekilde öğrenme sistemin geliştirilmesine katkı sağlar.

3. Hızlı Geri Bildirim (Rapid Feedback)

Sürekli geri bildirim olması projeyi iyileştiren bir süreçtir. Bu şekilde hataların sayısı azaltılır ve sağlıklı bir siste inşa edilmiş olur.

4. Kazanmak İçin Oynamak (Play to win)

Sınırsal programlama her zaman projeyi sonlandırmak ister. Bu şekilde müşteri memnuniyetini sağlamak ister. Proje ekip üyelerini motive ederek yazılım projeni sonlandırmaya çalışır.

5. Kaliteli İş (Quality Work)

Sınırsal programlamada projelerde kaliteli kod geliştirme yapılabilmesi için gerekli ortam sağlanmalıdır. Çalışma ortamının kalitesi yazılımcının daha rahat ve daha kaliteli kod yazmasını sağlar. Bu şekilde müşterinin talepleri istenilen şekilde karşılanmış olur.

6. Arttırımsal Değişiklik (Incremental Change)

Sınırsal programlama projeleri sade bir şekilde uygulamaktadır fakat proje büyüdükçe programın yapısı sade tasarımlardan oluşsa bile karmaşık hale gelecektir. Sistem üzerinde yapılacak değişikler fark edilmeyecek bir bölümde hatalara sebep olabilecektir. Bu hataların oluşabilme ihtimaline karşı yapılacak değişiklikler küçük

çapta olmalıdır. Bu şekilde küçük değişiklikler yapılarak sistem arttırımsal bir şekilde geliştirilmelidir.

7. Somut Denemeler (Concrete Experiments)

Bu yöntemde her zaman kontrol için sürekli testler vardır. Analiz aşamasında belirlenen ihtiyaçlar için birim testleri, entegrasyon testleri ve sistem testleri yapılarak geliştirilmiş olan yazılım projesinin tüm aşamalarının kontrolü sağlanmış olur. Bu şekilde somut denemeler yapılarak sistemin sorunsuz bir şekilde tamamlanması sağlanır.

8. Az Baslangıç Yatırımı (Small Initial Investment)

Sınırsal programlamada projeye başlamak için fazla yatırım yapılmaz. Başlama maliyeti ne kadar az seviyede tutulursa projenin iptal edilmesi halinde kayıp edilen miktarda düşmüş olur. Proje küçük bir yatırım ile başlanarak sistemin geliştirilmesi için daha önemli görevlere odaklanılır. Örneğin proje başlandığında en son sürüm Oracle veri tabanı kullanıcı araçları satın alınarak proje ekibini veri tabanı ve araç hakkında eğitim verilir. Bu şekilde fazla maliyetli bir yol seçilmiş olur bunun açık kaynak kodlu bir veri tabanı seçilerek yazılımcıya bununla ilgili bilgilendirme yapılırsa maliyeti düşük bir şekilde projeye başlanmış olur. Gerekli araçlar proje süresinde gerekli olduğu zaman satın alınmalıdır(Extreme Programming Nedir?, 2017).

9. Takımın İçgüdülerini Kullanmak ve Onlara Karşı Koyma (Work with people's instincs, not against them)

Proje ekibindeki bireylerin projenin gelişim safhası aşamasında projenin gelişimi ile ilgili tahmin ettikleri içgüdüleri vardır. Bu şekilde proje takımındaki bireylerin ortak iç güdülerini dile getirerek bir şeyin yanlış gittiğini söylerlerse bu duruma kayıtsız kalınmamalıdır. Projenin gelişmesine katkı sağlanması için takımın sesine kulak vermek gerekir.

10. Açık ve Samimi İletişim (Open, honest Communication)

Sınırsal programlama takım üyelerinden oluşur. Bir takımın başarılı olması içinde ekip üyelerinin bir birleri ile açık samimi bir şekilde iletişim kurması çok önemlidir. Çünkü ekip üyeleri deneyimlerini paylaşması ile sistemin iyileşmesine

yardımcı olurlar. Fakat ekip arasında sağlam bir iletişim olmazsa projenin başarısız olup tamamlanamaması gibi durumlar ortaya çıkabilir.

11. Değişimi İstemek (Embracing Change)

Değişimi isteme ilkesi, değişikliklere karşı olmak değil onları benimsemeye yöneliktir. Yenilik ve değişimlere açık olmak önemlidir. Bu şekilde sistemin geliştirilmesine katkı sağlanmış olur.

12. Sürecin Ortam Şartlarına Adapte Edilmesi (Local Adaptations)

Sınırsal programlamanın bahsettiği herşeyi birebir yerine getirmek zordur. Her proje ekibinin kendisine göre farklı bir yöntemi vardır. Amaç bir projenin hızlı bir şekilde sorunsuz olarak tamamlanmasıdır. Sınırsal programlama uygulanırken bu süreç içinde değişiklikler yapılması verimliliği arttırdığı düşünülüyorsa bu değişiklik yapılmalıdır.

13. Doğru Ölçüm (Honest Measurement)

Projenin gelişim safhalarını kontrol etmek için ölçümler yapılmalıdır. Testler bu ölçümleri yapmak için kullanılabilir. Bu şekilde projenin doğru bir şekilde ilerlemiş olduğundan emin olabiliriz.

14. Sorumluluk Üstlenmek (Accepted Responsibility)

Proje boyunca proje ekibi üyeleri işlerle ilgili sorumlulukları birinin kendilerine söylemesi yerine kendileri bu sorumlulukları üstlenmelidir. Çünkü bir işi severek yapmak motivasyonu arttırır bunun sonucunda başarı gelir. Bu şekilde bir ilerlemenin olması için sorumlukların kişilere verilmesi değil kişilerin sorumlulukları üstlenmesi gerekir.

15. Az yükle Yolculuk Yapmak (Travel light)

Projeyi hızlı bir şekilde geliştirebilmek için fazla bir yük bulundurulmamalıdır. Takım çalışmasını kolaylaştıracak araçlar tercih edilmelidir.

3.1.3. Sınırsal Programlama Rolleri

Sınırsal programlamada roller müşteri, yazılım geliştirme personeli, proje yöneticisi, test personeli ve uzmandan oluşur.

1. **Müşteri**

Projenin gereksinimlerini kullanıcı hikâyeleri oluşturan kişidir. Bu şekilde gereksinimleri yazılım geliştirme personellerine bildirirler. Gereksinimlerde anlaşılmayan durumlarda yazılım geliştirme personeline destek olurlar. Uygulama tamamlandıktan sonra son kullanıcı testlerini yapma sorumlulukları vardır.

2. **Yazılım Geliştirme Personeli**

Yazılım geliştirme personeli proje için sistem analizi, sistem tasarımı, birim testleri ve uygulamanın geliştirilmesinden sorumludur. Bu kişi her geliştirdiği program parçacığını test eder ve sisteme entegre edilmesini sağlar.

3. **Proje Yöneticisi**

Proje yöneticisi proje geliştirme sürecinde yapılan işlerle ilgili koordinasyon ve toplantılar düzenler ve proje aşamalarını takip eder. Yazılım geliştirme personeline görev atayamaz.

4. **Test Personeli**

Test personeli geliştirilen projenin oluşturulmuş kullanıcı hikâyelerine göre testini yapar. Hata ve eksiklik durumunda yazılım geliştirme personeline geri bildirimde bulunur.

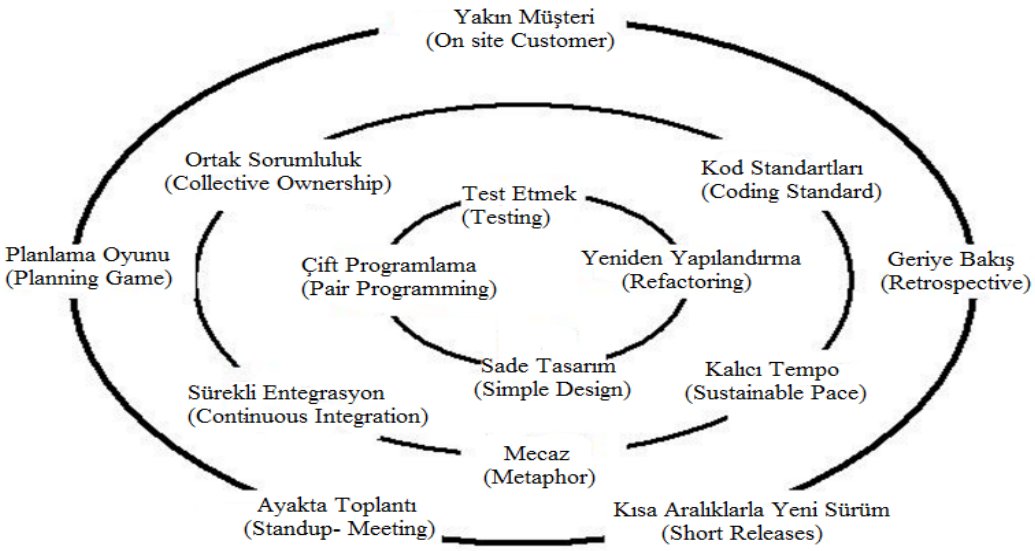
5. **Uzman**

Uzman sınırsal programlama modelinin nasıl uygulanmasını gerektiğini ve modelin süreçlerinin uygulanmasını takip eder. Bu modelin dışına çıkılmasına izin vermez.

3.1.4. Sınırsal Programlama Teknikleri

Sınırsal programlama, 16 sınırsal programlama tekniği ile desteklenmektedir. Bu teknikler programcıların sınırsal programlama değerlerini ve prensiplerini uygulamada yardımcı olur. Bu teknikler; yakın müşteri, planlama oyunu, ayakta toplantı, kısa aralıklarla yeni sürüm, geriye bakış, ortak sorumluluk, sürekli entegrasyon, mecaz, kalıcı tempo, kod standardı, test, çiftli programlama, sade tasarım ve yeniden yapılandırmadan oluşur. Bir projede sınırsal programlama pratiklerinin hepsi birden uygulanmayabilir. Projenin durumu ve insan kaynağına göre bu pratiklerin yararlı olacağı düşünülenler uygulanabilir.

Şekil 10: Sınırsal Programlama Teknikleri



(Extreme Programming Nedir?, 2017).

1. Ayakta Toplantı (Standup-Meeting)

Proje ekibinin günlük olarak gerçekleştirdiği toplantılardır. Bu toplantının süresi en fazla 15 dakika olmalıdır. Bu toplantılar ayaküstü yapılan bir toplantıdır. Toplantıda projenin amacı ve projenin gelişimi ile ilgili bilgi alış verişi yapılır.

2. Geriye Bakış (Retrospective)

Proje ekibi belli süreçlerin sonunda proje başlangıç aşamasından itibaren projeyi gözden geçirirler. Karşılaşılan problemler incelenir ve problemlerin oluşmaması için önlemler alınır. Bu değerlendirme aşaması birkaç gün sürebilir.

3. Planlama Oyunu (Planning Game)

Sınırsal programlamanın ana planlama sürecine Planlama Oyunu denir. Bu süreçte, projedeki her adım için genellikle haftalık olarak yapılan toplantılardır. Bu toplantılarda yayınlanacak sürümlerin ne zaman çıkarılacağı ve kullanıcı hikâyelerinin taleplere dönüştürülüp bu sürümler içinde olup olmamasına karar verilir. Bu aşamanın temel amacı projenin tamamlanmasını sağlamaya çalışmaktır. Basit bir şekilde proje yönlendirilir. Müşteriler ve yazılım geliştiriciler bu toplantıların katılımcıları arasındadır.

4. Kısa Aralıklarla Yeni Sürüm (Short Releases)

Sınırsal programlamada proje için yapılan değişiklikler için sık sık her küçük işlev için sürüm çıkarılır. Bu sürümler müşteri tarafından test edilir. Taleplerinin karşılanıp karşılanmadığı ile ilgili kontrol yapılmış olur. Talepler karşılanmamışsa bir sonraki sürümde bu talepler karşılanır. Bu şekilde proje adım adım kontrollü bir şekilde gelişme sağlar.

5. Programcıya Yakın Müşteri (On-site Customer)

Sınırsal programada proje ekipleri müşteri ile sürekli ile iletişim halindedir. Müşteri aynı zaman proje ekibi içinde yer alır. Yazılım geliştirme esnasında kullanıcı hikâyeleri ile ilgili bir belirsizlik olduğunda hemen aynı ortamda bulunan müşteri ile iletişime geçilir. Bu şekilde belirsizlikler hız bir şekilde ortadan kaldırılır. Proje daha sağlıklı bir şekilde gelişir. Sınursal programlamanın en önemli özelliği olan iletişim burada sıkı bir şekilde uygulanır.

6. Mecaz (Metaphor)

Mecaz, bir sistemin mantıksal mimarisini, yazılım geliştiricilerinin ve müşterilerin zaten tanıdıkları bir şekilde açıklamanın bir yoludur. Mecaz sayesinde hem müşterilerin hem de geliştiricilerin erişebildiği dilde projenin tartışılması kolaylaşır.

Örneğin bir alış veriş sistemi yazılımında mecaz olarak alışveriş sepeti kullanılabilir. Alışveriş sepetini duyan her programcının aklında, bir alışveriş sisteminin programlanması gerektiği fikri doğar.

7. Ortak Sorumluluk (Collective Ownership)

Sınırsal programlama projelerinde yazılım geliştirme süresince kaynak kodlarda bireysel sahiplik yoktur tüm proje ekibi kodlar üzerinde ortak sorumluluk sahibidirler. Bu şekilde herhangi bir kod bloğundan sorumlu yazılımcı yoktur herkes o kod parçacığı hakkında bilgi sahibidir gerektiğinde o kod bölümüne edebilir. Bu şekilde proje esnasında değişiklik istenen kod bölümleri rahatlıkla değiştirilebilir ve geliştirme süreci aksamamış olur.

8. Sürekli Entegrasyon (Continuous Integration)

Proje geliştirme aşamasında yapılan değişiklikler hemen test edilerek sisteme eklenir. Bu şekilde sık sürüm yayınlanmasıyla sistemin genelindeki tüm değişikler yazılım geliştirme ekibi tarafından görülebilecektir. Bu şekilde sistem sürekli entegrasyon ile düzgün bir şekilde gelişir.

9. Kod Standartları (Coding Standards)

Projede yazılım geliştirilirken standart kod yazılması için kod yazma kurallarının tarif edildiği bir doküman hazırlanır. Bu dokümanda isimlendirme formatlarının nasıl yapılacağı tarif edilir. Bu şekilde yazılım geliştiriciler ortak bir dilde aynı şekilde kod geliştirmiş olur. Projeye sonrada katılacak olan personelinde projeye entegre olması kolaylaşacaktır.

10. Kalıcı Tempo (Sustainable Pace)

Sınırsal programlama projelerinde yazılım geliştiricileri haftalık belli bir saat çalıştırılır. Bu sürelerin geçilmemesine dikkat edilmelidir. Çünkü yazılımın doğası gereği esnek bir şekilde çalışan yazılımcının moral ve motivasyonu düşer. Bunun sonucu olarak da normal sürede yapması gereken işleri daha geç bitirmeye başlar. Bunun sonucu olarak projenin gidişatı olumsuz etkilenir proje zamanında bitirilemeyebilir.

11. Test Etmek (Testing)

Geliştirilecek olan sistemin her bir adımda test edilmesi gerekir. Bu şekilde daha kaliteli ve daha sorunsuz yazılımlar ortaya çıkar. Testler yapılırken her bir program parçacığının yazılımcılar tarafından birim testleri yapılır. Daha sonra sistemin birleştirilmesinde entegrasyon testleri yapılır. Son olarak son kullanıcılar tarafından kabul testler yapılır. Bunların sonucun yazılım canlı sistemde yayınlanır.

12. Sade Tasarım (Simple Design)

Proje geliştirme safhasında sınırsal programın değerlerinden olan basitlik tasarım aşamasında kullanılır. Tasarımlar mümkün olduğunca yalın halde yapılır. Bu şekilde tasarımların değiştirilebilmesi ve genişletilebilmesi kolay olur. Sistemin bütününde anlaşılır olmasına yardımcı olur. Sistem karmaşıklığını azaltır. Basit tasarımlar sonucu sistem daha hızlı bir şekilde ilerler.

13. Yeniden Yapılandırma (Refactoring)

Proje geliştirme aşamalarında tasarımlar üzerinde yapılan hatalar sistemin yapısını bozabilir. Bu hatalar tespit edildiğinde tasarımların yeniden yapılandırılması gerekir. Yazılımcılar tarafından hazırlanmış olan birim testleri yapılan bu değişiklikleri etkilerini kontrol eder.

14. Çiftli Programlama (Pair Programming)

Sınırsal programlama projelerinde 2 yazılımcı bir bilgisayarda birlikte çalışır. Bu şekilde yazılım geliştiren personeller birbirlerinin deneyimlerinden faydalanırlar. Kod yazımı esnasında birbirlerini gözlemledikleri için bu durum yazılım kod kalitesinin artışına olanak sağlar.

3.1.5. Sınırsal Programlamanın Avantajları ve Dezavantajları

Avantajları

Sınırsal programlama, yazılım geliştirme aşamasında verimlilik kazandırmayı ve proje paydaşlarından gelen talepleri hızlı bir şekilde cevap vererek paydaş memnuniyetini arttırmayı amaçlar. Sınırsal programlamanın avantajları sağlamlık, esneklik, maliyet tasarrufları, daha az risk ve daha iyi genel memnuniyeti içerir.

Sınırsal programlama, geleneksel programlama yaklaşımından uzaklaşılmasını işaret eder. Proje ekibi ve paydaşlarının süreçler üzerindeki etkileşimlerine, müşteri ile görüşme sonucunda oluşan işbirliğine, bu doğrultuda bir plan hazırlanmasına ve bu plana uygun değişimlere ayak uydurmayı amaçlar.

- **Sağlamlık**

Sınırsal programlama sadelik ve kolaylığı amaçlar. Tasarım, küçük küçük parçalardan ve tekrarlanan döngüler sonucu gelişerek büyür. Bu tekrarlar sonucu küçük parçalara birleştirilerek nihai ürün oluşturulur. Bu şekilde mümkün olan küçük parçalar halinde geliştirildiği için bu parçaların test edilmesi ve kod gözden geçirmesi daha rahat yapılır. Her aşamada testler yapılır, ardından müşteri onayı alınmak için müşteri testleri yapılır. Sadece müşteri taleplerinin gerçekleştirilmesi amaçlanır. Bu şekilde yazılım mümkün olan en az hata ile geliştirilmiş olur. Bu işlemler sonucu yazılımlarda daha az hatalar bulunur ve daha hızlı bir şekilde geliştirme yapılır.

- **Esneklik**

Geleneksel programlamada, gereksinimler karşılandıktan sonra en iyi sonuç ortaya çıkar. Pratikte ise gereksinimler zaman içinde iş süreçleri değiştiğinden ya da başlangıçta belirlenen gereksinimler öngörülemediği veya tamamlanmadığı için değişmeye devam eder. Sınırsal programlama döngülerin başında kullanıcı hikâyelerini tekrar alır ve döngü sırasında geri bildirimler vasıtasıyla değişen gereksinimlerin yerine getirilmesini sağlar. Bu şekilde geliştirme aşamasında esneklik kazanılmış olur.

- **Maliyet**

Sınırsal Programlama, verimsiz bir şekilde çalışmayı önler ve proje maliyetini azaltmaya çalışır. Bu yöntem, yazılım geliştirme personellerinin dokümanlar üzerinde zaman kaybetmesi ve uzun toplantılar ile vakit kaymesi yerine kod geliştirmeye odaklanmasını ister. Bu sayede test araçlarına yapılacak olan yatırım azaltır. Yazılım üzerinde değişiklik yapma maliyeti yazılım geliştirme süreci ilerledikçe artar. Teslim sonrasında değişiklik maliyeti, tasarım aşamasındaki değişiklik maliyetine göre daha fazladır. Geleneksel programlama yöntemleri, ürün yaşam döngüsünün sonunda müşterinin geribildirimine dayalı değişiklikler yapar; buna karşın sınırsal programlama, geliştirme aşamasında değişikliklere izin verir.

- **Daha Az Risk**

Sınırsal Programlamanın en önemli avantajlarından biri, programlamayla ilgili riskleri azaltmasıdır. Geleneksel programlama, takımdaki kritik üyelere çok bağlıdır. Sınırsal Programlama, görevleri modüllere bölerek riski yayar ve herhangi bir mimar, proje yöneticisi veya bireysel kodlayıcıya bağımlılığı azaltır.

- **Genel Memnuniyet**

Sınırsal Programlama bireylerin gelişim sürecindeki önemini azaltırken aynı zamanda çalışanların memnuniyetini ve kalıcılığını arttırmaya yardımcı olur. Bu yöntem değer odaklı bir yaklaşımdır. Proje kapsamının alt bileşenlere bölünmesi ve sürekli müşteri geribildirimi, uzun bir sürenin öncesinde tamamlanması gereken birçok işin birikmesini önler.

Dezavantajları

Sınırsal Programlamanın en büyük dezavantajı, müşterinin sürekli katılımını üstlenmesidir. Başarı, geliştirme sürecinin birçok aşamasında veri toplamanıza bağlıdır. Birçok müşteri mevcut olmayabilir ve bazı müşteriler sürekli katılımlardan hoşlanmayabilirler. Sınırsal Programlama kodu, tasarım merkezli bir yaklaşım olmaktan çok merkezi bir yaklaşımdır. Uygun dokümantasyonun olmaması, proje üyeleri ayrıldığında ve daha sonra yeni üyeler geldiğinde ürünlerde büyük problem yaratabilir(What are the advantages and disadvantages of Extreme Programming?, 2017).

3.2. Scrum

Scrum, 1990'ların başında Jeff Sutherland ve geliştirme ekibi tarafından tasarlanan çevik bir yazılım geliştirme yöntemidir. Daha sonra scrum yöntemleri Schwaber ve Beedle tarafından daha fazla geliştirilmiştir(Pressman, Maxim, 2015).

Scrum, tekrar eden ve aşamalı bir çevik geliştirme yazılım yöntemidir. Bu yöntem geliştirme sürecinde müşterilerin ihtiyaçlarındaki değişimlere hızlı cevap verir. Bu yaklaşım gözleme dayalı bir yaklaşım olarak tanımlanır. Sorunun tam olarak anlaşılamayacağını veya tanımlanamayacağını kabul eder; scrum takımı ihtiyaç oluştuğunda sorunu hızlı bir şekilde çözmeye odaklanır. Scrum süreci, projeyi en başından mümkün olan en küçük parçalara bölen sade bir işleyiştir. Bu yöntem ile verimli zaman kullanımı maksimum seviyede tutulmaya çalışılır. Sonuç olarak bu yöntemi anlamak kolay ama bu yöntemde ustalaşmak zordur(Ardant, 2008).

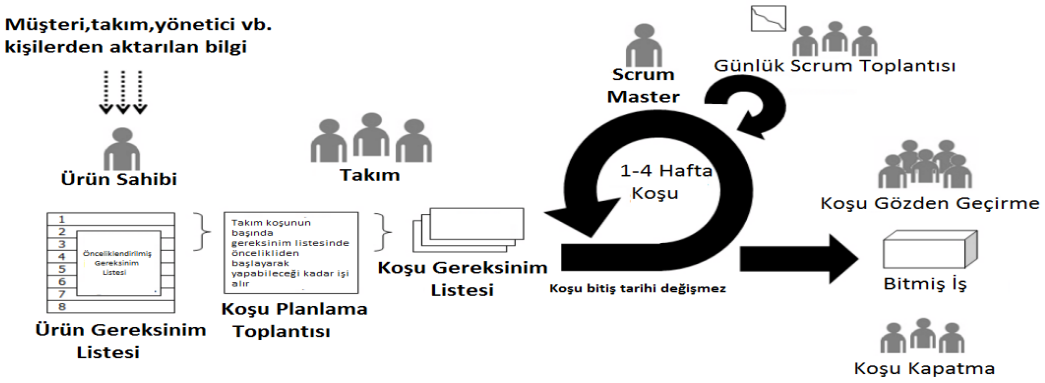
Scrum özelliklerinin listesi:

- Bu yöntem yazılım geliştirme aşamasını yöneten ve kontrol eden çevik bir süreçtir.
- Sahip olunan mühendislik uygulamalarının tamamını kapsar.
- Gereksinimler müşterinin talebiyle hızlı bir şekilde değişirse kademeli olarak sistemi ve yazılımı geliştiren proje ekibi döngüsel bir yaklaşımla değişime ayak uydurur.
- Proje paydaşları ve geliştirme ekibi arasındaki ihtiyaç ve çıkar çatışmasını kontrol eder.
- Proje ekibi arasında iletişimi arttırır ve aralarındaki işbirliğini en üst seviyeye çıkarır.

- Proje ekibi arasındaki uyum ile üretkenlik en üst seviyeye çıkar.
- Bu yöntem, tek bir projeden bütün organizasyonlara ölçeklenebilir. Bu yöntem kullanılarak çok fazla sayıda yazılım geliştiricisi ve uygulayıcısı olan çok sayıda birbiriyle ilişkili proje ve ürün geliştirmiştir.
- Bu yöntem kullanılarak proje ekibi içindeki tüm bireylerin kendi işini en iyi şekilde yapmasını ve bu işi yaparken iyi bir şekilde hissetmesini sağlar. (Uludağ,2006)

Scrum ilkeleri, agile (çevik) manifestosu ile uyumludur. İhtiyaç, analiz, tasarım, geliştirme ve ürün gibi aktiviteleri içeren bir süreç içinde geliştirme faaliyetlerine rehberlik etmek için kullanılır. Her bir aktivite içindeki görevler koşu(sprint) adı verilen bir süreç modelinde meydana gelir. Her bir proje için gerekli olan koşu(sprint) sayısı ürün karmaşıklığına ve boyutuna göre değişir. Bir koşu(sprint) elimizdeki probleme uygulanır ve scrum ekibi tarafından tanımlanır ve süreç içinde sık sık değiştirilebilir.

Şekil 11 Scrum Modeli



(Sutherland, Schwaber, 2011)

3.2.1. Scrum Teorisi

Scrum teorisi, deneysel süreç üzerine kurulmuştur. Riski kontrol etmek ve öngörebilmek için tekrarlanan artırımlı yaklaşım sağlar. Üç temel faktör, her türlü deneysel kontrolde altyapıyı sağlar(Sutherland, Schwaber, 2016).

Bunlar:

- **Şeffaflık:** Yazılım geliştirme sürecinin önemli yönleri tüm paydaşlar tarafından görülebilir olmalıdır. Sürecin tüm katılımcılar tarafından

anlaşılabilmesini sağlanmalıdır. Sürecin hangi aşamada olduğunun anlaşılabilmesi için de ortak bir dil geliştirilmelidir.

- **İnceleme:** Scrum kullanıcıları, projede istenmeyen değişiklikleri saptamak ve amaca ulaşılıp ulaşılmadığını anlamak için scrum sonucu oluşan ürünleri ve projedeki gelişimi sık sık incelemelidir.
- **Uyum:** Bir uzman bir sürece ait bir veya daha fazla özelliğin uygulanmadığını tespit ederse; Süreçteki sapmayı en aza indirgemek için süreci düzenlemelidir.

3.2.2. Scrum Rolleri

Scrum ekibi, ürün sahibi, scrum uzmanı ve geliştirme ekibinden oluşur.

1. Scrum Uzmanı (Scrum Master)

Scrum uzmanı, scrum sürecinin işlemesini etkin bir şekilde sağlamalıdır. Aynı zamanda scrum takımının servis sağlayıcısıdır. Bir yönetici değildir, ancak lider özelliklere sahip olmalıdır. Sorunların üstesinden gelmek zorunda olduğu için iş alanını çok iyi bilmelidir.

Scrum ustaları:

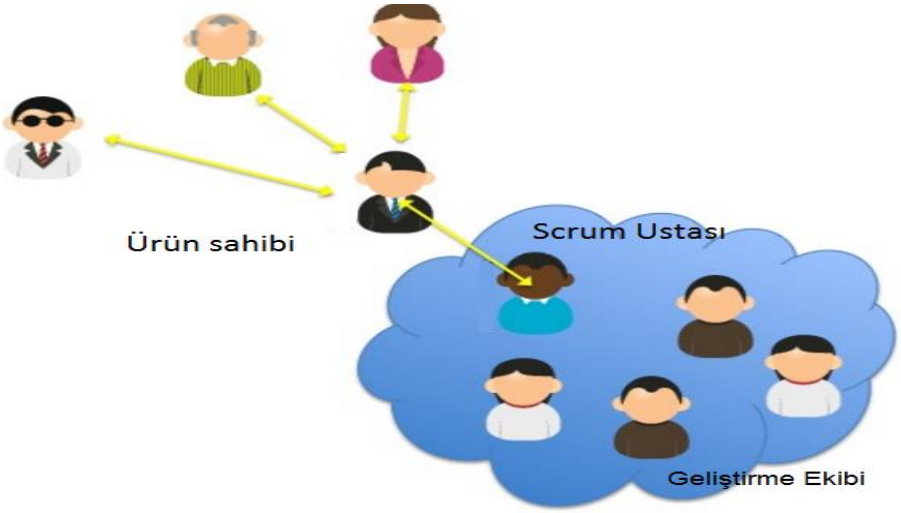
- Ekip işlevselliğini, verimliliğini ve motivasyonunu arttırmaya çalışarak ekip üzerindeki ilişkileri organize eder.
- Yüksek performanslı ekip kurmaya çalışır.
- Ekip işlevselliğini ve ürün işlemlerini etkileyen sorunları ortadan kaldırır.
- Ekibin dikkatinin dağılmasını önler ve her bir üyenin günlük scrum toplantısı, koşu(sprint) planlama ve sprint inceleme toplantılarına katılmasını sağlar.
- İş takibi yapar.
- Scrum sürecinin çalıştığından emin olur.

Mike Cohn, iyi bir scrum ustasının 6 özelliği olması gerektiğini ifade eder. Bu özellikler(Cohn, 2007):

- Sorumluluk sahibi
- Alçak gönüllü
- Takım çalışmasına yatkın
- İş bitirici

- Etkili
- Bilgili olmaktır.

Şekil 12: Scrum Rolleri ve Paydaşları



(Scrum Roles - The Scrum Team, 2017)

2. Ürün Sahibi (Product Owner)

Ürün sahibi, ürünün değerini en yükseğe çıkarmaya çalışır. Bunu yapmak için pazar eğilimleri arar. Ürünün kapsamının yönetiminden sorumlu tek kişidir. Ürün kapsamını belirleyen, öğeleri açıkça tanımlayan, ürün kapsamının şeffaflığını, görünürlüğünü ve netliğini ekip üyeleri için sağlayan kişidir.

Ürün Sahibi:

- Ürünü tanımlar.
- Piyasa değeri dikkate alınarak ürün kapsamındaki kalemleri belirler.
- Ürün teslim tarihi üzerine karar verir.
- İhtiyaç duyulması halinde, her koşunun(sprintin) önceliği ve özelliği üzerinde değişiklik yapar.
- Ürünün sonucundan sorumlu olur.
- Ürün özelliklerini kabul eder veya reddeder.
  (Sutherland vd., 2016)

3. Geliştirme Ekibi (Development Team / Scrum Team)

Geliştirme ekibi üründe çalışan bir yazılım parçacığı(increment) yapmak için çeşitli yeteneklere sahip olmalıdır. Geliştirme ekibi genellikle özel yeteneklere de sahiptir. (Programlama, kalite kontrol, iş analizi, mimari, kullanıcı arayüz tasarımı veya veritabanı tasarımı gibi) Bununla birlikte takım üyelerinin paylaştığı beceriler arasından gereksinimi belirlemek ve kullanılabilir ürün için gerçekleştirme yapmak önemli bir yetenektir. Kodlamayı reddeden insanlar mimarlar veya tasarımcılar vb. bireyler scrum takımları için uygun değildir. Herkes gerektiğinde yeni yetenekler öğrenilmesi veya eskilerin hatırlanması aktivitelerine katılmalıdır. Takımlarda unvan yoktur ve bu kural istisnası olmayan bir kuraldır. Takımlar test, iş analizi gibi işler için alt gruplara da sahip değildir(Sutherland vd., 2016).

Bir geliştirme ekibi, bir koşu(sprint) hedefi gerçekleştirmeyi taahhüt eder. Ekip, amacına ulaşmak için her şeyi yapar. Scrum uzmanı, geliştirme ekibi ile görüşür ve ürün gereksinimini inceler. Geliştirme ekibi, seçilen ürün gereksinimlerini çalışan bir ürüne dönüştürmeyi taahhüt eder. Geliştirme ekibi bu taahhüdü her koşuda(sprint) yapar.

Geliştirme Ekibi:

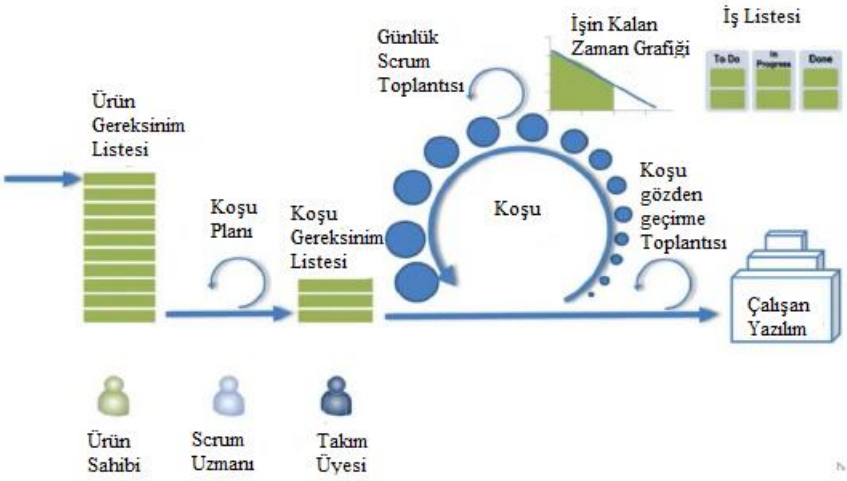
- Gereksinimlerin süre tahminini yapar.
- Küçük takımlardan oluşur. (7+-2)(Hundermark, 2009)
- Çiftli Programlama (Pair programing) yöntemini uygulayabilir.
- Koşuya(sprint) başlarken hedefi seçip, çalışma sürecini belirlerler.
- Koşu hedefine ulaşmak için proje sınırları dâhilinde her şeyi yapmakta serbesttirler.
- Takım üyeleri kendi kendini organize eder.
- Çalışma sonuçlarını ürün sahibine sunar ve geri dönüş alır.

3.2.3. Scrum Olayları(Scrum Events)

Scrum sürecinde denetim ve adaptasyonu sağlamak için 4 Scrum olayı vardır:

- Günlük Scrum Toplantısı
- Koşu (Sprint) Planlama Toplantısı
- Koşu (Sprint) İnceleme Toplantısı
- Koşu (Sprint) Gözden Geçirme Toplantısı

Şekil 13:Scrum Events



(Kır, 2014)

1. Günlük Scrum Toplantısı

Ekip üyeleri tarafından her gün yapılan bir toplantıdır. Her üye bu toplantıya katılmalı ve 3 soruyu yanıtlamalıdır:

- Dün sen ne yaptın?
- Bugün ne yapacaksın?
- İşleminiz için herhangi bir sorun var mı?

Zaman sınırı vardır, 15 dakika ile sınırlıdır(Sutherland vd., 2016).

2. Koşu (Sprint) Planlama Toplantısı

Bu toplantı, gelecek koşunun(sprint) tamamlanması için ürün gereksinim listesindeki işlerden seçilen bir koşu listesi hazırlanır. Her geliştirme üyesi ve ürün sahibi koşu planlama toplantısına katılmalıdır. Geliştirme ekibi, diğer paydaşları da bu toplantıya davet edebilir. Bu toplantı hedefi:

- Koşunun amacını (Sprint Goal)
- Koşu iş listesini(Sprint Backlog) belirlemektir.

Koşunun amacı, geliştirme ekibi tarafından birlikte oluşturulmuş bir bildiridir. Bir veya iki cümleyi içerebilir. Açık olmalıdır. Koşu iş listesi, geliştirme ekibinin tamamlamayı taahhüt ettiği ürün gereksinim listesindeki işlerden ve bu işleri tamamlamak için gerekli görevlerden oluşan bir iş listedir.

Bu işleri yapabilmek için geçirilecek toplantının süresi 2 saat olmalıdır(Kır, 2014).

3. Koşu (Sprint) İnceleme Toplantısı

Her koşunun(sprintin) sonunda geliştirme ekibi potansiyel olarak üründe çalışan bir yazılım parçacığı(increment) gerçekleştirmelidir. Geliştirme ekibi yaptığı tanımı gerçekleştirmek zorundadır. Bu toplantıda ürün içinde çalışan bir yazılım parçacığı (increment) geliştirme ekibi üyeleri tarafından sunulmalıdır. Toplantıda bir önceki koşu(sprint) hakkında bir inceleme olmalıdır. Bu toplantıda en önemli sorusu "Koşu (Sprint) hedefine ulaşıldı mı?" sorusudur.

Koşu(sprint) incelemesinin katılımcıları, Scrum uzmanı, ürün sahibi, geliştirme ekibi, Müşteriler ve Yönetim'dir. Bu toplantının süresi 4 saat olmalıdır(Kır, 2014).

4. Koşu (Sprint) Gözden Geçirme Toplantısı

Bu toplantı ekibi, önceki koşuları (sprinti) tartışır ve bir sonraki koşuyu(sprinti) daha üretken hale getirebilecek nelerin yapılacağını belirler. Bu toplantı için etkili yollardan biri başlama-durma-toplantıya devam etme yaklaşımıdır. Bu yaklaşımı kullanarak, her ekip üyesinden ekibin yapması gereken işleri tanımlaması istenir:

Başlama: Ekibin daha üretken, motive edilmiş ve işlevsel olmaya nasıl başlaması gerektiğini belirtmesi

Durma: Yararlı olmayan bir şeyler yapma ve fazladan problem oluşması

Devam etme: Geliştirme ekibinin verimli, motive ve işlevsel olarak devam etmesi şeklinde tanımlanır(Kır, 2014).

3.2.4. Scrum Eserleri(Scrum Artifacts)

Scrum eserleri 4 tanedir. Bunlar:

- Ürün Gereksinim Listesi (Product Backlog)
- Koşu Gereksinim Listesi (Sprint Backlog)
- Koşunun Kalan Zaman Grafiği (Burn-Down Chart)

- Çalışan Bir Yazılım Parçacığı (Increment)

3.2.4.1. Ürün Gereksinim Listesi (Product Backlog)

Ürün gereksinimi, nihai ürünün parçası olarak gereken özelliklerin sıralı bir listesidir ve üründe yapılacak değişiklikler için tek gereksinim kaynağıdır. Ürün sahibi listedeki sıralamayı değiştirme hakkına sahiptir.

Şekil 14: Ürün Gereksinim Listesi

					... Koşu İtibariyle Kalan Yeni Efor Tahmini					
İşler	Detaylar	Öncelik	Değer Tahmini	Başlangıç Efor Tahmini	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to to add or delete items on my wish list	...	8	4	20						

(Sutherland, Schwaber, 2011)

Ürün gereksinim listesi hiçbir zaman tamamlanmaz. İlk başlangıç aşaması bilinen ve en iyi anlaşılan ihtiyaçlardan oluşur. Kullanılacağı ortam ve ürün geliştikçe ürün gereksinim listesi gelişir. Ürün gereksinim listesi dinamiktir. Ürünün gereksinimlerinin uygun, rekabetçi ve kullanışlı olduğunu belirleyen şey sürekli değişimdir. Ürün var olduğu sürece, ürün gereksinim listesi de var olacaktır. Ürün gereksinim listesinin tüm özellikleri, fonksiyonları, ihtiyaçları, gelişmeleri ve iyileştirmeleri ürünün sonraki sürümlerinde yapılacak değişikliklerden oluşur. Ürün gereksinim listesi tanımlama özelliklerine, siparişlere, tahminlere ve sonuçlara sahiptir. Bir ürün kullanıldıkça ve değer kazandıkça ürünün kullanıldığı alandan geri dönüş sağlar. Bu şekilde ürün gereksinim listesi daha da büyür ve ayrıntılı bir liste haline gelir. Ürün gereksinim listesi asla değişmeyi bırakmaz bu yüzden canlı bir listedir. İş ihtiyaçlarındaki, kullanım alanındaki ya da teknolojideki değişikliler ürün gereksinim listesinde değişime sebep olabilir(Sutherland, Schwaber, 2016).

Aynı ürün üzerinde çoğunlukla birden fazla Scrum takımı çalışabilir.

3.2.4.2. Koşu İş Listesi (Sprint Backlog)

Koşu iş listesi, ürün parçasını teslim etme ve koşu hedefine ulaşma planını ve koşu için seçilen ürün gereksinim listesi kalemlerini içerir. Koşu iş listesi, geliştirme ekibinin ürün parçasında hangi işlevselliğin olacağının ve bu işlevselliği "Bitti" olan bir ürün parçasına dönüştürmek için gerekli olan işin öngörüsüdür(Sutherland, Schwaber, 2016).

Koşu iş listesi, geliştirme ekibinin koşu hedefine ulaşmak için gerekli gördüğü tüm işleri görünür kılar.

Şekil 15: Koşu İş Listesi

				... Gün İtibariyle Kalan Yeni Efor Tahmini					
Ürün Gereksinim Listesi	**Koşu Görevleri**	**Gönüllü**	**Başlangıç Efor Tahmini**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

(Sutherland, Schwaber, 2011)

Koşu iş listesi, Günlük scrum toplantılarında ilerlemenin anlaşılabilmesi için yeterli ayrıntıyı içeren bir plandır. Geliştirme ekibi, koşu boyunca koşu iş listesini değiştirir; geliştirme ekibi koşu içerisinde plana uygun çalıştıkça ve koşu hedefine ulaşmak için gerekli olan işi daha fazla anladıkça koşu listesi belirginlik kazanır(Sutherland, Schwaber, 2016).

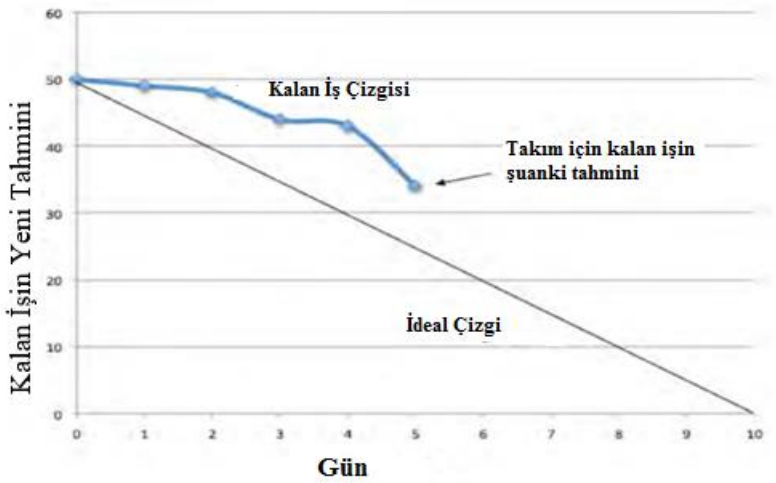
Yeni bir iş gerektikçe, geliştirme ekibi bunu koşu iş listesine ekler. İş yapıldıkça, kalan iş miktarı tahmini güncellenir. Gereksiz görülen her unsur plandan çıkarılır. Sadece geliştirme ekibi, koşu boyunca koşu iş listesini değiştirebilir. Koşu iş listesi, geliştirme ekibinin koşu boyunca başarmayı planladığı işin son derece görünür ve gerçek-zamanlı bir resmidir ve sadece geliştirme ekibine aittir.

Koşu ilerlemesini izlemek koşu iş listesindeki toplam kalan iş koşunun herhangi bir anında hesaplanabilir. Geliştirme ekibi, koşu hedefini gerçekleştirmeye ne derece yakın olduğunu görebilmesi için en azından her Günlük scrum

toplantılarında toplam kalan işi izler. Geliştirme ekibi, koşu boyunca kalan işi izleyerek ilerlemesini yönetebilir.

3.2.4.3. Koşunun Kalan Zaman Grafiği (Burn-Down Chart)

Şekil 16: Koşunun Kalan Zaman Grafiği



(Sutherland, Schwaber, 2011)

- İlgili koşu için tanımlanan görevlerin gözlemlenmesine yarayan görsel bir araçtır.
- Günlük güncellenmelidir.
- Bu sayede koşu sonuna kadar tamamlanması gereken görev miktarı herkes tarafından izlenebilir.
- Her gün sonunda üzerinde çalışılan görevle ilgili güncelleme yapılır ve böylece koşunun kalan zaman grafiği güncel tutulur.
- İstenen isteklerin döngü(iterasyon) süresi içerisinde gerçekleşip gerçekleşemeyeceği bu şema yardımıyla izlenebilmektedir.

3.2.4.4. Çalışan Bir Yazılım Parçacığı (Increment)

İlerleme ürün gereksinim listesindeki işlerden bir koşu boyunca tamamlanmış olanların toplamıdır ve önceki tüm koşuların ilerlemesinin değeridir. Bir koşu sonunda yeni ilerlemenin "tamamlandı" olması gerekir. Bu ilerlemenin kullanılabilir durumda olması scrum takımının "tamamlandı" tanımına karşılık gelir. Ürün sahibinin yayınlanmasına karar verip vermemesine bakılmaksızın ilerleme kullanılabilir durumda olmalıdır.

3.2.5. Scrum Performans Metrikleri

Performansı artırmak için scrum takımlarının performansının ölçülmesi şarttır. Eğer ekip tam performans kapasitesini bilirse o zaman performansı arttırma şansına sahip olabilir. Ancak çevik takımlar için veri toplama zordur. Ekip her zaman koşunun sonunda yapılacak işe odaklanır. Performansı etkileyen metrikler: verimlilik, kalite, kazanılan değer, öngörülebilirlik ve scrum sürecinin etkinliği ile ilgilidir(Performance Metrics for Agile SCRUM Process, 2017).

3.2.6. Scrum Avantajları ve Dezavantajları

Avantajları

- Scrum, şirketin zaman ve paradan tasarruf etmesine yardımcı olur.
- Scrum metodolojisi, iş ihtiyaç listesindeki iş miktarının belirlenmesinin zor olduğu projelerde başarılı geliştirme sağlar.
- Son teknolojilerle hızlı bir şekilde kodlanıp test edilen bu metot sayesinde hızlı hareket etmeden kaynaklanan bir hata kolayca düzeltilebilir.
- Düzenli toplantılar yoluyla işteki ilerlemenin sık sık güncellenmesi sayesinde kolayca kontrol edilebilir. Böylece proje gelişiminin net bir görünürlüğü oluşur.
- Bu yöntemde diğer çevik metodolojileri gibi döngüseldir. Kullanıcıdan sürekli geri bildirim sağlar.
- Kısa koşular ve sürekli geribildirim nedeniyle, değişikliklerle baş etmek daha kolaydır.
- Günlük toplantılar bireysel üretkenliğin ölçülmesini mümkün kılar. Bu şekilde ekip üyelerinin her birinin üretkenliğinde iyileşme olur.
- Konular, günlük toplantılar aracılığıyla önceden iyi tanımlanmış olması nedeniyle hızlı bir şekilde çözülebilir.
- Planlanmış bir zamanda kaliteli bir ürün sunmak daha kolaydır.
- Süreç ve yönetim bakımından genel maliyet minimumdur ve böylece daha hızlı, daha ucuz bir sonuç elde edilir.

Dezavantajları

- Scrum, belirli bir bitiş tarihi belirlenmediği sürece proje paydaşları proje teslim edilene kadar yeni işlevsellikler talep etmeye devam edecektir.

- Bir görev iyi tanımlanmamışsa, proje maliyetlerini ve zamanı tahmin etmek doğru olmayacaktır. Böyle bir durumda, görev birkaç koşu(sprint) üzerinde yayılabilir.
- Ekip üyeleri taahhütte bulunmazsa, proje hiçbir zaman tamamlanmaz ya da başarısız olur.
- Küçük ekipler birlikte iyi çalışırsa hızlı hareket eden projeler için iyidir.
- Bu yöntem deneyimli ekip üyeleriyle daha iyi bir şekilde yürütülür. Ekip yeni başlayanlardan oluşuyorsa, proje zamanında tamamlanamayabilir.
- Scrum ustası yönettiği ekibe güvenirse scrum iyi çalışır. Scrum ustası ekip üyeleri üzerinde çok katı denetim uygularsa, bu durum ekip üyeleri için son derece sinir bozucu olur ve moral bozukluğuna yol açar. Bu şekilde proje başarısızlıkla sonuçlanabilir.
- Ekip üyelerinden herhangi biri gelişme sırasında ayrılırsa, proje geliştirme üzerinde büyük bir ters etki yapabilir.
- Test ekibi her koşudan(spritten) sonra test yapamazsa proje kalite yönetimi uygulamak ve ölçmek zor olur(The Advantages and Disadvantages of Agile SCRUM Software Development, 2017).

3.3. Kristal Yöntemler

Kristal yöntem, Alistair Cockburn tarafından geliştirilmiştir. Kristal yaklaşımında, proje süreçlerine düşük öncelik verilmektedir. Bu yöntem süreçler yerine ekip iletişimi, takım üyesi becerileri, insanlar ve etkileşim üzerine odaklanmaktadır.

Cockburn'ın felsefesi, her takım farklı yetenek ve becerilere sahip olduğunu bilir ve bu nedenle her takım kendine özgü bir süreç kullanır. Sürecin asgariye indirilmelidir ama sürecin asgariye indirilmesi önemli değildir(Software Development Methodologies, 2017).

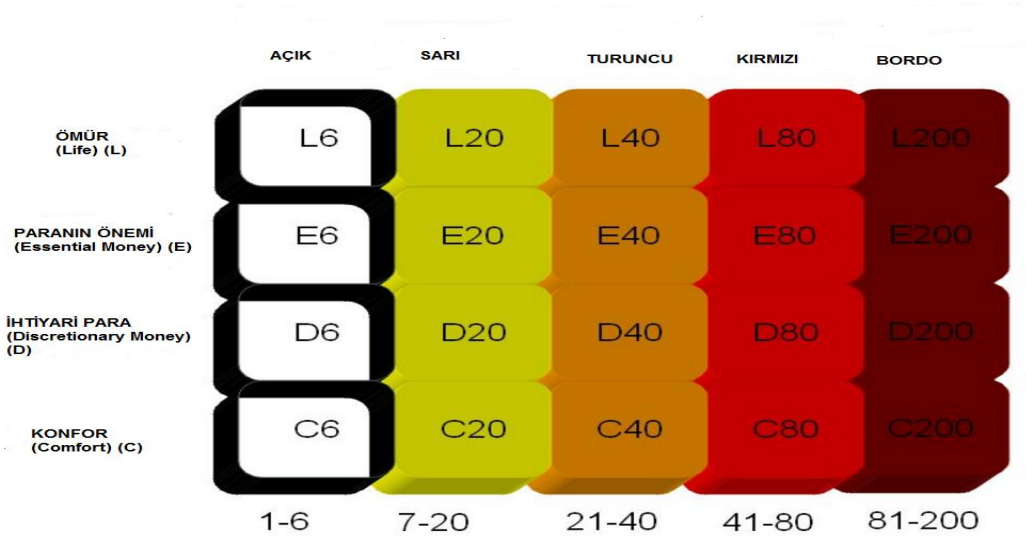
Cockburn çeşitli problemleri çözmek için farklı stratejilere ihtiyaç duyan farklı büyüklükte takımlar için uygun metotlar geliştirmiştir.

Kristal ailesi 8 farklı metodolojiden oluşur. Proje büyüklükleri ve proje kritikliğine uygun metot seçilir. Bu metotlar:

- Açık Kristal
- Sarı Kristal

- Turuncu Kristal
- Turuncu Kristal Web
- Kırmızı Kristal
- Bordo Kristal
- Elmas Kristal
- Safir Kristal (Crystal Methods, 2017)

Şekil 17: Kristal Ailesi



(The Crystal Family of Methodologie, 2017)

3.3.1. Kristal Metot Özellikleri

Kristal ailesindeki metotlar arasında 7 ortak özellik vardır. Bir projenin başarılı olması bu özelliklere bağlıdır. Bu özellikler:

- Sık sık Teslimat (Frequent delivery)
- Yansıtıcı Gelişim (Reflective improvement)
- Yakın İletişim (Close communication)
- Kişisel Güvenlik (Personal safety)
- Odaklanma (Focus)
- Uzmanlara Kolay Erişim (Easy access to expert users)
- Otomatik testler, yapılandırma yönetimi ve sık entegrasyon ile teknik ortam (Technical environment with automated tests, configuration management, and frequent integration)

  (Crystal Methods, 2017)

1. Sık Sık Teslimat

Yazılım için geliştirilen program parçacıklarının düzenli olarak yayınlanmasıdır. Kristal yöntemlerde yazılıma eklenen yeni özelliklerin sürümlerde yayınlanması projenin boyutuna bağlı olarak haftalık, aylık ya da 3 ayda bir şeklinde olabilir. Kristal yöntemlerde bir sürümün içinde birde fazla çalışan program parçacığı olabilir. Her program parçacığını yayınlamak uygun olmayabilir bunlar bir araya toplanıp tek bir sürümde yayınlanabilir(Crystal Methods, 2017).

2. Yansıtıcı Gelişim

Yazılım geliştiricilerin proje geliştirme sürecinde iyileştirme çözümleri üretmesidir. Belli aralıklarla proje hakkında düzenli toplantılar yapılarak fikir alış verişi yapılır. Süreçler iyileştirilmeye çalışılır.

3. Yakın İletişim

Geliştirme ekibinin bir odada olup programla ilgili soruları yanındaki diğer programcıya sorarak çözüme ulaştırması anlamına gelir. Aynı oda içinde diğer programcılarda bu tür sorunlar ve çözümler üzerinde bilgi ve deneyim kazanırlar.

4. Kişisel Güvenlik

Bu bir ekip içindeki kişilerin görüşlerini özgürce söylemesidir. Bir kişi soru sorduğunda diğer kişiler gülümserse ya da sorusunu küçümserse o kişinin daha sonra soru sorma ihtimali azalır. Ekipteki insanlar birbirlerine güvenmeli ve ortaya çıkabilecek sorunlar için konuşmaktan çekinmemelidirler(Crystal Methods, 2017).

5. Odaklanma

Projenin yönünün belirlenmesi ve kişilerin proje içindeki işlerine odaklanmasıdır. Bu kişilerin uzun süreli toplantılar, uzun sorular, telefon görüşmeleri ile çalışmaları bölünmemelidir. Benzer şekilde yazılımcılar çalışırken kesintisiz belli bir süre çalışmalıdırlar ve o süre içinde onlarla iletişim kurulmamalıdır.

6. Uzmanlara Kolay Erişim

Bir projenin geliştirme aşamasında işi iyi bilen bir kişiyi kapsar. Geliştirme ekibinin sorularını cevaplar. Belli periyotlarla yazılımcılar ile telefon ile ya da yüz yüze görüşmesi proje geliştirme sürecini verimli hale getirir.

7. Otomatik Testler ve Yapılandırma Yönetimi ve Sık Entegrasyon ile Teknik Ortam

Bu ortam sürekli entegrasyon yapılan ve testlerin yapıldığı ortamdır. Böylece yeni güncellemelerde hatalar tespit edilip çözülür.

3.4. Dinamik Sistem Geliştirme Yöntemi (Dynamic Systems Development Method - DSDM)

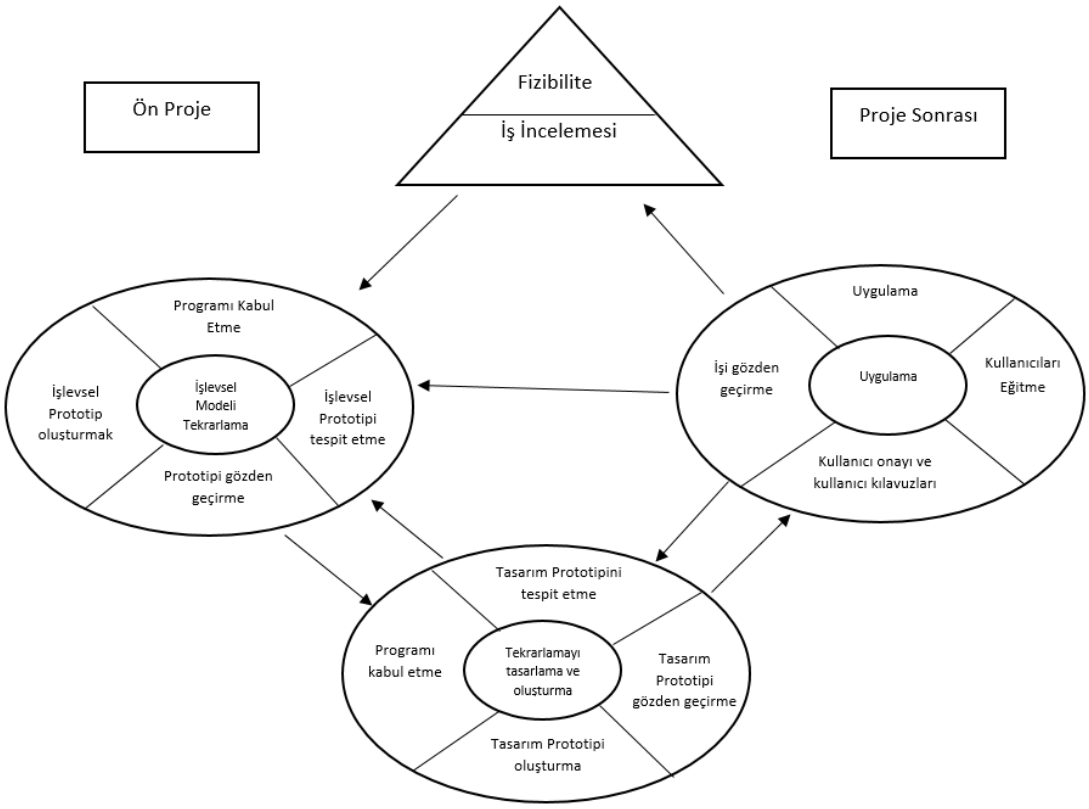
Dinamik yazılım geliştirme yöntemi, proje yönetimi ile ilgili mevcut bilgileri somutlaştıran bir altyapıdır. Bu yaklaşım bir üründeki işlevselliğin derecisine dikkat etmek ve ardından zamanı ve kaynakları hedefe ulaşmak için düzenlemek yerine ilk önce zaman ve kaynağı ilgilenir sonra da işlevsellikle ilgilenir (Johnson, Higgins, 2008).

Dinamik Sistem Geliştirme Özellikleri:

- Geliştirme sonuçları doğrudan ve hemen görülebilir.
- Kullanıcılar sistemin geliştirilmesine aktif olarak katıldıkları için sistemi benimserler ve sorunların üstesinden gelme olasılıkları daha yüksek olur.
- Temel işlevler hızlı bir şekilde verilir ve düzenli aralıklarla daha fazla işlevsellik elde edilir.
- Bürokrasiyi ve ilgili taraflar arasındaki iletişim engelini ortadan kaldırır.
- Kullanıcılardan gelen sürekli geribildirim nedeniyle, geliştirilen sistemin gereksinimi karşılama olasılığı daha yüksek olur.
- Proje geliştirme aşamasının yarısında karşılaşabilecek kötü sonuçların aksine projenin işe yarayıp yaramayacağına ilişkin erken göstergeler vardır.
- Sistem zamanında ve bütçeye uygun teslim edilir.
- Kullanıcıların yetenekleri projenin gelişimini etkiler.

(The Dynamic Systems Development Method (DSDM)-Agile Methodology, 2017)

Şekil 18:Şekil DSDM Süreci



(Johnson vd., 2008)

3.4.1. DSDM Prensipleri

Dinamik sistem geliştirme yönteminin 9 tane prensibi vardır. Bu prensipler:

1. Aktif Kullanıcının Katılımı Zorunluluğu

Kullanıcının aktif bir şekilde projeye katılması hataları etkin bir şekilde azaltır ve projenin hata maliyetleri düşer. Dinamik sistem geliştirme yöntemi küçük belirli bir kullanıcı grubuyla sürekli çalışmayı önerir.

2. Takımların Karar Vermesinin Güçlendirilmesi

Projedeki katılımcılar ve yöneticiler arasındaki iletişimden kaynaklanan sürtüşmeler süreçlerin yavaşlamasına sebep olabilir. Bu sebeple aşağıdaki belirtilen alanlarla ilgili proje takımlarına sınırlı yetki verilmelidir.

- Uygulamadaki gereksinimler için,
- Hangi fonksiyonda belirli bir artış olması gerektiği için,
- İhtiyaçların ve özelliklerin önceliklerinin belirlenmesi için,

- Detaylı teknik çözümün için yetki verilmelidir.(The Dynamic Systems Development Method (DSDM)-Agile Methodology, 2017)

3. Sık sık Teslimat Yapılmalı

Projelerdeki program parçacıklarının belli periyotlarla güncellenmesi o program parçacığındaki hataların test ekipleri ya da son kullanıcılar tarafından kolaylıkla belirlenebilir. Ve bu şekilde hızlıca hatalar düzeltilebilir.

4. İyi Bir İş Kabul Edilen Teslimat Kriteri

Dinamik sistem geliştirme yöntemi için öncelik işin gereksiniminin yerine getirilmesidir. Daha sonraki aşama bu sürecin iyileştirilmesidir.

5. Tekrarlı ve Artırımlı Gelişimin Zorunlu Olması

Proje karmaşıklığının yönetilebilmesi için proje küçük küçük parçalara ayrılır. Her sürüm yeni özellikler eklenerek çıkarılır.

6. Gelişme Sırasındaki Tüm Değişikliklerin Tersinir Olması

Proje süresince gereksinimlerdeki değişikliklere tepki verebilmesidir. Bu şekilde esnek bir şekilde geliştirme süreci devam edebilir.

7. Üst Seviye Gereksinimlerin Belirlenmesi

Proje süresi boyunca gereksinimlerin değiştirilmesi göz önüne alınarak üst seviye gereksinimler belirlenmelidir. Bu gereksinimler ölçüsünde değişikliğe izin verilmelidir.

8. Proje Boyunca Sürekli Test Etme

Dinamik sistem geliştirme yöntemi sürecin başından test etmeyi gerektirir. Hatta bir kontrol grubu ya da çapraz sorgulama yaparak sürecin kontrol edilmesini ister.(The Dynamic Systems Development Method (DSDM)-Agile Methodology, 2017)

9. İşbirliği ve Ortak Çalışma Yaklaşımı

Bir projede çalışan teknik personel ve iş personelinin ortak çalışması önemlidir. Her personel birbirine güvenerek çalışmalı ve geri bildirimler düzgün olmalıdır.

3.4.2. DSDM'nin Yedi Aşaması

Dinamik sistem geliştirme yöntemi 7 aşamadan oluşur. Bu aşamalar:

1. **Ön Proje (Pre Project)**

Proje önerisi ve önerilen projenin seçilmesi aşamasıdır. Bu aşamada projenin gerçekleştirilip gerçekleştirilmeyeceği belirlenir.

2. **Fizibilite Çalışması (Feasibility Study)**

Problemin tanımı, yaklaşık maliyetin hesaplanması ve problemin çözümlenmesine dair teknik hususlar ele alınır.

3. **İş Katmanı (Business Study)**

İhtiyaçların belirlenmesi ve iş planının yapılması sürecidir.

4. **Fonksiyonel Model Döngüsü (Functional Model Iteration -FMI)**

İş katmanında belirlenen ihtiyaçlar ve iş planı çerçevesinde belirlenen işlerin bilgisayar sistemleri üzerinde modellenmesi sürecidir.

5. **Dizayn ve Döngüyü Oluşturma (Design and Build Iteration- DBI)**

Tasarım ve kodlamanın oluşturulduğu aşamadır.

6. **Uygulama (Implementation)**

Kodlamadan sonraki aşamadır. Ürünün son kullanıcıların kullanacağı aşamaya gelmesidir.

7. **Proje Sonrası (Post Project)**

Proje bittikten sonraki sistemin nasıl performans gösterdiği ve iyileştirme yapılıp yapılmayacağına karar verilen aşamadır.

3.4.3. DSDM Rolleri

Dinamik sistem geliştirme yöntemi için farklı roller vardır. Bu roller: (What Is DSDM?, 2017)

- **Temsilci (Ambassador)**: Müşteriler ya da kullanıcılar ve geliştirme ekibi arasında bir geçiş yapan kişidir. Geliştirme ekibini koordine eder ve sistemin nasıl çalışacağı konusunda genel bir anlayışa sahip olmalıdır.
- **Vizyoner (Visionary):** Projenin arkasındaki itici güçtür. Projeyi, iş hedeflerine doğru yön bulmaya devam ettirir. Genellikle projeyi başlatan ya da düşünen kişidir.

- **Danışmanlar (Advisers):** İşin alanlarında otomatikleştirilmesi gereken veya bu alanları otomatikleştirmek için gerekli teknolojiler konusunda pratik bilgiye sahip insanlardır.
- **Teknik Koordinatör(Technical Coordinator):** Teknik koordinatör, sistemin çeşitli teknik yönlerini koordine eder ve bunların düzgün ve doğru biçimde etkileşimini sağlar.
- **Yönetici Sponsor (Executive Sponsor):** Projeye fon ve kaynak katkısı olan sistemin güçlü bir savunucusudur.
- **Proje Yöneticisi (Project Manager):** Proje gelişimini ve prototip sonuçlarını denetler.
- **Ekip Yöneticisi(Team Manager):** Ekibin yönünü belirler ve ekibin sorunsuz bir şekilde çalışmasını sağlar.
- **Kıdemli Geliştirici (Senior Developer):** İş planlarını ve gereksinimleri koda çevirebilen bilgi ve uzmanlığa sahip olan üst düzey bir geliştiricidir.
- **Kolaylaştırıcı (Facilitator):** Ekibin çalışmalarının pratikleştirir.
- **Katip (Scribe):** Ekip tarafından alınan kararları, tartışmaları ve planları belgelemekten sorumlu olan kişidir.
- **Testçi (Solution Tester):** Projeye testler uygulayarak, projenin teknik açıdan doğru çalışıp çalışmadığını kontrol eder. Test sonuçlarını belgeler.

3.4.4. DSDM'de Uygulanan Temel Teknikler

- **MoSCoW Önceliklendirmesi**

Var olmalı - kesinlikle gerekli olan ve sistemin temelini oluşturan şeylerdir.

Olmalıdır - iş çözümü için önemli olan işlerdir.

Olabilir - yararlı şeylerdir, ancak kısa bir süre olmasına gerek yoktur.

Bu Zamanda Olmayacak - Daha sonra bekleyecek işlerdir.

Her şeyi yapmak için zaman yoktur. Öncelik sırası yapılıp ona göre işler yapılmalıdır.

- **Prototipleme**

Prototipler DSDM'de yoğun olarak kullanılmaktadır. İlgili tüm tarafların sistemin çeşitli yönlerini net bir şekilde görmelerini sağlarlar.(What Is DSDM?, 2017)

DSDM, prototip türleri:

İş Prototipi: Gelişen sistemin değerlendirmesine izin verir.

Kullanılabilirlik Prototipi: Kullanıcı ara yüzünü kontrol eder.

Performans Prototipi: Çözümün performansı sağlayacağından veya hacmi ele alacağından emin olur.

Yetenek Prototipi: Olası seçenekleri değerlendirir.

(The Dynamic Systems Development Method (DSDM)-Agile Methodology, 2017)

- **Hafif Toplantılar**

Fikirlerin oluşturulması için ideal ortam sağlar.

Kararlar daha geniş bir paydaş grubu tarafından alınabilir.

Kararlar hızlı ve doğru bir şekilde verilir.

- **Zaman Çizelgesi**

Sonunda bir ürün teslimi yapılmalıdır.

Görevlerin ne zaman tamamlanacağı bilgisi içerir.

İş ihtiyaçlarına göre görevler değiştikçe güncellenebilir.

3.4.5. DSDM Avantajları ve Dezavantajları

Avantajları

- Projenin ömrü boyunca etkin kullanıcı katılımlı olması ve gelişimin yinelemeli doğası sayesinde, ürünün kalitesini geliştirir.
- DSDM hızlı teslimat sağlar.
- Proje maliyetlerini azaltır.

Dezavantajları

- Lisanslama masrafı fazla olabilir.
- Proje başlangıcında göreceli bilinmezlikler fazla olabilmesidir.
- Organizasyonda kültürel değişimlere sebep olabilir.

(Dynamic Systems Development Method (DSDM)(a)(b), 2017)

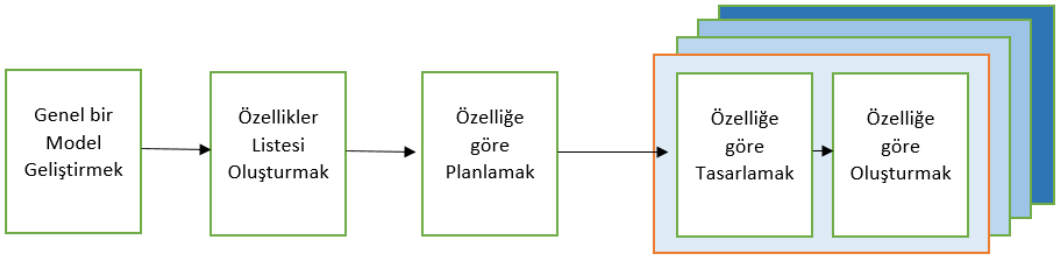
3.5. Özellik Odaklı Geliştirme Yöntemi (Feature Driven Development -FDD)

Özellik odaklı geliştirme (FDD) yöntemi ilk olarak 1999'da UML ile Java Modelleme adlı kitap aracılığıyla dünyaya tanıtıldı. Bu kitap, Jeff DeLuca'nın şirketi tarafından takip edilen ve Peter Coad'ın özelliklerini içeren yazılım sürecinden bahsetmektedir. FDD 1997'de büyük bir Singapur bankası için 15 ay süren 50 kişilik bir projeye uygulandı ve ardından 18 ay süren 250 kişilik projeye uygulandı.(Feature Driven Development (FDD) and Agile Modeling, 2017)

Özellik odaklı geliştirme yöntemi tekrar eden beş ana faaliyet vardır. Bunlar:

- Genel bir model geliştirmek.
- Özellik listesi oluşturmak.
- Özelliğe göre planlamak.
- Özelliğe göre tasarlamak.
- Özelliğe göre oluşturmak.

Şekil 19: Özellik Odaklı Geliştirme Yöntemi



(Johnson, Higgins, 2008)

Özellik odaklı geliştirme yöntemi, yazılım geliştirme yaşam döngüsünde sadece tasarım ve geliştirme aşamalarını etkileyen çevik bir yazılım geliştirme yöntemidir. Beş adet sıralı ana faaliyeti vardır: tüm sistemi modelleme, özellik listesi hazırlama, özellikleri planlama, özelliğe göre tasarım ve özelliğe göre geliştirmedir. Sistemin modelleme aşamasında, takım üyeleri ve uzmanlar sistem için bir yol planı hazırlar. İkinci aşamada gerekli özellikler tüm tarafların anlayabileceği basitlikte parçalara bölünür. Üçüncü aşamada "tasarım paketleri" adı verilen bu parçaların öncelikleri belirlenir ve her bir parça, bir kaç programcıdan sorumlu bir şef programcıya atanır. Özelliğe göre tasarım aşamasında sürecin döngüsel kısmı başlar. Şef programcı kendisine atanmış tasarım paketinden 1-2 hafta içerisinde yapılabilecek

bir alt iş seçer. Geliştirme aşamasında bu alt iş yeniden ayrıntılı olarak analiz edilir tasarlanır, kodlanır, test edilir ve entegre edilir. (Palmer, Felsing, 2001)

FDD yönteminde, kod yazıldıktan sonra birim testi ve kod inceleme alt süreçlerine geçilir. Hangisinin önce yapılacağına şef programcı karar verir. Birim testinin nasıl yapılacağına dair sınırlama yoktur. Manuel veya otomasyon aracılığı ile yapılabilir. Kod inceleme FDD için zorunlu bir uygulamadır ve bunun için kod yazma süresinin yaklaşık dörtte biri kadar süre ayrılır. Kod inceleme, iyi bir şekilde yapıldığında test etmekten daha fazla ve çeşitli yazılım kusurlarını bulabilmektedir.(Mcconnell, 2004)

FDD yöntemi için Anderson'a göre 6 önemli özellik vardır. Bunlar:

1. Projenin Gözden Geçirilmesi(Domain Walkthrough) – Gereksinimler yüz yüze görüşülerek geliştiricilere açıklanır.
2. Tasarım (Design) - Sıra diyagramının oluşturulur.
3. Tasarım Denetimi (Design Inspection) - Tasarımın gereksinimleri karşıladığından emin olmak için eş değerlendirmesi yapılır.
4. Kodlama (Code) - Metotlar, tasarımı sunmak için sınıf dosyalarında yazılmıştır.
5. Kod Denetimi ve Birim Testi (Code Inspection and Unit Test) - Kodun tasarımda belirtilenleri yapıp yapmadığını kontrol etmek için test ve eş değerlendirmesi yapılır.
6. Derleme (Promote To Build) - Ürün testi yapılarak sisteme entegre edilir. (Anderson, 2004)

Tablo 2: Proje özellikleri ve tamamlanma yüzdesi

Proje Özelliği	**Tamamlanma Yüzdesi**
Projenin Gözden Geçirilmesi	1%
Tasarım	40%
Tasarım Denetimi	3%
Kodlama	45%

Kod Denetimi ve Birim Testi	10%
Derleme	1%

(Anderson, 2004)

3.5.1. Özellik Odaklı Geliştirme Pratikleri

Özellik geliştirme pratikleri sekiz (8) tanedir. Bu pratikler:

1. Etki Alanı Nesne Modellesi(Domain Object Modeling):

Bir etki alanındaki önemli nesne türlerini ve aralarındaki ilişkiye tasvir eden diyagramlardan oluşur. Bu şekilde proje belirli parçalara ayrılıp nesnelerle modellenir. Her modeldeki nesne veya sınıflar daha küçük bir problemi çözmek için tanımlanır. Bu şekilde sınıflar tanımlanarak büyük resim netleşmeye başlar.

2. Özellikle Geliştirme (Developing by Feature):

Her problem daha küçük fonksiyonlara ayrılır bu küçük fonksiyonlara özellik adı verilir. Bu şekilde problemleri küçük fonksiyonlara bölmek sistemin geliştirilmesini ve değiştirilmesini kolaylaştırır.

Özellik tanım olarak iki hafta içinde uygulanabilen müşteri tarafından değer verilen küçük işlevdir.

Özelliklere örnekler şunlardır:

- Toplam satış miktarını hesaplanması
- Bir satış elemanının performansını değerlendirilmesi
- Bir kullanıcının şifresini doğrulanması (Palmer vd., 2001)

Bu işlemlerin gerçekleştirilmesi için;

Toplam satış miktarının hesaplanması için Satış sınıfının içinde Toplam Satışı Hesapla() fonksiyonunun tanımlanması,

Bir satış elemanının performans değerlendirilmesi için satıcı sınıfının içinde Performans Değerlendir() fonksiyonunun tanımlanması,

Bir kullanıcının şifresinin doğrulanması için Kullanıcı sınıfının içinde Şifre Doğrula fonksiyonunun tanımlanması gerekir.

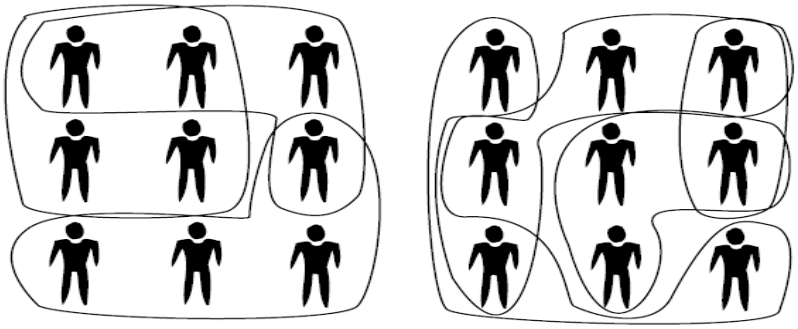
3. Sınıf Sahipliği (Class (Code) Ownership) :

Sınıf sahipliği, farklı parçaların veya kod grubunun tek bir kişiye atandığı anlamına gelir. Sınıf, tutarlılık, performans ve kavramsal bütünlükten sorumludur.(Feature-driven development(a), 2017)

4. Özellik Takımları (Feature Teams):

Bir özelliğin uygulanması için birden fazla sınıfa ihtiyaç duyulabilir. Bu şekilde ihtiyaç duyulduğu durumda sınıf sahiplerinden dinamik küçük ekipler oluşturulur. Aynı zamanda bu takımlar sayesinde tasarımlar birden fazla kişi tarafından değerlendirilerek seçilir.

Şekil 20: Özellik Takımları



(Palmer vd., 2001)

Şekildeki gibi ihtiyaç duyulduğu zaman sınıf sahiplerinden özellik takımları oluşturulur.

5. Denetimler (Inspections):

Denetimler, öncelikli olarak kod içindeki hataların tespit edilmesi, kaliteli tasarım ve kod geliştirmek için yapılır.

Aetna Sigorta Şirketi bir programdaki hataların% 82'sini denetim yaparak bulmuştur ve geliştirme kaynaklı hatalarını % 25 oranında azaltmıştır. (Palmer vd., 2001)

6. Düzenli Derleme Takvimi (Regular Build Schedule):

Düzenli aralıklarla tamamlanan özelliklerin tüm kaynak kodu alınır ve bağlı bulunduğu kütüphaneler ve bileşenler ve komple sistem derlenir. Bu, her zaman çalışan bir sistemin mevcut olmasını sağlar. (Goyal, 2007)

7. **Konfigrasyon Yönetimi (Configuration Management) :**

Konfigrasyon yönetimi yaparak çalışan sistemin tüm sürümlerinin takibi yapılabilir. Kod içinde yapılan değişikler ve bu değişiklerin hangi ekipler tarafından yapıldığı görülebilir.

8. **İlerlemenin ve Sonuçların Görülebilirliği (Visibility of progress and results):**

Proje boyunca yapılan işlerin takip edilmesi raporlanması ve yöneticilere bilgilendirme yapılarak projenin yönlendirilmesine yardımcı olmayı sağlar. Proje kapsamında yapılan işlerin detaylarına sahip olunabilir.

3.5.2. Özellik Odaklı Geliştirme Rolleri

Özellik odaklı geliştirme rolleri proje büyüklüklerine ve insan kaynağına göre farklılık gösterebilir. Başlıca özellik odaklı geliştirme rolleri şunlardır:

1. **Proje Yöneticisi (Project Manager):** Proje yöneticisi; projenin yönetilmesi, personel temini, proje bütçesi ve teknik ekipmanların sağlanmasında sorumlu kişidir.
2. **Tasarım Yöneticisi (Chief Architect):** Sistemin genel tasarımlarından sorumlu kişidir.
3. **Yazılım Yöneticisi (Development Manager):** proje kapsamında günlük yazılım geliştirme işlerinden sorumlu kişidir. Yazılım uzman personellerinin kendi aralarında programlama üzerine yaşadıkları sorunları çözmekten sorumludur.
4. **Yazılım Uzmanı (Chief Programmers):** Yazılım yaşam döngüsü boyunca yazılımların geliştirilmesinden sorumlu uzman personellerdir. Analiz ve tasarım sürçlerine katılırlar. Yazılımların özelliklerini geliştirilmesi için küçük geliştirici ekiplerden sorumludur.(Goyal, 2007)
5. **Sınıf Sahipleri (Class Owners):** Sınıf sahipleri, yazılım uzmanı rehberliğinde küçük yazılım ekibi personeli olarak çalışan ve gerekli özellikleri tasarlamak, kodlamak ve test eden geliştiricilerdir.(Goyal, 2007)
6. **Alan Uzmanı (Domain Experts):** Alan uzmanları, bir projenin geliştirilmesi sürecinde proje kapsamındaki iş süreçlerini bilen kişilerdir. Bu kişileri analist olarak adlandırılabiliriz. Sistemin doğru bir şekilde geliştirilmesinden sorumludurlar. İşi yapan birimler ile geliştirme ekibi arasında köprü görevi görürler.

7. **Sürüm Yöneticisi (Release Manager):** Sürüm yöneticileri, yazılım uzmanlarının yaptığı geliştirmeleri takip eder. Proje yöneticisine rapor verir.
8. **Dil Rehberi (Language Guru):** Dil rehberi, bir programlama dilini veya teknolojiyi iyi şekilde bilen kişidir. Yeni bir programlama dili veya teknoloji kullanıldığı durumlarda bu rol kritiktir.
9. **Derleme Mühendisi (Build Engineer):**Derleme mühendisi, düzenli olarak derlemelerin yapılmasından sorumludur.
10. **Araç-Gereç Sorumlusu (Toolsmith):** Araç-gereç sorumlusu, geliştirme ekibi, test ekibi ve veri dönüştürme ekipleri için küçük araçlar oluşturmaktan sorumludurlar.(Goyal, 2007)
11. **Sistem Yöneticisi (System Administrator):** Sistem yöneticisi, proje ekibi için herhangi bir sunucu temini, sunucu yapılandırılması ve sistemle ilgili taleplerin gerçekleştirmesini sağlayan kişidir.
12. **Test Sorumluları (Testers):** Test sorumluları, sistemin işlevlerinin kullanıcı gereksinimlerini karşılayıp karşılamadığını ve sistemin düzgün bir şekilde çalıştığını kontrol eden kişilerdir.
13. **Dağıtımcılar (Deployers):** Dağıtımcılar, mevcut verilerin yeni sistem tarafından gerekli olan formatlara dönüştürür ve sistemin sürümlerinin yayınlanması için çalışırlar.
14. **Teknik Yazarlar (Technical Writers):** Teknik yazarlar, çevrimiçi ya da fiziksel olarak kullanıcı dokümanlarını hazırlarlar.

3.5.3. Özellik Odaklı Geliştirme Avantajları ve Dezavantajları

Avantajları

- Büyük projeler için daha uygundur.
- Tasarımlar küçük parçalara ayrıldığı için riskler azalır.
- Proje özellikleri belirlendiği için proje maliyeti daha doğru bir şekilde belirlenebilir.

Dezavantajları

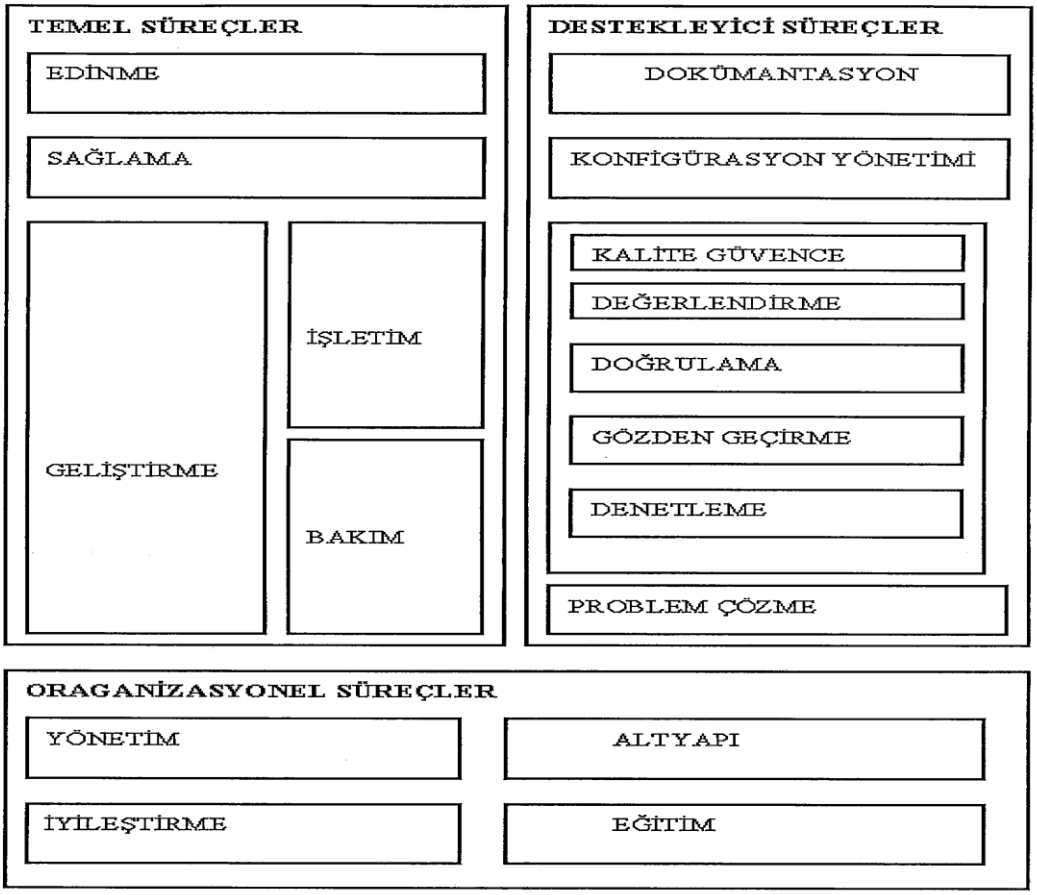
- Küçük projeler için etkili bir yöntem değildir.
- Yazılım uzmanına bağımlılık ve güven fazladır.
- Dokümantasyon çok fazla değildir.

(Feature Driven Development(b), 2017)

4. ISO/IEC 12207 YAZILIM YAŞAM DÖNGÜSÜ STANDARDI

ISO/IEC 12207, yazılım geliştirme sürecindeki kullanıcı, tedarikçi, geliştirici, yüklenici, alt yüklenici ve teslimattan sonraki destek ile bunlar arasındaki bağlantıların tanımlandığı sistem mühendisliği ilkesine dayanmaktadır. Bu standart herhangi bir yazılım yaşam döngüsü modeli ile birlikte kullanılabilmektedir.

Şekil 21: ISO/IEC 12207 Standardı



(Altun, 2004)

Bu standardı temel süreçler, destekleyici süreçler ve organizasyonel süreçler olarak üç bölüme ayırabiliriz. Bu süreçler birbiriyle ardışık olarak değil paralel olarak beraber kullanılırlar.

4.1. Temel Süreçler

Temel yazılım yaşam döngüsünün esas sağlayıcılarıdır. ISO/IEC 12207 standardı bir yazılım projesinin başlangıcından bitişine kadar yaşam döngüsü boyunca

icra edilecek; yazılım edinme, sağlama, geliştirme, işletme ve bakım süreci faaliyetlerini tanımlamaktadır. (Gümrükçü, 2010)

4.1.1. Edinme Süreci

Edinim süreci, bir sözleşme yoluyla yazılım veya sistem satın alınması sürecindeki eylemlerin tanımlamaktadır. Satın alma sürecinde ihtiyaçların tanımlamasıyla başlar, tedarikçi için teklif talebi ve satın alınma sürecini kapsar.

4.1.2. Sağlama Süreci

Yazılım sistemini sağlayan tedarikçinin, teklife yanıt hazırlanması, sözleşme, planlama, yürütme ve denetim, gözden geçirme ve değerlendirme, teslim ve tamamlama faaliyetlerini içermektedir. (Gümrükçü, 2010)

4.1.3. Geliştirme Süreci

Geliştirilecek olan sistem için ihtiyaç analizi, tasarımı, kodlaması, test edilmesi, yazılımın yüklemesi ve kabulünün yapılması sürecidir. Bu süreçte geliştirme için bir yazılım yaşam döngüsü modeli seçilerek bu modele uygun geliştirme yapılabilir.

4.1.4. İşletim Süreci

İşletim süreci, sistemin teslim edilmesinden sonra sistemin işletilmesi sırasındaki kullanıcı faaliyetlerini kapsar.

4.1.5. Bakım Süreci

Bakım süreci, sistemin kullanılması esnasında ihtiyacımız olabilecek işlevler, sistemle ilgili iyileştirmeler ve arızaların giderilmesini kapsar.

4.2. Destekleyici Süreçler

Projenin başarılı olması ve kalitesinin arttırılması için yazılım yaşam döngüsü boyunca diğer süreçlere destek sağlayan faaliyetler yerine getirilmektedir.

Bu süreçler, dokümantasyon, konfigürasyon yönetimi, kalite güvence, değerlendirme, doğrulama, gözden geçirme, denetleme ve problem çözme süreci olarak sekiz tanedir.

4.2.1. Dokümantasyon Süreci

Yazılım gerçekleştirme aşamalarında üretilen bilgilerin sonradan kullanılabilmesi için düzgün bir formatta kayıt altına alınması gerekmektedir.

4.2.2. Konfigürasyon Süreci

Yazılım adımlarının belirlenmesini sağlayan bir süreçtir. Sürümlerin, denetim altında tutulmasını sağlayarak değişiklik isteklerinin raporlanmasını sağlar.

4.2.3. Kalite Güvence Süreci

Yazılımın planlamalara ve sözleşme içeriğine uygunluğunun güvence altına alınmasını sağlayacak faaliyetlerden oluşmaktadır. (Gümrükçü, 2010)

4.2.4. Değerlendirme Süreci

Bu süreçte, sistemin son durumu ile daha önceden belirlenmiş durumunun birbirlerini tutup tutmadığına bakılır. Projenin büyüklüğü ve değerlendirme birbiri ile doğru orantılıdır. Oluşturulacak sistem, ne kadar büyük ve detaylı ise değerlendirme süreci de o orantıda detaylı ve geniş bir biçimde ele alınmalıdır.

4.2.5. Doğrulama Süreci

Süreçlerin doğru bir şekilde yürütülmesi ile ilgilenir. Doğrulama süreci, sistemin yapılması için gerekli olan işlerin tam olarak doğru olup olmadığından sorumludur. Yapılan işler sonucunda ortaya çıkan verilerle iş yapılmadan önce belirlenen verilerin bir birlerini tutup tutmadığına bakılır.

4.2.6. Gözden Geçirme Süreci

Bir yazılım yaşam döngüsü sürecinde, belirli aşamalarda yapılacakları veya yapılanları iş sahibi ve proje ekibinin bir araya gelerek gerçekleştirdiği toplantılardır. Bu toplantılar proje hakkında değerlendirme ve projenin gidişatı hakkında bilgi alma toplantılarını da kapsar. Bu toplantı süreçleri sözleşme içerisinde hangi periyotlarda olacağı belirtilir ve kayıt altına alınır.

4.2.7. Denetleme Süreci

Yüklenicinin ürünü isteklere ve planlamalara uygun olarak geliştirdiğinin değerlendirilmesi için alıcının gerçekleştireceği faaliyetlerden oluşmaktadır.(Gümrükçü, 2010)

4.2.8. Problem Çözme Süreci

Bu süreç, geliştirme, işletim ve bakım kapsamı içinde karşılaşılan problemlerin çözülmesini kapsar. Bu problemlerin kayıt altına alınır ve çözülüp çözülmediği kontrol edilir.

4.3. Organizasyonel Süreçler

Yazılım yaşam döngüsünün gerçekleştirilmesini, kontrol edilmesini ve iyileştirilmesini sağlamak üzere yönetim, altyapı, iyileştirme ve eğitim süreci olarak dört (4) adet süreç olarak tanımlanmaktadır.

4.3.1. Yönetim Süreci

Yazılım yaşam döngüsünün en genel anlamda etkinlik ve görevlerini tanımlar. Bu süreç planlanmayı, denetimi, gözden geçirme, değerlendirme ve proje kapanışını kapsamaktadır.

4.3.2. Altyapı Süreci

Altyapı kapsamında gerekenlerin tanımlanması, planlanması, uygun zamanda kurulması ve sürecin ihtiyaçlarına göre devam ettirilebilmesi için gerekli faaliyetlerden oluşmaktadır.(Gümrükçü, 2010)

4.3.3. İyileştirme Süreci

Süreçleri değerlendirip ölçerek süreçlerde denetlemelerinin yapıldığı kısımdır. Bu süreçte ana amaç süreçleri geliştirip ihtiyaç doğrultusunda bir yapı oluşturulmasını sağlamaktır.

4.3.4. Eğitim Süreci

Bu süreç, personelin yetişmesi için gerekli teknik eğitimler verilerek personelin yetiştirilmesini amaçlamaktadır.

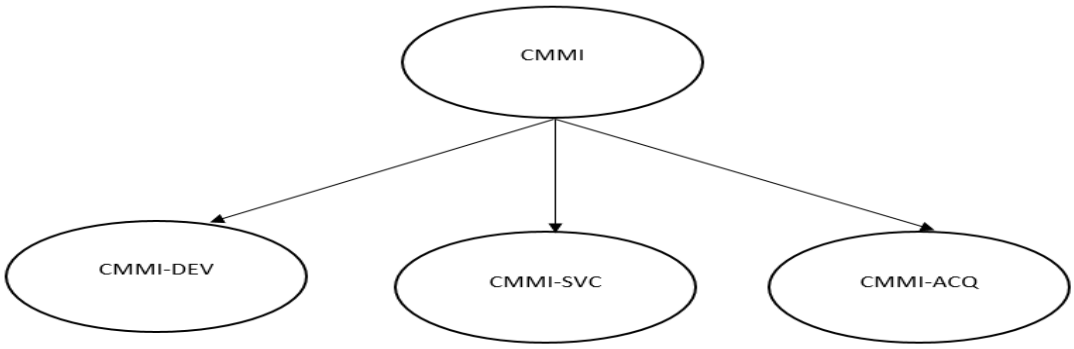
5. BÜTÜNLEŞİK YETERLİLİK OLGUNLUK MODELİ (CMMI)

CMMI, bir süreç modeli olup, kurumların yazılım süreçlerinin olgunluğunu değerlendirir ve bu süreçlerin iyileştirmesi konusunda gereken temel adımları gösterir. Bu model süreçlerinde ne yapılması gerektiğini anlatır fakat nasıl yapılmasından bahsetmez. Bu yaklaşım bir kurumun süreçlerini iyileştirme ve geliştirmeye rehberlik eder.

CMMI kurumların performansını arttıran etkin süreçlerin temel öğelerini destekleyen bir süreç geliştirme modelidir. CMMI bir proje sürecinde, bir kurumun bir alt bölümünde veya bütün bir kurum süreçlerinin geliştirilmesine yol gösteren bir kılavuz olarak kullanılabilir. Geleneksel olarak ayrık olan kurumsal fonksiyonların birleştirilmesine, süreç geliştirme hedefleri ve öncelikleri belirlenmesinde, mevcut süreçlerin değerlendirilmesinde referans noktası sağlama konularında yardımcı olur.(Şimşek, 2011)

CMMI yıldız kümesi adı verilen 3 yaklaşımdan oluşur. Bunlar:
1. Ürün ve Servis Geliştirme — CMMI for Development (CMMI-DEV)
2. Servis Kurulumu, Yönetimi ve Dağıtımı — CMMI for Services (CMMI-SVC)
3. Ürün ve Servis Edinme — CMMI for Acquisition (CMMI-ACQ)

Şekil 22: CMM Yıldız Kümesi



(Omacan, 2008)

- **Ürün ve Servis Geliştirme (CMMI-DEV):** CMMI-DEV, hizmet ya da ürün geliştiren kuruluşlar için tasarlanmıştır. Projelerinin sonunda yeni bir ürün ya da hizmet oluşturan kuruluşlar, bu modelden faydalanabilirler.

- **Servis Kurulumu, Yönetimi ve Dağıtımı (CMMI-SVC):** CMMI-SVC, geliştirilmesi tamamlanmış, müşteriye teslim edilmiş bir hizmetin ya da ürünün, yürütülmesi, bakımının yapılması ya da işletilmesi ile ilgili hizmetler için kullanılmaktadır.
- **Ürün ve Servis Edinme (CMMI-ACQ):** Satın alma çift yönlü bir yoldur. Satın almanın başarısı, tedarikçinin başarısı kadar, satın alma makamının başarısına da bağlıdır. CMMI-ACQ, satın alma süreçlerinin olgunluğunun ölçülmesi ve iyileştirilmesini amaçlayan bir modeldir. Yoğun satın alma yapan kuruluşların faydalanması için tasarlanan bu modelin kökeni General Motors tarafından yapılan bir çalışmaya dayanmaktadır.(Saraçoğlu, 2009)

CMMI modeli mevcut haline gelene kadar birçok aşamadan geçmiştir. Bu aşamalar aşağıdaki tabloda yıl ve gelişim sekmesi şeklinde gösterilmiştir.

Tablo 3:CMMI'ın gelişimin aşamaları

Yıl	**Gelişme**
1987	Yetenek Olgunluk Modeli (Capability Maturity Model, CMM) Carnegie Mellon Üniversitesindeki, Yazılım Mühendisliği Enstitüsü (SEI) tarafından, Amerikan Savunma Bakanlığı için geliştirilmeye başlandı.
1991	Yazılım Yetenek olgunluk modeli (SW-CMM) için ilk sürüm yayınladı.
1995	Kurumsal Süreç İyileştirme İşbirliği (Enterprise Process Improvement Collaboration, EPIC), Sistem Mühendisliği Yetenek Olgunluk Modeli (Systems Engineering Capability Maturity Model, SE-CMM) yayınlandı.
1996	Sistem Mühendisliği Yetenek Değerlendirmesi Modeli (Systems Engineering Capability Assessment Model, SECAM) Sistem Mühendisliği Konseyi (Council on Systems Engineering, INCOSE) tarafından geliştirilip yayınlandı.
1997	Yazılım Yetenek olgunluk modeli (SW-CMM) Versiyon 2 taslağı ve Entegre Ürün Geliştirme Yetenek Olgunluk Modeli (Integrated Product Development CMM) oluşturuldu.
1998	Elektronik Endüstrileri İşbirliği Ara Standardı (EIA 731) oluşturuldu.
1998	İlk takım buluşması yapıldı.

1999	Operasyonların kavramı yayınlandı. İlk deneme tamamlandı.
2000	Ek denemeler tamamlandı. Yazılım Yetenek Olgunluk Modeli (CMMI-SW) ve Sistem Mühendisliği Olgunluk Yetenek Modeli (CMMI-SE) için sürüm 1.0 başlangıç amaçlı olarak yayınlandı. Yazılım Yetenek Olgunluk Modeli (CMMI-SW), Sistem Mühendisliği Olgunluk Yetenek Modeli (CMMI-SE) ve Entegre Ürün Geliştirme Yetenek Olgunluk Modeli (CMMI-IPD) sürüm 1.0 başlangıç amaçlı olarak yayınlandı. Yazılım Yetenek Olgunluk Modeli (CMMI-SW), Sistem Mühendisliği Yetenek Olgunluk Modeli (CMMI-SE), Entegre Ürün Geliştirme Yetenek Olgunluk Modeli (CMMI-IPD) ve Tedarikçi Temini Yetenek Olgunluk Modeli (CMMI-SS) test amaçlı olarak yayınlandı.
2002	CMMI-SE/SW sürüm 1.1 yayınlandı. CMMI-SE/SW/IPPD sürüm 1.1 yayınlandı. CMMI-SE/SW/IPPD/SS sürüm 1.1 yayınlandı. CMMI-SW sürüm 1.1 yayınlandı.
2006	Ürün ve Servis Geliştirme Yetenek Olgunluk Modeli (CMMI-DEV) sürüm 1.2 yayınlandı. CMMI-DEV +IPPD sürüm 1.2 yayınlandı.
2007	Ürün ve Servis Edinme Yetenek Olgunluk Modeli (CMMI-ACQ) sürüm 1.2 yayınlandı.
2010	CMMI-ACQ, CMMI-DEV, CMMI-SVC modellerini kapsayan CMMI V1.3 son ve güncel sürüm olarak 1 Kasım 2010'da yayınlandı.

(Şimşek, 2011)

5.1. Dünyada ve Türkiye'de CMMI Kullanımı

Dünyada CMMI süreç iyileştirme çalışmaları hızla yayılmaktadır. Özellikle Hindistan ve Çin gibi uzak doğu ülkelerinde CMMI sertifikası alan birçok yazılım

firması bulunmaktadır. Aralık 2010 itibari ile dünya üzerinde ki bazı istatistikler şöyledir:

Tablo 4: CMMI Değerlendirmesi Yapılan Ülkeler

Sıra	Ülke	Toplam Değerlendirilme Sayısı	Seviye 1	Seviye 2	Seviye 3	Seviye 4	Seviye 5
1	ABD	1719	30	611	683	23	149
2	Çin	1475	1	142	1213	44	51
3	Hindistan	576	-	19	320	25	197
4	Japonya	324	19	88	146	16	17
5	İspanya	198	1	117	61	3	5

(Tarıkulu, 2011)

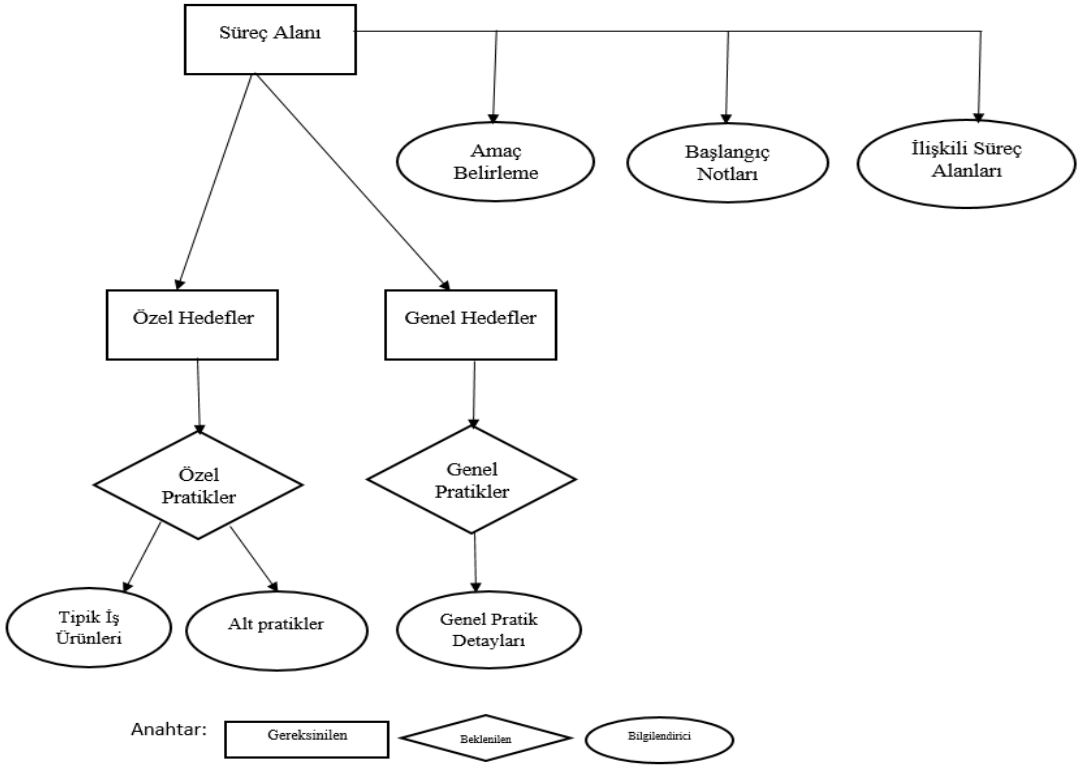
CMMI çalışmalarında ülkemizde artmaya başlamıştır. Öncelikle Savunma Sanayii alanında kullanılmaya başlanmıştır. Ülkemizdeki bazı CMMI bilgileri şöyledir:

- 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 TUBİTAK-UEKAE CMMI Level 3 sertifikası almıştır.
- 2008 yılında TUBİTAK-BTE CMMI Level 3 sertifikası almıştır.
- 2011 yılında SOFTECH Level 3 sertifikası almıştır. (Tarıkulu, 2011)
- 2017 yılında TUBİTAL CMMI Level 5 sertifikasını almıştır.

5.2. CMMI Modelinin Yapısı

CMMI modelinin temel yapısal elemanı "Süreç Alanı"dır (Process Area). CMMI modelinde toplam 22 adet süreç alanı vardır. Model her süreç alanı için birtakım hedefler tanımlamaktadır. Bu tanımlamayı yaparken hedefleri özel ve genel olmak üzere ikiye ayırmaktadır. Genel hedefler; birden fazla süreç alanında sağlanması beklenir ve ilgili olduğu süreç alanlarının kurumsallaşmasını sağlar. Yani o süreç alanlarının etkili, tekrar edilebilir ve kalıcı olmasını sağlar. (SEI CMMI Product Team, 2002) Özel hedefler ise belirli bir süreç alanına karşılık gelir. Bu süreçler 4 kategoriye ayrılır. Proje yönetimi, mühendislik, destek ve süreç yönetimidir.

Şekil 23: CMMI Süreç Yapısı



(Erdoğmuş, 2009)

Tablo 5:CMMI'da süreç kategorileri - süreç alanlarının ilişkisi

Süreç Kategorisi	Süreç Alanları	
Proje yönetimi	• Proje planlama, • Risk yönetimi, • Nicel proje yönetimi	• Proje izleme ve kontrol, • Bütünleşik proje yönetimi, • Tedarikçi anlaşma yönetimi
Mühendislik	• Gereksinim yönetimi, • Teknik çözümleme,	• Gereksinim geliştirme, • Ürün entegrasyonu

	• Doğrulama	• Geçerli kılma
Destek	• Konfigrasyon yönetimi, • Süreç ve ürün kalite güvencesi, • Gerekçe analizi ve çözümleme	• Ölçme ve analiz • Karar analiz ve çözümleme
Süreç yönetimi	• Kurumsal süreç odağı, • Kurumsal eğitim, • Organizasyonel inovasyon ve uyarlama	• Kurumsal süreç tanımı, • Kurumsal süreç performansı

(Tarıkulu, 2011)

5.3. CMMI Modeli Gösterimleri

CMMI modeli iki (2) şekilde gösterilir.

1. Basamaklı gösterim
2. Sürekli Gösterim

5.3.1. Basamaklı Gösterim

CMMI modeli süreç alanlarını "olgunluk seviyesi" adı altında beş gruba ayırmıştır. Bu seviyeler:

Seviye 1 - Başlangıç (Initial): Yazılım süreçleri önceden tahmin edilemez ve süreç kontrolü zayıftır. Kriz durumlarında süreçler takip edilmez. Başarı kişisel yeteneklere bağlıdır.

Seviye 2 - Yönetilen (Managed): Süreçler proje bazında tanımlanır ve buna uygun şekilde gerçekleştirilir. Proje yönetimi ve destek süreçleri ayrıntılı bir şekilde işletilir.

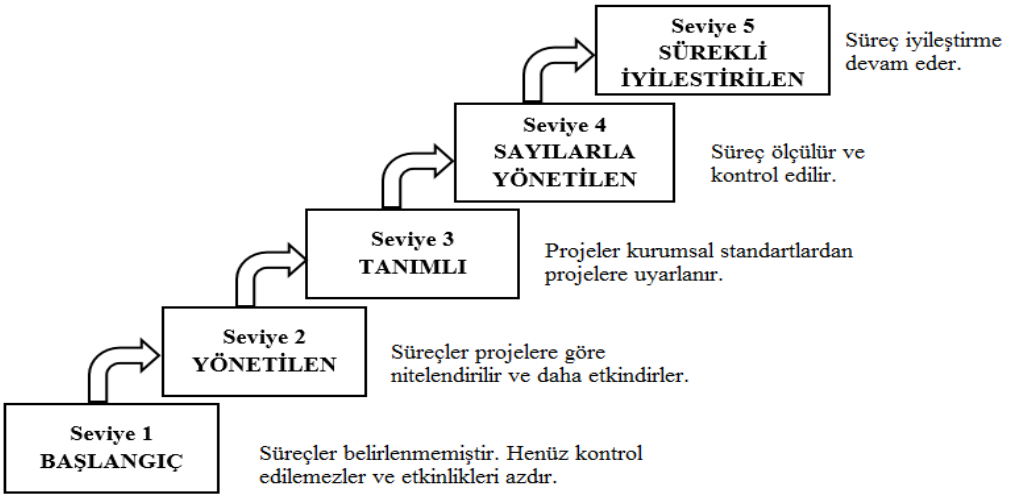
Seviye 3 - Tanımlı (Defined): Süreçler detaylı bir şekilde kurumsal boyutta tanımlanmıştır. Standartlar, prosedürler ve talimatlar belirlenmiştir. Süreçler 2. seviyeye göre daha özenli ve detaylı olarak tanımlanmıştır.

Seviye 4 - Nicel Yönetilen (Quantitatively Managed): Süreçler nicel olarak ölçülmekte ve kontrol edilmektedir. Kalite ve süreç performansının nicel hedefleri belirlenir ve süreçlerin yönetiminde bir kıstas olarak kullanılır. Nicel hedefler sürecin paydaşlarının görüşleri dikkate alınarak belirlenir. Süreçler ölçülür ve tahminler yapılır. 3.seviye ile temel farklılık süreç performansının öngörülebilmesidir.

Seviye 5 - İyileşen (Optimizing): Süreçler ve süreç iyileştirme odak noktasıdır. Bu olgunluk seviyesinde organizasyon, iş hedefleri ve performans ihtiyaçları nicel algılamasına dayalı olarak sürekli iş süreçlerini iyileştirir.(CMMI Product Team, 2002) 5. ve 4.seviye arasındaki önemli bir ayrım, süreçlerin değişme derecesidir. 4.seviyede süreçler, süreç değişkenliğinin özel nedenleri ve sonuçların istatistiksel

öngörüsüne odaklanmıştır. 5.seviyede ise süreçler, süreç değişkenliğinin genel nedenleri ve sürecin değişimini adreslemekle ilgilenir.(Saraçoğlu, 2009)

Şekil 24: Basamaklı gösterim olgunluk seviyeleri



(Kılınç, 2010)

5.3.2. Sürekli Gösterim

Bu gösterim, CMMI modeli süreç alanlarındaki bir sürecin gelişmesini, yetenek seviyeleriyle ifade eder. Altı (6) adet seviyeden oluşur. Bu seviyeler:

Seviye 0 - Eksik (Incomplete): Bu seviye, kısmen başarılmış ya da başarılmamış bir süreçtir. Süreç alanının özel hedeflerin ve genel hedeflerinin uygulanmadığı bir süreçtir.

Seviye 1 - Gerçekleştirilen (Performed): Bu seviye, süreç alanının özel hedeflerini sağlayan bir süreçtir. İş ürünlerini üretmek için gereksinim duyulan işi sağlar ve destekler(Erdoğmuş, 2009).

Seviye 2 - Yönetilen (Managed): Bu seviyede, yönetilen süreç verilen bir amacı gerçekleştirmek için, planlanır, yerine getirilir, izlenir ve bireysel projeler, gruplar ya da bağımsız süreçlerle kontrol edilir. Süreç yönetme, hem süreçlerde model hedeflerini hem de diğer hedefleri gerçekleştirmektir.

Seviye 3 - Tanımlı (Defined): Bu seviyede süreç, organizasyonun politikalarına göre, organizasyonun standart süreçlerinden uyarlanmış bir yönetilen süreçtir. Organizasyonel süreç becerileri için diğer süreç iyileştirme bilgilerine, ölçümlerine ve iş ürünlerine katkı sağlar.(Erdoğmuş, 2009)

Seviye 4 - Nicel Yönetilen (Quantitatively Managed): Bu seviye, sayılarla yönetilir. Sayısal teknikler kullanılarak kontrol edilen tanımlı bir süreçtir. Ürün kalitesi, süreç başarısı ve diğer iş hedefleri yaşam döngüsünden kontrol edilir.

Seviye 5 - İyileşen (Optimizing): Bu seviyede süreçler sürekli iyileştirilir, iyileştirilmiş sayılarla yönetilir. Süreç başarısı iyileştirmeyi hem artırarak hem de yenileyerek süreklileştirmeye odaklanmıştır. Tanımlanmış süreçlerde ve kurumun standart süreçlerinde iyileştirme faaliyet hedefleri vardır. 4. Yeterlilik seviyesi süreç başarısında ölçümlere ve modele temel oluşturmaya odaklanmıştır. 5.Yeterlilik seviyesi başından sonuna kadar kurumun başarı sonuçlarına odaklanmıştır. Sorunların ortak nedenlerini bulup, sorunları gidermek için işlerin nasıl yapılacağına ve işte de ne gibi düzetmeler yapılacağına karar verir. Bu düzeltmeler eğitimlerle ve belgeleri yenilemekle olur. (Kulpa, Johnson, 2003)

5.3.3. Yetenek ve Olgunluk Düzeylerinin Karşılaştırılması

Sürekli modele ait olan yetenek düzeyleri, ayrı süreç alanları içinde bir organizasyonun süreç geliştirme başarısını kapsarlar. Bu düzeyler, belirli bir süreç alanına karşılık gelen süreçlerin adım adım gelişimi için bir araçtır. Altı yetenek düzeyi vardır, 0 ile 5 arasında numaralandırılmışlardır. Basamaklı modele ait olan olgunluk düzeyleri, birçok süreç alanı karşısında bir organizasyonun süreç geliştirme başarısını kapsarlar. Bu düzeyler, üstlenilen bir sonraki projenin genel sonuçlarının tahmin edilmesi için bir araçtır. Beş olgunluk düzeyi vardır, 1 ile 5 arasında numaralandırılmışlardır. Tablo 6'da beş olgunluk düzeyi ile altı yetenek düzeyi karşılaştırılmıştır. Düzeylerin dördünün isimlerinin her iki modelde de aynı olduğu görülmektedir. Basamaklı modelde düzey 0'da olgunluk düzeyi yoktur. Düzey 1'de ise olgunluk düzeyi "Başlangıç (Initial)" iken, yetenek düzeyi "İcra Edilen (Perfomed)" düzeydir. Bundan dolayı, başlangıç noktası her iki model için farklıdır. (Yücalar, 2006)

Tablo 6: Yetenek ve Olgunluk Düzeylerinin Karşılaştırılması

Düzey	**Sürekli Model Yetenek Seviyeleri**	**Basamaklı Model Olgunluk Seviyeleri**
Düzey 0	Eksik (Incomplete)	-
Düzey 1	İcra Edilen (Performed)	Başlangıç (Inital)
Düzey 2	Yönetilen (Managed)	Yönetilen (Managed)

Düzey 3	Tanımlı (Defined)	Tanımlı (Defined)
Düzey 4	Nicel Yönetilen (Quantitatively Managed)	Nicel Yönetilen (Quantitatively Managed)
Düzey 5	İyileşen (Optimizing)	İyileşen (Optimizing)

(Yücalar, 2006)

Sürekli ve basamaklı gösterim modellerinin kıyaslamasını aşağıdaki tablodaki gibi verebiliriz.

Tablo 7: Sürekli ve Basamaklı Gösterimlerin Karşılaştırılması

Sürekli Gösterim	Basamaklı Gösterim
• İş hedefleri ile uyumlu iyileştirme yapma esnekliği tanır. • Her süreç alanının yeteneği hakkında daha görünür bilgi oluşturur. • Yetenek seviyelendirmesi kurum içinde iyileştirme amaçlı kullanılır dış dünya ile pek paylaşılmaz. • Farklı süreçler için farklı hızlarda iyileştirme şansı tanır. • Yapılan yatırım geri getirme (return of investment ROI) açısından, henüz yaklaşım yeni olduğu için bilinmiyor. • ISO 15504(Spice) yaklaşımı ile uyumludur.	• Kurumun önceden tanımlanmış ve etkinliği ispatlanmış bir iyileştirme programı uygulamasını sağlar. • Kurumunun belirli bir olgunluğa gelmesi için gerekli süreç gruplarını tanımlar. • Olgunluk seviyeleri yönetim ve dış dünya ile paylaşılır. • Süreç iyileştirme sonuçlarını basit bir formda belirler. Olgunluk seviyesi sayısı şeklinde belirlenir. • Yapılan yatırım geri getirmenin (return of investment ROI) yüksek olduğu deneyimler ile gösterilmiştir.

(Yücalar, 2006)

6. YAZILIM YAŞAM DÖNGÜSÜ ARAŞTIRMASI, BULGULAR VE YORUM

6.1. Yöntem

Bu bölümde sırasıyla araştırmanın modeline, araştırma ve çalışma grubuna, verilerin toplanmasına ve verilerin analizine ilişkin bilgiler ile bulgular ve yorumlara yer almaktadır.

6.1.1. Araştırma Modeli

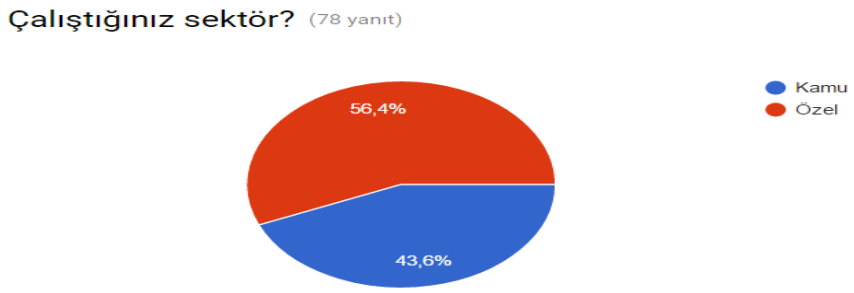
Ülkemizde kamu sektöründe ve özel sektörde yazılım geliştirici olarak çalışan bilişim personelinin "Yazılım Yaşam Döngüsü" konusu hakkındaki düşüncelerini, uygulamalarını öğrenebilmek ve ülkemizde geliştirilen yazılımlarda hangi metodolojilerin kullanıldığı konusunda fikir edinebilmek amacıyla yapılan bu çalışma tarama araştırma modeli kullanılarak gerçekleştirilmiştir.

Tarama modeli bir grubun belirli özelliklerini belirlemek için verilerin toplanmasını amaçlayan çalışmalara tarama araştırması denir. Tarama araştırmasının önemli avantajı oldukça çok bireyden oluşan örneklemden elde edilen birçok bilgiyi bize sunmasıdır (Büyüköztürk, 2012).

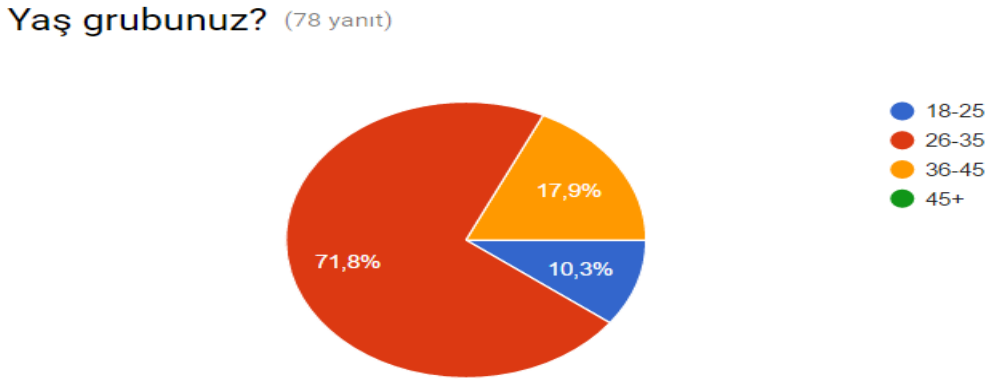
6.1.2. Araştırma- Çalışma Grubu

Ülkemizde aktif olarak yazılım ve proje geliştiren çalışanların "Yazılım Yaşam Döngüsü" konusu hakkındaki görüşlerinin neler olduğunu belirlemek amacıyla kamu ve özel sektör çalışanlarının oluşturduğu 78 kişi araştırmanın çalışma grubunu oluşturmuştur (Son erişim tarihi 30/03/2017). Anılan çalışma grubundan Google Belgeler aracılığıyla çevrimiçi olarak doldurdukları form kullanılarak görüşleri alınmıştır.

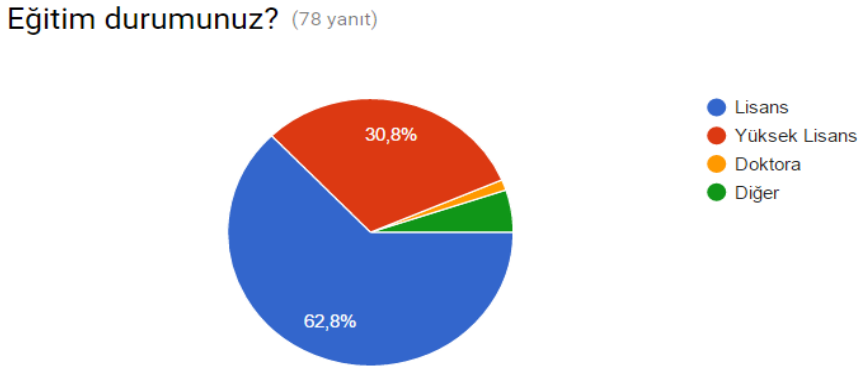
Araştırmaya konu olan yazılım geliştiren bilişim personelinin oluşturduğu çalışma grubunun bazı genel özellikleri ve bu özelliklere ait dağılımlar aşağıdaki gibidir;



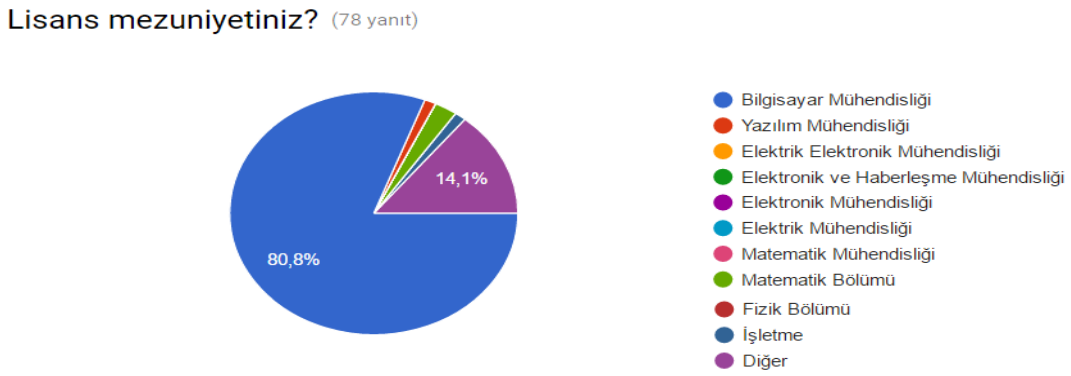
Yukarıda görüldüğü gibi araştırmaya dâhil olan katılımcıların %56,4'ü özel sektörde, %43,6'sı kamu sektöründe çalışmaktadır. Katılımcıların çalıştıkları kurumlarda bilişim ile ilgili birimlerde görev yaptığı bilgisi edinilmiştir.



Toplam 78 kişiden oluşan çalışma grubunun yukarıda görüldüğü gibi %71.8'i 26-35 yaş aralığında, %17.9'u 36-45 yaş aralığında, %10.3'ü 18-25 yaş aralığındadır. Katılımcıların hiç biri 45 yaş üzerinde değildir. Bu sonuçlardan anlaşılacağı üzere yazılım sektöründe çalışanların genel itibariyle genç personellerden oluştuğu görülmektedir.

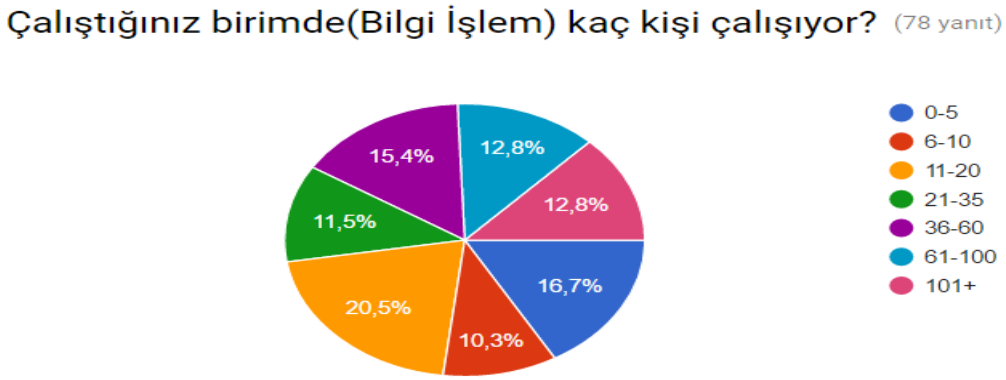


Katılımcıların eğitim durumu incelendiğinde 49 kişinin lisans, 24 kişinin yüksek lisans, 1 kişinin doktora mezunu ve 4 kişinin de diğer seçeneğini işaretlediği görülmektedir.



Çalışma grubunda bulunan katılımcılar lisans mezuniyeti seçeneklerinden 63'ü Bilgisayar Mühendisliği, 11'i diğer, 2'si Matematik Bölümü, 1'i Yazılım Mühendisliği, 1 İşletme bölümü seçeneğini işaretlemiştir. Seçenek grubunda bulunan Elektrik Elektronik Mühendisliği, Elektronik ve Haberleşme Mühendisliği, Elektronik Mühendisliği, Elektrik Mühendisliği ve Matematik Mühendisliği seçenekleri işaretlenmemiştir.

Katılımcıların kurumdaki pozisyonlarına bakıldığında 42'si Yazılım Uzmanı, 24'ü diğer ve 12'si Proje yöneticisi seçeneğini işaretlemiştir.



Çalışma grubundaki katılımcıların çalıştıkları birimdeki kişi sayılarına bakıldığında

ise 16'sı 11-20 kişi aralığında, 13'ü 0-5 kişi aralığında, 12'si 36-60 kişi aralığında, 10'u 61-100 kişi aralığında, 10'u 101+ kişi aralığında, 9'u 21-35 kişi aralığında ve 8'i 6-10 kişi aralığındaki seçeneği işaretlemiştir.

6.1.3. Veri Toplama Aracı

Yazılım geliştirme konusunda çalışan bilişim personelinin görüşlerinin neler olduğunu belirlemek amacıyla "Yazılım Yaşam Döngüsü Anketi" isimli form hazırlanmıştır.

Görüşleri belirlemek amacıyla kullanılan formda aşağıdaki sorular yer almıştır;

1. Yazılım geliştirirken bir yazılım yaşam döngüsü kullanıyor musunuz?
2. Hangi yazılım yaşam döngüsünü kullanıyorsunuz?
   (Şelale Modeli, V-Şeklinde Model, Artırımlı Model, Spiral Model, Prototip Model, Agile Model, Diğer ve Yazılım Yaşam Döngüsü Kullanmıyor)
3. Yazılım geliştirirken yazılım yaşam döngüsü kullanmanın faydalı olduğunu düşünüyor musunuz?
4. Yazılım yaşam döngüsünün yazılım geliştirme sürecine katkısına puan veriniz. (1-En az, 10- En çok)
5. Çalıştığınız Kurum yazılım süreç iyileştirme modellerinden CMMI sertifikasına sahip mi?
6. Çalıştığınız Kurumun CMMI seviyesi nedir?
   (Seviye1, Seviye2, Seviye3, Seviye4, Seviye5 ve Sahip değil)
7. CMMI sertifikasına sahip olmanın yazılım süreçlerinin iyileştirilmesine olan katkısına puan veriniz. (En az 1- En çok 10)
8. Herhangi bir ISO sertifikasına sahip misiniz?
   (ISO 9001 Kalite Yönetim Sistemi, ISO 27001 Bilgi Güvenliği, ISO 12207 Yazılım Yaşam Döngüsü, Diğer ve Sahip değil)
9. Yazılım geliştirmek için kurumunuza özel geliştirilen standart bir dokümanınız var mı?
10. Görüş ve Önerileriniz.

6.1.4. Veri Analizi

Yazılım geliştiricilerin "Yazılım Yaşam Döngüsü" konusu hakkındaki görüşlerinin neler olduğunu belirlemek amacıyla hazırlanan form vasıtasıyla elde edilen verileri betimlemek için içerik analizi yöntemi kullanılmıştır.

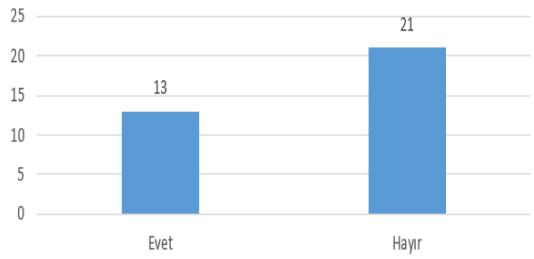
Anket yoluyla elde edilen verileri analiz edebilmek için aşağıdaki adımlar takip edilmiştir;

1. Görüşler konu ile ilgili soruları içeren çevrimiçi soru formu ile toplanmıştır.
2. Soru formundaki sorulara verilen cevaplar benzerliklerine göre kategorize edilmiştir.
3. Her bir soru için belirlenen kategoriler gözden geçirilip tekrarlar giderilerek, birbirine yakın ve benzer kategoriler birleştirilmiştir.
4. Her bir soru için oluşturulan kategoriler, içerikleri dikkate alınarak belli ana başlıklar altında toplanmıştır.
5. Katılımcıların kategorilere ilişkin görüşleri frekans ve yüzde olarak görselleştirilmiştir.

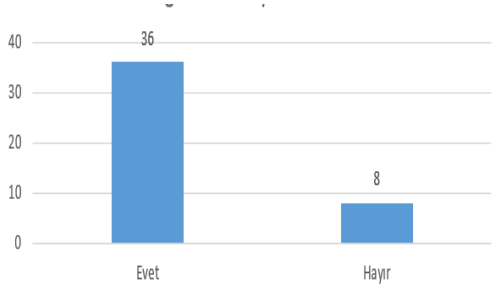
6.2. Bulgular ve Yorum

Araştırmanın bu bölümünde, katılımcıların "Yazılım Yaşam Döngüsü" konusu hakkında görüşlerini belirlemek amacıyla hazırlanan anket formu aracılığıyla yöneltilen sorulara verdikleri cevapları içeren analiz formlarına, her soru özelinde gelen görüşlerin belirlenen kategorilere ayrılarak bu kategorilerdeki dağılımların gösterildiği analiz sonuçlarına ve eldeki verilerin analizi sonucunda ortaya çıkan bulgular ile bu bulgular baz alınarak yapılan yorumlara yer verilmiştir.

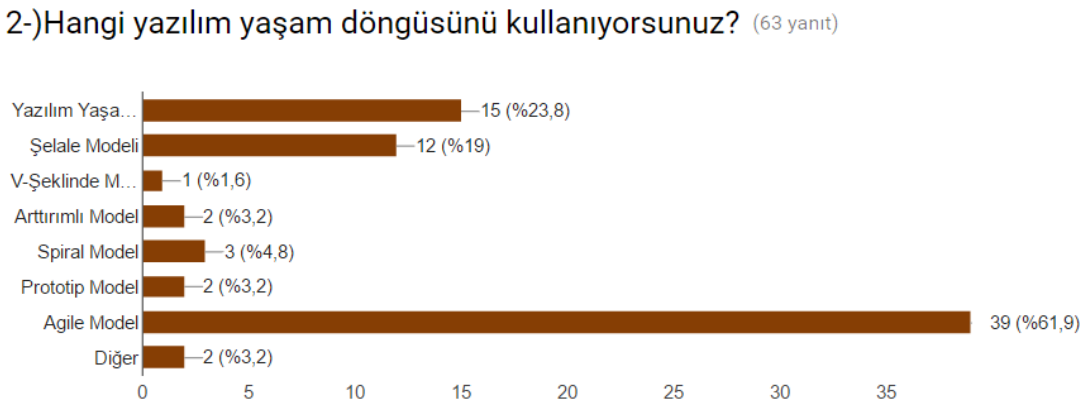
Kamu Sektörü



Özel Sektör

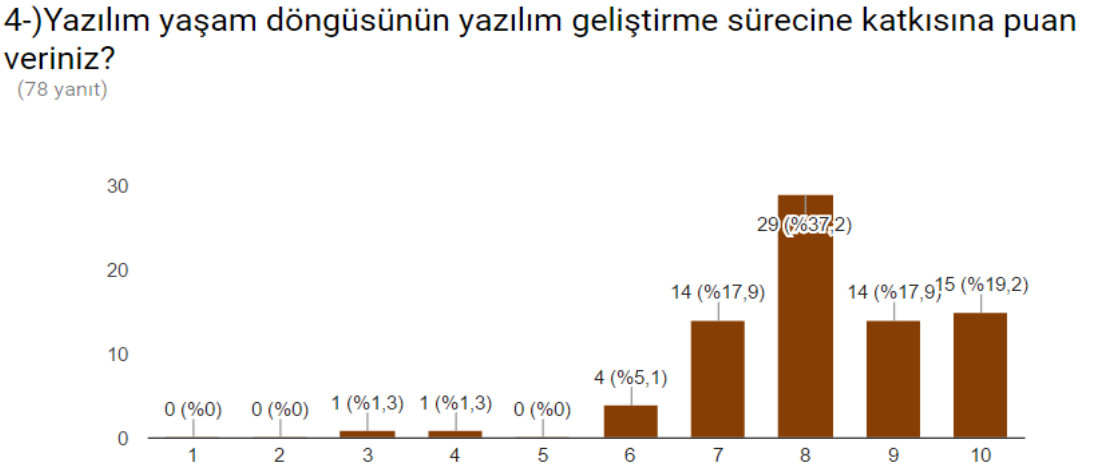


Yukarıdaki grafikte görüldüğü üzere kamu kurumlarından yazılım yaşam döngüsü kullanımının özel sektöre oranla daha az kullanıldığı görülmektedir. Genel değerlendirmede ise katılımcıların 49'u (%62.8) yazılım yaşam döngüsü kullandıklarını 29'u (%37.2) ise yazılım yaşam döngüsü kullanmadıklarını belirtmiştir. Bu sonuçlara bakıldığında genel olarak yazılım geliştirmeye bir disiplin kazandıran yazılım yaşam döngüsü kullanımının yaygın olduğu görülmektedir.



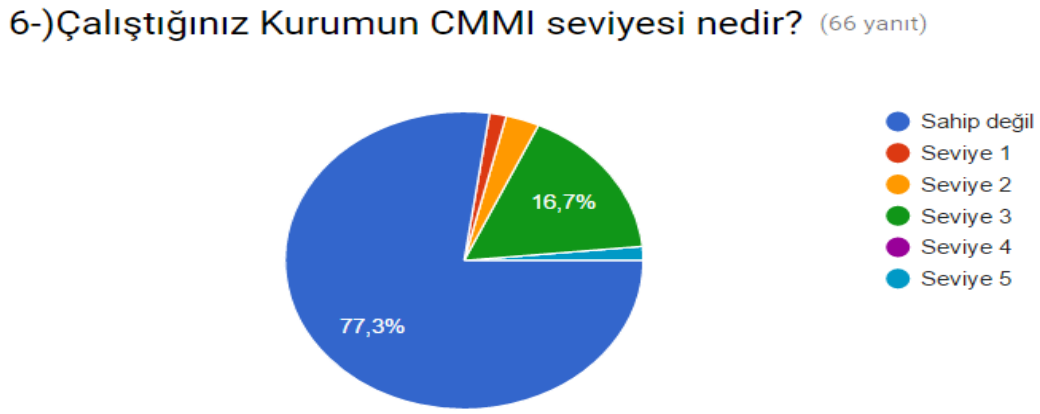
Yapılan araştırmaya göre yaygın olarak agile model tercih edilmektedir. Katılımcıların 39'u (%61.9) agile modeli, 12'si (%19) şelale modeli, 3'ü (%4.8) spiral modeli, 2'si (%3.2) prototip model, 2'si (%3.2) artırımlı model, 2'si (%3.2) diğer, 1'i (%1.6) v şeklinde model seçeneğini işaretlemişlerdir. Katılımcıların 15'i(%23.8) yazılım yaşam döngüsü kullanmadığını belirtmiştir. Son dönemlerde yazılım sektöründe çevik(agile) yöntemlerin yaygınlaşması anket sonuçlarında da kendini göstermiştir.

Ankete katılan tüm katılımcılar yazılım yaşam döngüsü kullanmanın faydalı olduğunu düşünmektedir. Anket sonuçlarından da anlaşılacağı üzere yazılım yaşam döngüsü kullanmanın kaliteli bir yazılım geliştirmek için zorunluluk haline geldiğini söyleyebiliriz.



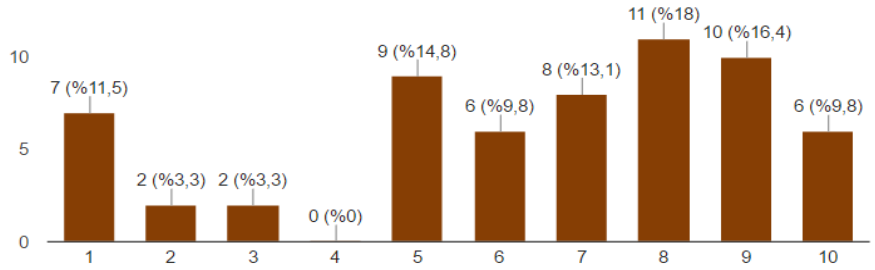
Katılımcıların tamamı yazılım yaşam döngüsü kullanmanın faydalı olduklarını düşündükleri halde yazılım geliştirme sürecine katkısına en fazla 29 kişi %37.2 oranında 10 üzerinden 8 puan vermişlerdir. Buradan yazılım yaşam döngüsü kullanmanın yazılım yaşam döngüsü iyileştiren bir süreç olduğu fakat tek başına yeterli olmadığı gözlenmektedir.

Ankete göre CMMI sertifikasına sahip olma oranı çok düşüktür. Yazılım yaşam döngüsü genellikle kullanılmasına rağmen CMMI süreçlerinin kurumlarda işletilmesinin zor olmasından dolayı CMMI sertifikasına sahip olma oranı düşüktür. Katılımcıların 66'sı (%84.6) hayır seçeneğini 12'si (%15.4) evet seçeneğini işaretlemiştir.



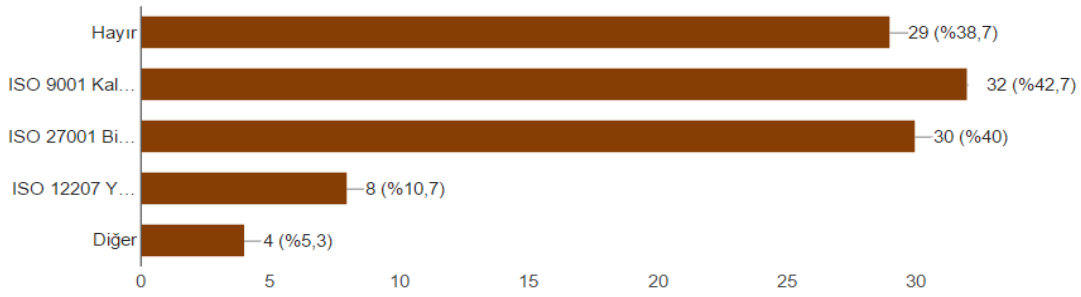
Katılımcıların 51'i (%77.3) CMMI seviyesine sahip değil, 11'i (16.7) seviye 3, 2'si (%3) seviye 2, 1'i(%1.5) seviye 1 ve 1'i(%1.5) seviye 5 seçeneğini işaretlemiştir. Bir önceki sorudan anlaşılacağı üzeri CMMI sertifikasına sahip olma oranı düşük olduğu için CMMI seviyelerine sahip olma oranları da düşüktür. Burada yazılım sektöründeki kurumlar CMMI modelinde olgunluk seviyesi olarak her kurum seviye 1 olarak kabul edilir. Sahip değil seçeneğinin büyük oranda işaretlenmiş olması CMMI süreci hakkında yeterli bilgi birikimine sahip olunmadığı göstermektedir.

7-)CMMI sertifikasına sahip olmanın yazılım süreçlerinin iyileştirilmesine olan katkısına puan veriniz?
(61 yanıt)

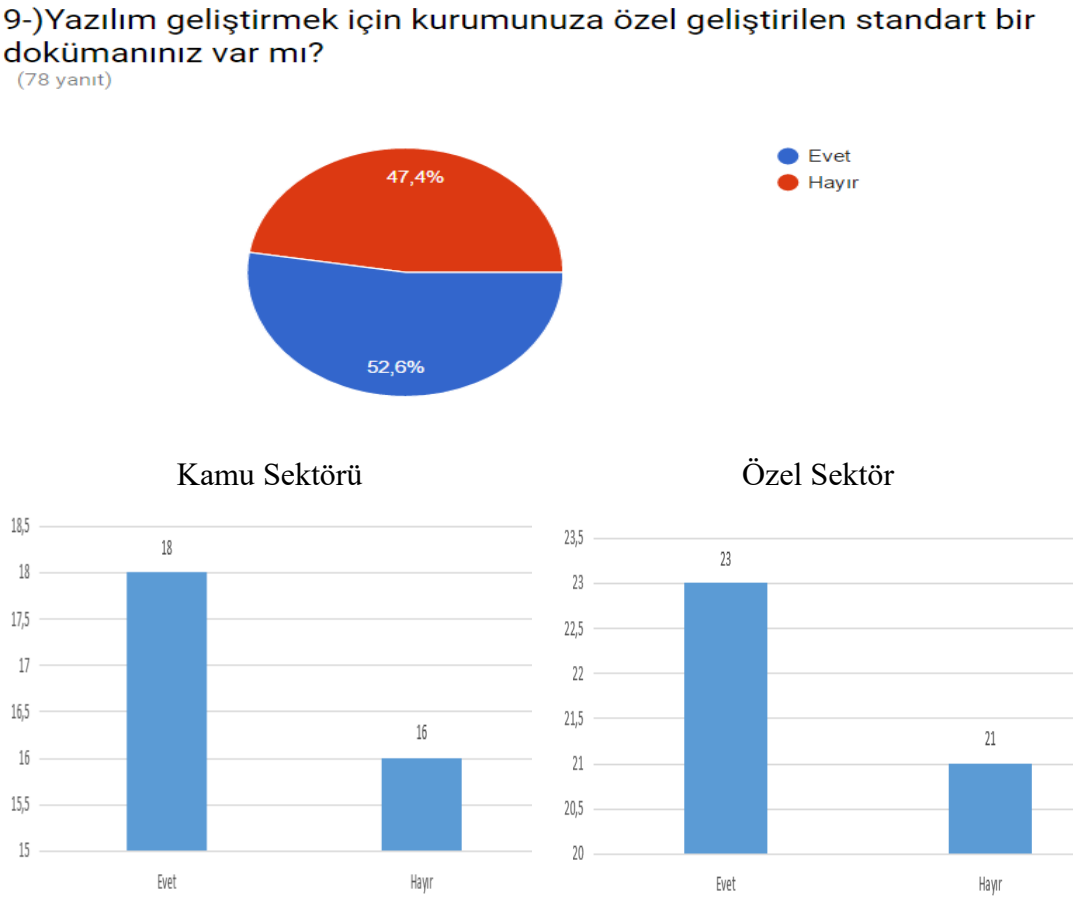


Katılımcıların CMMI sertifikasına sahip olmanın yazılım süreçlerine iyileştirmesine olan katkısına verdikleri puanlara bakıldığında 4.sorudaki yazılım yaşam döngüsü kullanmanın etkisine göre daha olumsuz bir bakış açısı sergilenmektedir. Yazılım yaşam döngüsünün yazılım sürecine katkısına katılımcılar yaklaşık % 75'i 8 ve üzeri puan vermelerine rağmen CMMI için 8 ve üzeri puan verenler yaklaşık %45 olduğu görülmektedir. CMMI sürecini işletmenin zor olması ya da süreç hakkında yeterli bilgi sahibi olunamaması sonuçların bu şekilde çıkmasına sebep olmuş olabilir.

8-)Herhangi bir ISO sertifikasına sahip misiniz? (75 yanıt)



Katılımcıların 32'si (%42.7) ISO 9001 Kalite Yönetim Sistemi, 30'u (%40) ISO 27001 Bilgi Güvenliği, 29'u (%38.7) hayır, 8'i (%10.7) ISO 12207 Yazılım Yaşam Döngüsü ve 4'ü (%5.3) diğer seçeneğini işaretlemiştir. Anket sonucuna göre kurumlar genellikle ISO sertifikalarına sahip olma eğilimindedir. Süreçlerini standart haline getirmek için belli dokümantasyon yapıp bu düzene uymanın yazılım sürecini iyileştirdiği söylenebilir.



Üstteki şekilde görüldüğü üzere hem kamu sektöründe hem de özel sektörde yazılım geliştirirken kuruma özel bir dokümana sahip olma eğilimi gözlenmektedir. Ankete katılan katılımcıların 41'i(%52.6) evet 37'si(%47.4) hayır seçeneğini işaretlemiştir.

10. soru olan Görüş ve Önerilerinize verilen cevapların bir kısmı aşağıdaki tabloda verilmiştir.

Görüşlerden Elde Edilen Temel İfadeler
"ISO sertifikasına sahibiz" veya "yaşam döngüsü kullanıyoruz" ifadeleri kâğıt üzerinde kalabilir. Bu nedenle asıl soru bu süreçleri/iyileştirmeleri projelerimize/iş yaşamımıza ne kadar entegre ettiğimizdir. Yani kurumumuzda CMMI sertifikası olması CMMI gerekliliklerini yerine getirdiğimizi göstermez. Personelin bu gereklilikleri/yöntemleri benimseyip tüm iş süreçlerine entegre etmesi sonucu yapılan işler kaliteli/profesyonel hale gelecektir. Yetersiz/bilgisiz personeller/yöneticiler olduğunda kamu düzeninde bunların uygulanması çok zordur.
Yazılım yaşam döngüleri mutlaka uygulanmalıdır.

Yazılım yaşam döngülerinin, yeterli personel planlaması ile eksiksiz uygulanması gerekir.
Sertifikası olanlar dahil bir kere alıp çerçeveletiyor sonrası klasik doğrudan kodlama oluyor.
Sistemin sağladığı düzen çok iyi ve bence gerekli fakat sistemin verimi için gereken özveri ve iş yükünü dengelemek gerekiyor.

Bu görüş önerilere bakıldığında personel ya da yöneticilerin eğitimli olması durumunda yeterli personel ile planlanıp iş yükünün dengelenerek alınan sertifikaların gereklerini yerine getirildiğinde bu sürecin yararlı olacağından bahsedilmektedir.

Yapılan Yazılım Yaşam Döngüsü Araştırmasının sonucunda ulaşılan en çarpıcı husus katılımcıların tamamının yazılım yaşam döngüsü kullanımın faydalı olduğunu düşünmesidir. Yazılım yaşam döngüsünün yazılım süreçlerini iyileştirmesi büyük oranda olumlu bakılırken CMMI sertifikasına sahip olmanın bu süreç kadar etkisi olacağı düşünülmemektedir. Genel olarak yazılım yaşam döngüsü kullanma, CMMI sertifikasına sahip olma, ISO sertifikasyonu ve yazılım süreçleri için standart bir dokümana sahip olma eğilimi gözlenmektedir. Bu süreçlerin genel olarak yaygınlaşmasının yazılım süreçlerine olumlu etkisinden dolayı olduğunu söyleyebiliriz.

SONUÇ

Bu bölümde Bakanlıktaki mevcut durum, Bakanlık için öneriler ve sonuç kısımlarından bahsedilecektir.

Mevcut Durum

Bakanlığımız bünyesinde geliştirilen ve Bakanlık dışından satın alınan her türlü uygulama yazılımı ile ilgili çalışmalar Coğrafi Bilgi Sistemleri Genel Müdürlüğü(CBS) altında bulunan Yazılım Teknolojileri Dairesi Başkanlığındaki Yazılım Geliştirme Şubesi ve Bilişim Projeleri Koordinasyon Şubeleri ve ilgili birim tarafından yürütülmektedir. Bakanlığımız bünyesinde bulunan uygulama yazılımlarının nasıl geliştirildiği tespit edilmeye çalışılırken Yazılım Geliştirme şubesinden alınan bilgilere başvurulmuştur.

Yazılım geliştirme süreci için ilgili birim CBS Genel Müdürlüğüne resmi yazı ile talepte bulunur. Yazılım Geliştirme Şubesi tarafından mevcut kaynaklar ve iş yükü doğrultusunda yazılımın istenilen zamanda geliştirip geliştiremeyeceğine dair resmi yazı ile dönüş yapılır. Mevcut durumda uzman teknik personel sayısı artırılmaya devam etmekte gelen talepler geri çevrilmeden geliştirme yapılmaya çalışılmaktadır. Eğer yazılım geliştirme şubesi uygulamayı geliştiremeyecekse ilgi birim Bilişim Koordinasyon Kuruluna bu talebi iletir. Müsteşar başkanlığında toplanan Bilişim Koordinasyon Kurulunda talep kabul edilirse satın alma yoluyla temin yoluna gidilir. Yazılımla ilgili şartname ilgili birim tarafından hazırlanır. Teknik hususlarda görüş almak için 2014-14 sayılı Bilişim ve Coğrafi Bilgi Sistemi Projeleri Yürütme ve Koordinasyon Esasları Genelgesine istinaden CBS genel müdürlüğüne gönderilir. Teknik görüş verildikten sonra satın alma süreci gerçekleşir. Satın alınan yazılımlar yüklenici firma tarafından .NET teknolojisi kullanılacaksa bakanlıkta kullanılan ASP.Net MVC teknolojisi ile oluşturulmuş altyapı üzerinde geliştirilir. Burada yazılım geliştirilirken Yazılım Teknoloji Dairesi tarafından çıkarılmış olan "Yazılım Geliştirme ve Birlikte Çalışma Esasları Prosedürüne" uygun olarak yazılım geliştirilir.

Yazılım Geliştirme Şubesi tarafından geliştirilecek yazılımlar için ise resmi yazıyla başvurulduktan sonra ilgili birim yazılım geliştirme şubesi tarafında çıkarılmış "Uygulama Geliştirme Formu"nu doldurur. Analiz yapılması için ilgili birim ile iletişime geçilir ve analiz çalışması yapılır. Analiz çalışmasında ilgi birim ne yapmak istediği genel hatlarıyla anlatır. Teknik personel ile büyük resim detaylandırılmaya

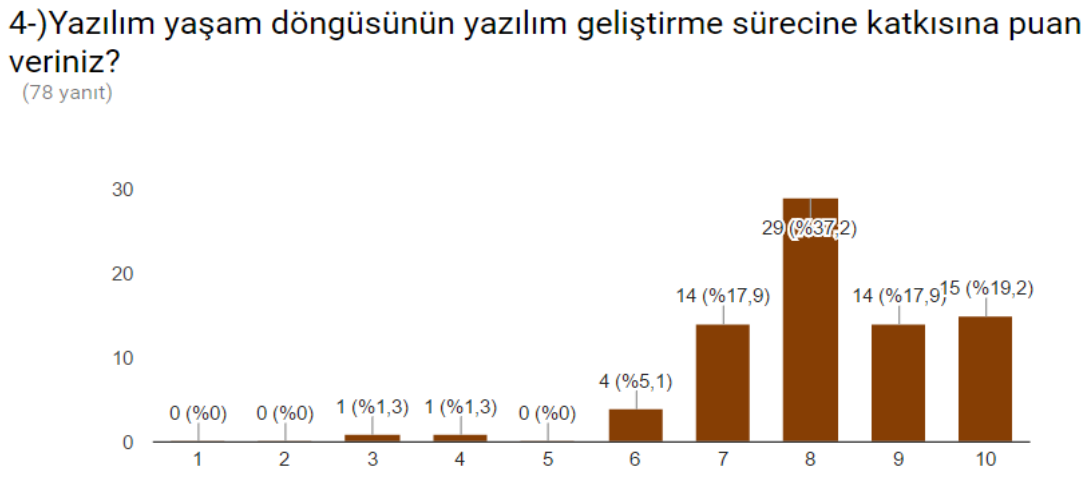
çalışılır. Mutabık kalınan analiz çalışması yazılı kayıt altına alınır. Bu talepler yazılım takip aracına yüklenir. Sonra teknik personel bu talepler doğrultusunda prototip çalışması yaparak ilgili birimle iletişime geçilir. İstenilen uygulamanın bir modeli incelenir, talepler dikkate alınır ve yazılım geliştirme süreci devam eder. Yazılım modüler olarak geliştirilir. Her modülden sonra birim testleri yapılır. Yazılım tamamlanma aşamasına geldiğinden sistem testleri yapılır. Daha sonra kullanıcı testleri ilgili birim tarafından yapılır. Artık çalışan yazılım ekip lideri gözetiminde test ortamından alınıp yayınlanma ortamına alınır. Yazılım geliştiriciler yayınlanma ortamında yazılıma müdahale etmezler. Uygulama kullanıcılarının kullanımına açılır. Yazılım için talepler geldikçe gelen talepler yazılım takip aracına yüklenir ve yazılım güncelleştirmeleri yapılır. Bu yazılım güncelleştirmeleri sürüm takip sistemlerinden takip edilir. Yazılım bittikten sonra yazılıma ait teknik bir doküman hazırlanır. Bu doküman içinde uygulama genel hatları ile açıklanır. Bu doküman uygulamada yer alan modülleri, kullanıcı tiplerini, uygulamanın dışardan aldığı servisleri, uygulamanın dışarıya verdiği servisleri ve uygulamada dikkat edilecek hususları içerir. Bu doküman yaşayan bir dokümandır. Uygulama geliştikçe doküman içeriği de değişmektedir.

Öneriler

Bakanlığın mevcut durumu, kamu kurumu ziyaretleri ve yapılan anket çalışması göz önünde bulundurulduğunda; yazılım geliştirme personelinin sayısı, idarenin çalışma sahası genişliği, sahip olunan yazılım sayısı ve mevcut personelin iş yükü incelendiğinde Bakanlığımızın teknik personel sayısı arttırılmalıdır. Yine bu teknik personelin gelişimlerinin arttırılması için personel tarafından talep edilen eğitimler belli aralıklarla yapılmalı ve kişisel gelişime destek olunmalıdır. Bu destek sayesinde kendini geliştirebilen personelin işe olan aidiyeti artacak bunun yanında moral ve motivasyonu arttığı için yapılan işlerde kalite artacaktır. Bu şekilde kurumsal süreçlerde iyileşmeler gözlemlenebilecektir.

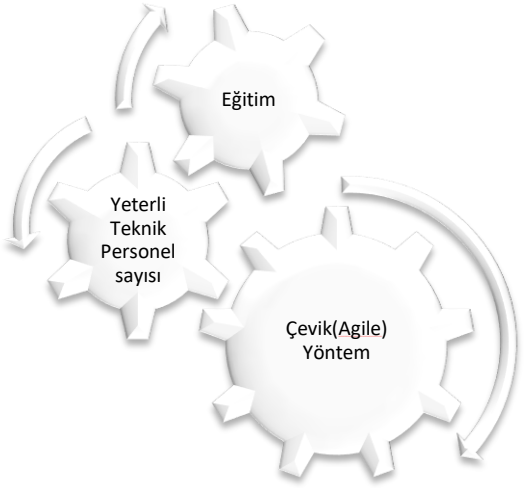
Satın alma yoluyla gerçekleştirilen yazılımlar için kullanabilecek yaşam döngüsü için ilgili mevzuat gereği teknik şartname ile oluşturulması sebebiyle “Şelale Modeli” kullanılarak yazılımın takibi yapılması uygun olacaktır. Bu modelin uygulanması Bakanlığın mevcut durumu göz önünde bulundurularak uyarlanmalı ve yapılan işlemler kayıt altında tutulmalıdır. Bu takiplerin yapılması için uygulama takip yazılımları kullanılmalıdır.

Bakanlık tarafından geliştirilecek olan yazılımlarda ise ilgi birim taleplerinin başlangıçta net olmaması durumunda süreçlerin zaman içinde değişebileceği göz önünde bulundurularak ve kısa zamanda sonuç görülme isteği sebebiyle çevik(Agile) yöntemlerden herhangi biri kullanılabilir. Anket çalışması için yapılan saha ziyaretleri kapsamında genellikle çevik yöntemlerden scrum yönteminin tercih edildiği görülmüştür. Aynı şekilde Bakanlığımız tarafından süreçlerin bir yazım yaşam döngüsü süreçlerine bağlı kalınarak scrum yönteminin Bakanlığımıza uyarlanarak uygulanması uygun olacaktır. Mevcut durumda yazılım geliştirme sürecinde çevik yöntem uygulanmaya çalışılıyor ve araçlar ile takip yapılıyor ama bunun seçilen bir yazılım yaşam döngüsünün bütün süreçlerinin kuruma uyarlanarak ve bu uyarlanan süreçlerin sıkı bir şekilde yapılıp araçlar üzerinde takip edilmesi yazılım geliştirme sürecini iyileştirecektir. Mevcut durumun bir çevik yönteme entegre edilmesi kolay olacaktır. Yapılan saha çalışmasındaki ankete katılan teknik personelinde 3. Sorunda geçen "Yazılım geliştirirken yazılım yaşam döngüsü kullanmanın faydalı olduğunu düşünüyor musunuz?" ve 4.soruda geçen "Yazılım yaşam döngüsünün yazılım geliştirme sürecine katkısına puan veriniz?" sorularının cevaplarına bakıldığında;



Bu şekilde cevaplar ortaya çıktığı görülmüştür. Yani ankete katılanların tamamı yazılım yaşam döngüsü kullanmanın faydalı olduğunu söylemiştir. Yine aynı şekilde yazılım geliştirme sürecine etkisine %90'nından fazlası 7 ve üzerinde puan vermiştir. Bunun sonucu olarak seçilen çevik yöntemin kuruma uyarlanmasının yazılım süreçlerini iyileştireceği düşünülmektedir. Yazılım yaşam döngüsü uygulandığında Bakanlığın yazılım geliştirme olgunluk seviyesini ve dijital olgunluk seviyesini artacaktır. Mevcut durumda uygulama geliştirilirken çevik yöntem ilkeleri uygulanmaya başlanmış olması seçilecek olan bir çevik yönteme entegre olmasını kolaylaştıracaktır. Bakanlık bünyesinde geliştirilecek yazılımlar için sahip olunan uygulama sayısını ve teknik personel sayısını göz önünde bulundurduğumuzda yapılan önerileri bir şema üzerine gösterecek olursak Şekil 25'deki gibi olur.

Şekil 25: Bakanlık İçin Yapılan Öneri Şeması



Şekilde görüldüğü gibi mevcut uygulama sayısı ve iş yükü göz önünde bulundurularak yeterli sayıda teknik personele sahip olunması ve personelin eğitim ile birlikte geliştirmesi sonucunda seçilecek olan çevik yöntemin uygulanması yazılım geliştirme süreçlerini makine dişlilerinde olduğu gibi sorunsuz bir şekilde işletecektir. Seçilecek çevik yöntem kuruma uyarlandıktan sonra bu süreçlere için roller ait personeller belirlenmelidir. Bu rollerden biri olan genellikle uzman ibaresi şeklinde geçen role ait kişi belirlenen çevik yöntemin süreçlerinin takip edilmesini ve geliştirme aşamasında bu süreçlerin dışına çıkılmamasını sağlamalıdır. Bu şekilde seçilmiş olan çevik yöntemin belirlenen süreçleri bütün şekliyle uygulandığında daha verimli yazılım geliştirme sağlanmış olur. Proje tamamlamak geçen sürelerde iyileşmeler olur. Bu süreçler uygulanırken zaman çizelgeleri sürekli projenin gelişimi ve durumu takip altında tutulmalıdır. Sonuç olarak değişen ve gelişen zamana göre yazılım dünyasının esnek yapısından dolayı çevik yöntemlere geçiş bir zorunluluk halini almıştır. Bu yöntemlerden Scrum yönteminin seçilip bakanlık için uygulanması öngörülmektedir. Bu yöntem Ek-1: Bakanlık için Scrum Akış Diyagramı ve Ek2: Bakanlık için Scrum Yönteminin Uygulama adımlarına göre uygulanmalıdır. Bakanlıktaki projelere Scrum yöntemini uygulanırken Ek-3: Örnek Proje için Scrum Yöntemi dikkate alınmalıdır. Bu projeler bakanlık bünyesinde bulunan proje takip uygulaması olan Jira programında ise Ek-4: Örnek Projenin Jira programında takip edilmesine göre uyarlanabilir.

Bakanlığa bu yöntemin uygulanması sağlandığında ve bu süreçlerin sıkı takibi yapıldığında projelerin gelişiminde esneklik yeterli sayıdaki personel ile sağlanmış olacaktır. Bu şekilde ilgili birimlerden gelen taleplere daha hızlı ve esnek bir şekilde cevap verilip proje tamamlama süreleri mevcut durumlarına göre daha da kısalacaktır. Projenin zaman içerisindeki gelişimi takip edilerek mevcut iş yükü görülebilecektir. Buna göre yazılım projelerinde zaman planlaması daha doğru bir şekilde yapılabilecektir.

Sonuç

Bilgi çağının sonucu olarak değişen ve gelişen dünyada bilişim alanında yeniliklere ayak uydurmak zorunluluk halini almaktadır. Kamu kurumlarının da bu değişime ayak uydurması amacıyla Ülkemizde bu yönde adımlar atılmış olup E-Devlet çalışmaları başlamıştır. Bu çalışmalarla ilgili en son olarak Ulaştırma Denizcilik ve

Haberleşme Bakanlığı, bilgi toplumu politikası çerçevesinde Türkiye'nin e-Devlet politikasının bütüncül bir bakış açısı ile şekillendirilmesi için 2016-2019 Ulusal e-Devlet Stratejisi ve Eylem Planı'nı hazırlamıştır.

Yazılımların bu çağın en önemli enstrümanlarından biri olduğunu göz önüne aldığımızda geliştirdiğimiz yazılımların gelişen teknolojiye uygun olması gerekmektedir. Ulusal e-Devlet Stratejisi dikkate alındığında geliştirilecek yazılımların kalitesini arttırmalı, süreçlerini iyileştirmeli, çıkacak son ürünün talepleri karşılamalı ve kamu işlevlerini dijital ortamda sağlıklı bir şekilde yürütülmesini sağlamalıdır.

Bu kapsamda yapılan tez çalışmasında yazılım yaşam döngüsü modelleri, çevik yöntemler, CMMI ve yazılım yaşam döngüsü standardı açıklanmıştır. Bunun yanında "Yazılım Yaşam Döngüsü" konusunda yapılan anket çalışmasıyla yazılım geliştirme yapan birimlerden ilgili teknik personelin düşünceleri öğrenilmeye çalışılmıştır. Anket çalışması yapılırken kurum saha ziyaretleri yapılmıştır. Saha ziyaretleri kapsamında kamu kurumları ziyaret edilmiş olup yazılım geliştirme süreciyle ilgili yaklaşımları araştırılmıştır. Saha çalışması kapsamında Sosyal Güvenlik Kurumu(SGK), Türkiye Atom Enerjisi Kurumu(TAEK), Başbakanlık Afet ve Acil Durum Yönetim Başkanlığı(AFAD), Posta ve Telgraf Teşkilatı Genel Müdürlüğü(PTT) ve Türksat Uydu Haberleşme Kablo TV ve İşletme A.Ş (TÜRKSAT) yazılım geliştirme konusunda yaptıkları çalışmalarla ilgili yerlerinde görüşmeler yapılmıştır. Ayrıca Türkiye Bilimsel ve Teknolojik Araştırma Kurumu Ulusal Akademik Ağ ve Bilgi Merkezinden(TÜBİTAK) konu ile ilgili çalışmaları hakkında bilgi alınmıştır.

Yapılan saha çalışmalarında kamu kurumlarında yazılım geliştirme süreçlerinin sistematik yapılması, araçlar üzerinden takibi ve seçilen bir yazılım yaşam döngüsünün uygulanması, eğitimli personel sayısı fazla olan kurumlarda uygulanmakta olduğu görülmüştür. Yine bazı kurumlarda proje bazlı uygulandığı tespit edilmiştir. Süreçlerin daha sağlıklı yapılabilmesi için dokümanlar yazıldığı görülmüştür. Bazı kurumlarda benimsenen çevik yöntemin Bakanlığımızda olduğu gibi uygulanmaya çalışıldığı tespit edilmiştir. Teknik personel sayısı az olan kurumlarda yüklenici firma üzerinden geliştirme ve bakımların yapıldığı tespit

edilmiştir. Genel olarak teknik personel geliştirilmesi için hizmet içi eğitimlerin belli aralıklarla verildiği gözlemlenmiştir.

Yapılan çalışma sonucunda elde edilen bulgular göz önünde bulundurulduğunda karşımıza çıkan sonuçlar aşağıdaki gibidir;

- Yazılım yaşam döngüsü konusu kurumlar tarafından ciddiyetle ele alınmış bu yönde çalışmalara başlanmıştır. Bu kapsamda bir yazılım yaşam döngüsü modeli seçilip kurumun işleyişine uygun modellenip uygulanmalıdır. Bu model seçilirken esnek yapısı nedeniyle çevik yöntemlerden birisi tercih edilmelidir. Saha çalışması ve anket çalışması sonucuna göre bu çevik yöntemlerden Scrum yöntemi Ek-1: Bakanlık için Scrum Akış Diyagramı ve Ek2: Bakanlık için Scrum Yönteminin Uygulama Adımlarına göre uygulanmalıdır. Yeni projeler uygulanırken örnek projenin uygulandığı Ek-3: Örnek Proje için Scrum Yönteminden ve Ek-4: Örnek Projenin Jira programında takip edilmesinden yararlanılmalıdır.
- Yazılım geliştirme süreci gelişen değişen bir süreç olduğundan eğitimler belli aralıklarla yapılmalıdır. Bu eğitimlerin konusu teknik personelin talebi dikkate alınarak belirlenmelidir. Ve bu eğitimler belli aralıkla sürekli yapılarak teknik personelin kendini sürekli olarak geliştirilmesi sağlanmalıdır. Teknik personeller belirlenirken test ve analiz personeli yazılım geliştirme personelinden farklı olarak seçilmelidir. Bu kapsamda uzmanlaşmaları sağlanmalıdır.
- Sahip olunan uygulamalar ve mevcut iş yükü göz önüne alınarak yeterli olacak personel sayısı belirlenmelidir. Belirlenen teknik personel sayısına ulaşıldığında satın alma yoluna gidilmeden kurum kendi iç kaynaklarıyla bu taleplere karşılık vermesi sağlanmalıdır.

Bu çalışma Bakanlık mevcut durumu göz önünde bulundurularak yapılmıştır. Bakanlığımız bilişim dünyasındaki değişmelerin farkında olup çalışmalara başlamıştır. Bu çalışmalar sayesinde belli modeller benimsenmesi ve bu modellerin uygulamaya geçirilmesi hızlı bir şekilde olabilecektir. Yine bu süreçlerin yazılım takip araçları ile takip edilmesi çalışmaları başlamıştır. Bu süreçler zaman içinde oturduğunda bakanlığımızı olgunluk seviyesi artacağı öngörülmektedir.

Bu tez çalışması, Bakanlığımızın yapısına en uygun yazılım yaşam döngüsü seçimi, seçilen bir yazılım yaşam döngüsünün Bakanlığımıza uygulanması ve bütünleşik yeterlilik olgunluk modelinin (CMMI) kamu kurumlarında uygulanması gibi tez konularına referans olacaktır.

KAYNAKÇA

ALTUN Nefize, 2004, Proje Tasarımı ve Yönetimi için Veritabanı Sistemi Yüksek Lisans Tezi syf:9

ANARAL Furkan, 2011, Çok Kriterli Karar Verme Yöntemi İle Yazılım Geliştirme Metodolojisi Seçimi Yüksek Lisans Tezi syf:8

ANDERSON David J.,2004, Feature-Driven Development: Towards a TOC, Lean and Six Sigma Solution for Software Engineering, Microsoft Corporation

ARDANT Eric Lefevre, 2008, Is Scrum evil, http://ericlefevre.net/wordpress/2008/10/07/scrum-is-evil/, (22/03/2017)

BABAN Nalkar Sanjivani, 2016, Processing Models Of SDLC, http://www.csimumbai.org/newstrack/April-June-2013/SDLC%20-%20Big%20Data.pdf, (21/12/2016)

BAKAN Ali, 2007, Ekstrem Programlama ve Ekstrem Programlama Projelerinin Yönetimi Yüksek Lisans Tezi, syf: 8-10

BLANCHARD Benjamin S., WOLTER J. Fabrycky, 2006, Systems engineering and analysis 4th ed. New Jersey:Prentice Hall

BÜYÜKÖZTÜRK Şener, 2012, Bilimsel Araştırma Yöntemleri. Ankara: PegemAkademi syf:14

CHRİSTENSEN Mark J.,THAYER Richard H., 2001, The Project Managers Guide to Software Engineering's Best Practices, IEEE

CMMI Product Team, 2010, CMMI for Development, Version 1.3, Software Engineering Institute

COHN Mike, 2007, Six Attributes of a Good Scrum Master, http://www.scrumalliance.org/community/articles/2007/february/leader-of-the-band (23.03.2017)

"Crystal Methods", 2017, https://en.wikiversity.org/wiki/Crystal_Methods, (24.03/2017)

"Çevik Modelleme ve Çevik Yazılım Geliştirme", 2017, http://e-bergi.com/y/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme, (09/03/2017)

"Çevik Yazılım Geliştirme Manifestosu", 2017, http://agilemanifesto.org/iso/tr/manifesto.html, (09/03/2017)

"Dynamic System Development Method (DSDM)(a)", 2017, http://www.freetutes.com/systemanalysis/sa2-dynamic-system-development-method.html, (25/03/2017)

"Dynamic Systems Development Method (DSDM)(b)", 2017, http://dsdmofagilemethodology.wikidot.com/, (25/03/2017)

ERDOĞMUŞ Umur, 2009, Süreç İyileştirme CMMI Modelleri ve Türkiye'de CMMI Uygulamarının Durumu Yüksek Lisans Tezi syf:42-50

"Extreme Programming Nedir?", 2017, http://www.kurumsaljava.com/2008/11/21/extreme-programming-nedir/, (22/03/2017)

"Feature-driven development(a)", 2017, https://en.wikipedia.org/wiki/Featuredriven_development, (26/03/2017)

"Feature Driven Development(b)", 2017, http://sliitmscfdd.wikidot.com/introduction-to-fdd, (26/03/2017)

"Feature Driven Development (FDD) and Agile Modeling", 2017, http://agilemodeling.com/essays/fdd.htm, (25/03/2017)

GOYAL Sadhna, 2007, Agile Techniques for Project Management and Software Engineering, syf :6 -9, Technical University Munich Major Seminar On Feature Driven Development

GÜMRÜKÇÜ Gülbahar, 2010, Savunma Sistemlerinde Yazılım Proje Yönetimi Yüksek Lisans Tezi, syf:18-19

HİÇDURMAZ Mümin, 2011, Yazılım Yasam Döngüsü Modeli Seçimi için Bir Bulanık Çok Kriterli Karar Verme Yaklaşımı, Syf:10

HUNDERMARK Peter, 2009, Do Better Scrum, syf:7, http://www.scrumsense.com/wpcontent/uploads/2009/12/DoBetterScrum-v2.pdf, (23/03/2017)

"ISTQB Foundation Level", 2016, http://www.defnesarlioglu.com/istqb-foundation-level-bolum-5/ , (05/12/2016)

"IEEE 12207-2008, 2008": Systems and software engineering—Software life cycle processes,

JOHNSON Brian, HIGGINS John, 2008, Itıl ve Yazılım Yaşam Döngüsü: Pratik Stratejii ve Tasarım Prensipleri kitapı, syf:42-44

KAN Stephen H., 2002, Metrics and Models in Software Quality Engineering 2nd Edition.

KENDAL Kenneth E., 1992, Systems analysis and design. (Vol. 2). Prentice-Hall

KIR Gülname, 2014, Scrum Adoptıon In Kuveyt Turk It Yüksek Lisans Tezi, syf: 6-10

KILINÇ Ayşe, 2010, Süreç İyileştirmede Bütünleşik Yeterlilik Olgunluk Modeli Yüksek Lisans Tezi syf:28

KOÇER Havva Gül, 2007, Xp Ve Uml'nin Yazılım Geliştirme Sürecine Etkileri Yüksek Lisans Tezi, syf: 50-53

KULPA Margaret K., JOHNSON Kent A. 2003, Interpreting the CMMI A Process Improvement Approach, Auerbach Publication, ISBN 0-8493-1654-5, Florida, syf:36-39

MCCONNELL, Steve, 2004, Code Complete 2nd Edition. Programming

NİZAM Ali, 2015, Yazılım Proje Yönetimi Kitap Papatya Yayıncılık Syf:53

OCAK Şeyda, 2012, Güvenli Yazılım Geliştirme Süreç Modelinin Seçimi Yüksek Lisans Tezi syf:54-55

OMACAN Ceren, 2008, Yazılım Şirketlerinde CMMI ve Bir Yazılım Şirketinde CMMI'ın Performansa Etkilerinin İncelenmesi Yüksek Lisans Tezi syf:13

PALMER Stephen R., FELSING John M., 2001, A Practical Guide to Feature-Driven Development, syf: 35-49

"Performance Metrics for Agile SCRUM Process", 2017, https://josephvargheese.wordpress.com/2013/02/17/1000-performance-metrics-for-agile-scrum-process/, (23/03/2017)

PRESSMAN Roger S.,MAXIM Bruce R.,2015, Software Engineering: A Practitioner's Approach syf:74-82

ROSSBERG Joachim, 2008, Pro Visual Studio Team System Application Lifecycle Management

SARAÇOĞLU İbrahim Halil, 2009, CMMI Modeli ve Çevik Yazılım Geliştirme Yöntemlerinin Entegrasyonu Yüksek Lisans Tezi syf:10-27

SCHACH, Stephen R., 1993. Software engineering, Aksen Associates: Irwin, Homewood, IL

"Scrum Roles - The Scrum Team", 2017, http://www.scruminstitute.org/Scrum_Roles_The_Scrum_Team.php, (23/03/2017)

"SDLC-Agile Model", 2017, https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm, (09/03/2017)

"SDLC - Iterative Model", 2016, https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm, (21/12/2016)

"SDLC-Software Prototype Model", 2016, https://www.tutorialspoint.com/sdlc/sdlc_software_prototyping.htm, (22/12/2016)

"SDLC-Spiral Model", 2016, https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm, (22/12/2016)

"SDLC-V-Model", 2016, https://www.tutorialspoint.com/sdlc/sdlc_v_model.htm, (16/12/2016)

SEI CMMI Product Team, 2002, CMMI Version 1.1 (CMMI-SE/SW/IPPD/SS, V1.1), Software Engineering Institute, Carnegie Mellon University

"Software Development Methodologies", 2017, http://www.itinfo.am/eng/software-development-methodologies/, (24/03/2017)

"Software Development Life Cycle", 2016, http://www.ihsany.com/blog/27/software-development-life-cycle.aspx, (21/12/2016)

SOMMERVİLLE Ian, 2004, Software Engineering, Addison-Wesley Publishing Co., 9th Edition, syf:30

SUTHERLAND Jeff, SCHWABER Ken , 2011, The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework, syf: 16-28

SUTHERLAND Jeff, SCHWABER Ken, 2016, Scrum Guide, syf:3-14 http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf#zoom=100, syf:3-14, (23/03/17)

"System Engineering Fundamentals", 2001, Defense Acquisition University, Virgina

ŞEKER Sadi Evren, 2015, Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü, YBS Ansiklopedi 2.cilt Sayı:3 syf:18-20, http://ybsansiklopedi.com/, (01.12.2016)

ŞEKER Sadi Evren, 2014, BT Projelerinde Yaşanan Problemler ve Çözüm Önerileri, YBS Ansiklopedi 1.cilt Sayı:3 syf:23-25, http://ybsansiklopedi.com/, (01.12.2016)

ŞEKER Sadi Evren, 2014, Bilgi Yönetimi (Knowledge Management), YBS Ansiklopedi 1.cilt Sayı:2 syf:10-16, http://ybsansiklopedi.com/, (01.12.2016)

ŞİMSEK Umut, 2011, Cmmı Rehberliğinde Prınce2 ile Servis Odaklı Mimari Tabanlı Uygulama Geliştirme Yüksek Lisans Tezi syf:15-16

TARIKULU Orhan, 2011, CMM'ın Bir Türk Bankacılık Sektöründe Uygulanması ve Analizi Yüksek Lisans Tezi syf:9-16

"The Advantages and Disadvantages of Agile SCRUM Software Development", 2017, http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-agile-scrum-software-development.html, (23/03/2017)

"The Crystal Family of Methodologie", 2017, http://www.devx.com/supportitems/showSupportItem.php?co=32836&supportitem=figure2, (24/03/2017)

"The Dynamic Systems Development Method (DSDM)-Agile Methodology", 2017, https://www.linkedin.com/pulse/dynamic-systems-development-method-dsdm-agile-alaa, (24/03/2017)

TS ISO/IEC 12207/Haziran 2007 syf:1

ULUDAG İbrahim, 2006, A Software Process Assessment Model And A Tool For Xp@Scrum Agıle Method Yüksek lisans tezi syf:11-12

VOIGT Benjamin J. J., 2004, Dynamic System Development Method, syf:1, https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws03/14_Voigt_DSMD_Ausarbeitung.pdf, (24/03/2017)

"What are the advantages and disadvantages of Extreme Programming?", 2017, https://www.quora.com/What-are-the-advantages-and-disadvantages-of-Extreme-Programming, (22/03/2017)

"What Is DSDM?", 2017, https://www.codeproject.com/Articles/5097/What-Is-DSDM, (25/03/2017)

"What is Extreme Programming", 2017, http://ronjeffries.com/xprog/what-is-extreme-programming, (22/03/2017)

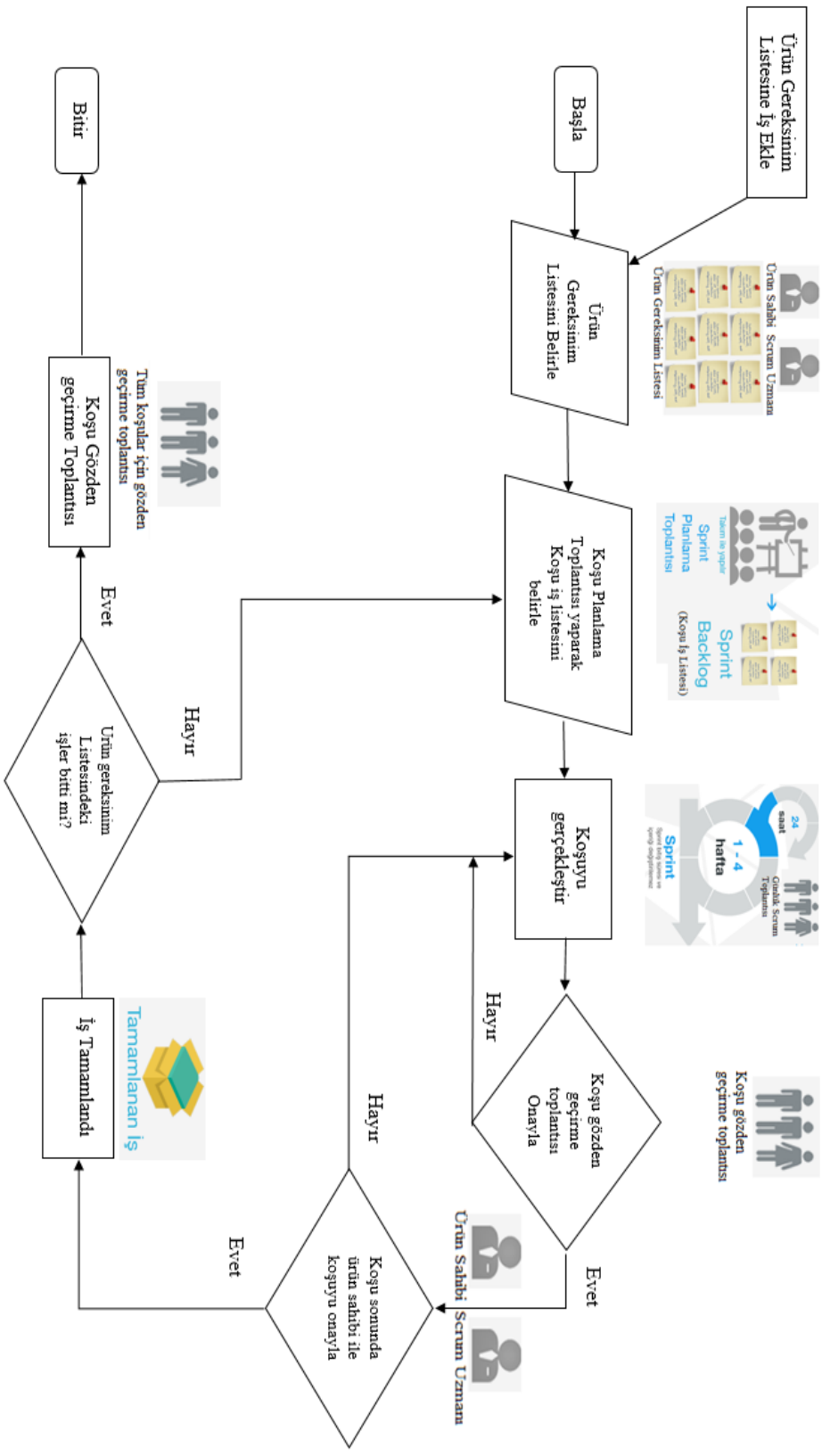
"What is Incremental model- advantages and disadvantages", 2016, http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/, (21/12/2016)

"Yazılım Geliştirme Metodolojilerimiz", 2016, http://www.bimetri.com/urunler/yazilim/ozel/yazilim-gelistirme-metodolojilerimiz/, (01.12. 2016)

"Yazılım Geliştirme Süreç Modelleri", 2016, http://www.mehmetduran.com/Blog/Makale.html/Yazilim-Gelistirme-Surec-Modelleri/299 (21/12/2016)

YÜCALAR Fatih, 2006, Süreç Odaklı Kalite Yönetimi Anlayışına Hakim Yazılım Sektöründeki Firmaların CMMI Basamaklı Modeli İle Değerlendirilmesi Yüksek Lisans Tezi syf:50-58

EK-1: Bakanlık için Scrum Yönteminin Akış Diyagramı



EK-2: Bakanlık için Scrum Yönteminin Uygulanması

Adım1: Scrum Uzmanı ile birlikte Ürün Sahibi Ürün Gereksinim Listesini belirler. Ürün Gereksinim Listesindeki her bir iş için öncelik ve süre belirlenir.

Adım 2: Geliştirme Ekibi Koşu Planlama Toplantısı yaparak her bir iş için koşu iş listesi oluşturulur. Belirlenen işler için süreler verilir ve bu süreler uyulması gerekir. Koşu iş listesi ile çalışan bir yazılım parçacığı oluşturulması hedeflenir. Koşunun süresi 1 hafta ile 4 hafta arasında olmalıdır.

Adım 3: Koşu Gerçekleştirilir. Koşu gerçekleştirilirken günlük koşu hakkında ayaküstü 15 dakikadan fazla sürmeyecek toplantılar yapılır.

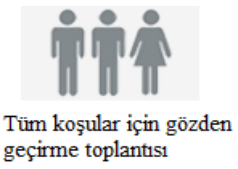
Adım 4: Koşu tamamladıktan sonra koşu gözden geçirme toplantısı yapılır. Koşu sonunda oluşan yazılım parçacığı geliştirme ekibi tarafından kabul edilirse bir sonraki adıma geçilir. Kabul edilmezse 3. Adıma dönülür.

Adım 5: Koşu sonunda çalışan yazılım modülü ürün sahibine gösterilir. Yapılan işin onaylanması işlemi gerçekleştirilir. Yapılan için onay verilirse 6. Adıma geçilir ve koşu kapatılmış olur. Eğer onay verilmezse 3. Adıma geçilir.

Adım 6: Koşu ürün sahibi tarafından kabul edilmişse işlem tamamlanmış olur.

Adım 7: Ürün gereksinim listesindeki işler bitmemişse bir sonraki iş için koşu iş listesi oluşturmak için 2. Adıma geçilir. Ürün gereksinim listesindeki işler bitmişse bir sonraki adıma geçilir.

Adım 8: Biten projede gerçekleştirilen tüm koşular için gözden geçirme toplantısı yapılır ve Proje sonlanır.



EK-3: Örnek Proje için Scrum Yönteminin Uygulanması

Proje Adı: Faaliyet İzleme Sistemi, Bakanlıktaki mevcut uygulamalardan veriler alınarak raporlama sisteminin yapılması

Faaliyet İzleme Sistemi için Scrum Uzmanı ile birlikte Ürün Sahibi Ürün Gereksinim Listesini belirler. Ürün Gereksinim Listesindeki her bir iş için öncelik ve süre belirlenir.

	Ürün gereksinim listesi	Öncelik	Başlangıç Efor Tahmini (saat)	1	2	3	4	5
1	İl Faaliyet Sisteminden Verilerin alınarak raporlanması	1	40	32	24	16	8	0
2	Hava Kalitesi Sisteminden Veriler alınarak raporlanması	2	12	4	0			
	Toplam		**52**	**36**	**24**	**16**	**8**	**0**

Geliştirme Ekibi Koşu Planlama Toplantısı yaparak her bir iş için koşu iş listesi oluşturulur. Belirlenen işler için süreler verilir ve bu süreler uyulması gerekir. Koşu iş listesi ile çalışan bir yazılım parçacığı oluşturulması hedeflenir. Koşunun süresi 1 hafta ile 4 hafta arasında olmalıdır.

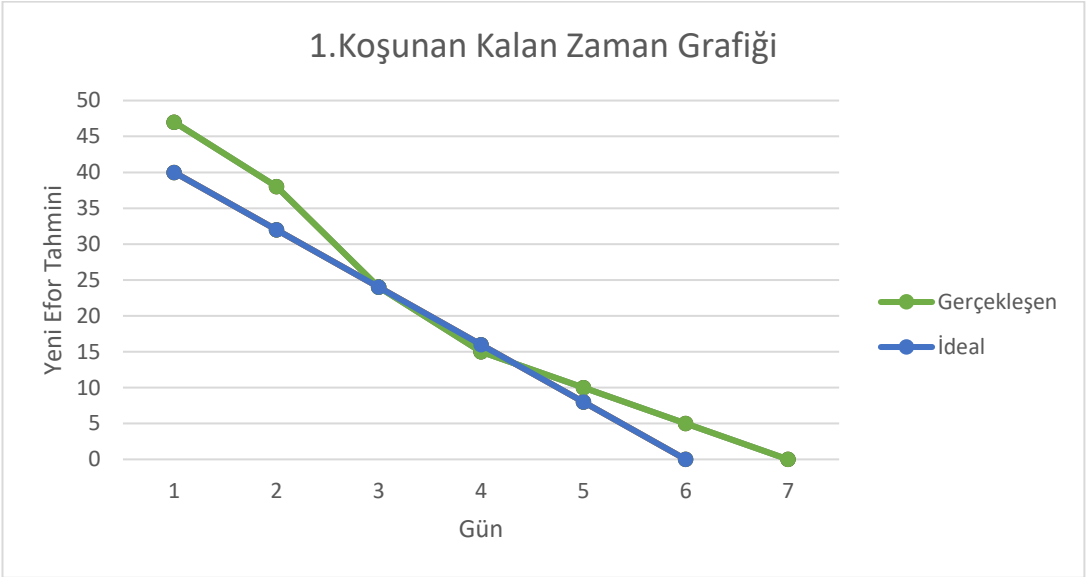
Ürün gereksinim listesi	Koşu İş Listesi	Gönüllü	Başlangıç Efor Tahmini (saat)	1	2	3	4	5	6	7
İl Faaliyet Sisteminden Verilerin alınarak raporlanması	Harita alt yapısının oluşturulması		5	4	2	0				
	Kullanılacak ara yüzün tasarlanması		5	4	0					
	Altyapı Kentsel Dönüşüm Ekranlarının rapora aktarılması (2 Ekran)		2	2	2	1	0			
	Çevre Yönetimi Ekranlarının rapora aktarılması (7 Ekran)		7	7	7	4	2	1	0	
	ÇED Ekranlarının rapora aktarılması (5 Ekran)		5	5	5	3	2	0		
	Mesleki Hizmetlerin Ekranlarının rapora aktarılması (3 ekran)		3	3	3	3	3	2	0	
	Bütün Ekranlarının analizinin yapılması (17 Ekran)		10	5	0					
	Veri Tabanın oluşturulması ve Tabloların belirlenmesi		3	2	0					
	Grafik raporlarının oluşturulması verilerin incelenmesi		3	3	3	2	1	0		
	Grafik türlerinin belirlenmesi		2	1	0					
	Oluşan raporların verilerle uyumlu olduğunun incelenmesi		2	2	2	2	2	2	0	
	Toplam		**47**	**38**	**24**	**15**	**10**	**5**	**0**	

Koşu Gerçekleştirilir. Koşu gerçekleştirilirken günlük koşu hakkında ayaküstü 15 dakikadan fazla sürmeyecek toplantılar yapılır.

Koşu tamamladıktan sonra koşu gözden geçirme toplantısı yapılır. Koşu sonunda oluşan yazılım parçacığı geliştirme ekibi tarafından kabul edilirse bir sonraki adıma geçilir. Kabul edilmezse koşuya tekrar gidilir.

Koşu gözden geçirme toplantısı

Ürün gereksinim listesi	Koşu İş Listesi	Gönüllü	Başlangıç Efor Tahmini (saat)	1	2	3	4	5	6	7
İl Faaliyet Sisteminden Verilerin alınarak raporlanması	Harita alt yapısının oluşturulması	✓	5	4	2	0				
	Kullanılacak arayüzün tasarlanması	✓	5	4	0					
	Altyapı Kensel Dönüşüm Ekranlarının rapora aktarılması (2 Ekran)	✓	2	2	2	1	0			
	Çevre Yönetimi Ekranlarının rapora aktarılması (7 Ekran)	✓	7	7	7	4	2	1	0	
	ÇED Ekranlarının rapora aktarılması (5 Ekran)	✓	5	5	5	3	2	0		
	Mesleki Hizmetlerin Ekranlarının rapora aktarılması (3 ekran)	✓	3	3	3	3	3	2	0	
	Bütün Ekranlarının analizinin yapılması (17 Ekran)	✓	10	5	0					
	Veri Tabanın oluşturulması ve Tabloların belirlenmesi	✓	3	2	0					
	Grafik raporlarının oluşturulması verilerin incelenmesi	✓	3	3	3	2	1	0		
	Grafik türlerinin belirlenmesi	✓	2	1	0					
	Oluşan raporların verilerle uyumlu olduğunun incelenmesi	✓	2	2	2	2	2	2	0	
	Toplam		47	38	24	15	10	5	0	

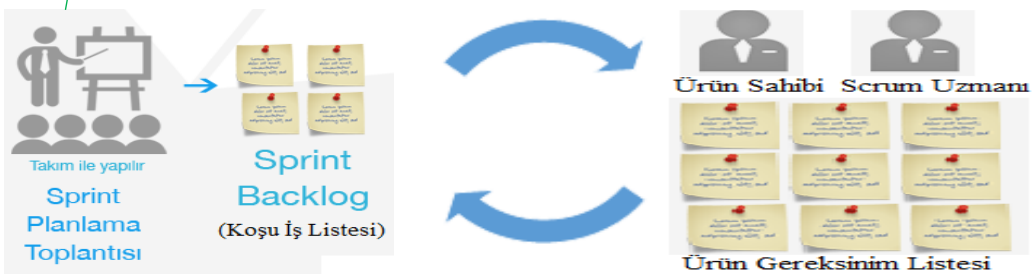


Koşu sonunda çalışan yazılım modülü ürün sahibine gösterilir. Yapılan işin onaylanması işlemi gerçekleştirilir. Yapılan için onay verilirse koşu kapatılmış olur. Eğer onay verilmezse koşuya tekrar dönülür.

Koşu ürün sahibi tarafından kabul edilmişse işlem tamamlanmış olur.

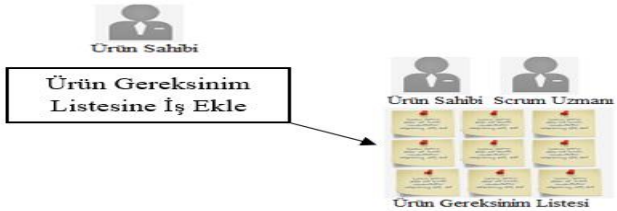
Ürün gereksinim listesindeki işler bitmemişse bir sonraki iş için koşu iş listesi oluşturmak için Koşu Planlama Toplantısı yapılır. Ürün Gereksinim listesindeki işler bitmişse koşuların sonunda koşu gözden geçirme toplantısı yapılır.

	Ürün gereksinim listesi	Öncelik	Başlangıç Efor Tahmini (saat)	1	2	3	4	5
1	~~İl Faaliyet Sisteminden Verilerin alınarak raporlanması~~	1	40	32	24	16	8	0
2	Hava Kalitesi Sisteminden Veriler alınarak raporlanması	2	12	4	0			
	Toplam		**52**	**36**	**24**	**16**	**8**	**0**



Ürün gereksinim listesi	Koşu İş Listesi	Gönüllü	Başlangıç Efor Tahmini (saat)	1	2	3	4	5	6	7
Hava Kalitesi Sisteminden Veriler alınarak raporlanması	Harita alt yapısının oluşturulması		3	2	0					
	Hava Kalitesi uygulamasının incelenmesi		3	1	0					
	Veri Tabanı Tabloların belirlenmesi		2	1	0					
	Veri entegrasyonun sağlanması		5	3	1	0				
	Verilerin gösterim şeklinin belirlenmesi		2	2	0					
	Harita üzerinde oluşan verilerin Hava Kalitesi sistemi ile uyumluluğunun incelenmesi		3	3	2	0				
	Toplam		**18**	**12**	**3**	**0**				

Bu şekilde koşular ürün gereksinim listesindeki tüm işler bitene kadar gerçekleştirilir. Ürün gereksinim listesindeki işler gerçekleştirilirken ürün sahibi herhangi bir zaman da ürün gereksinim listesini güncelleyebilir.



	Ürün gereksinim listesi	Öncelik	Başlangıç Efor Tahmini (saat)	1	2	3	4	5
1	İl Faaliyet Sisteminden Verilerin alınarak raporlanması	1	40	32	24	16	8	0
2	Hava Kalitesi Sisteminden Veriler alınarak raporlanması	2	12	4	0			
3	Çevre izin Lisans Uygulamasından veriler alınarak raporlanması	3	9	1	0			
4	İl Faaliyet Sisteminden Verilerin alınarak raporlanması için iyileştirme çalışmaları	4	10	2	0			
	Toplam		**71**	**39**	**24**	**16**	**8**	**0**

Bu işler için oluşacak koşu iş listesi aşağıdaki gibi olur.

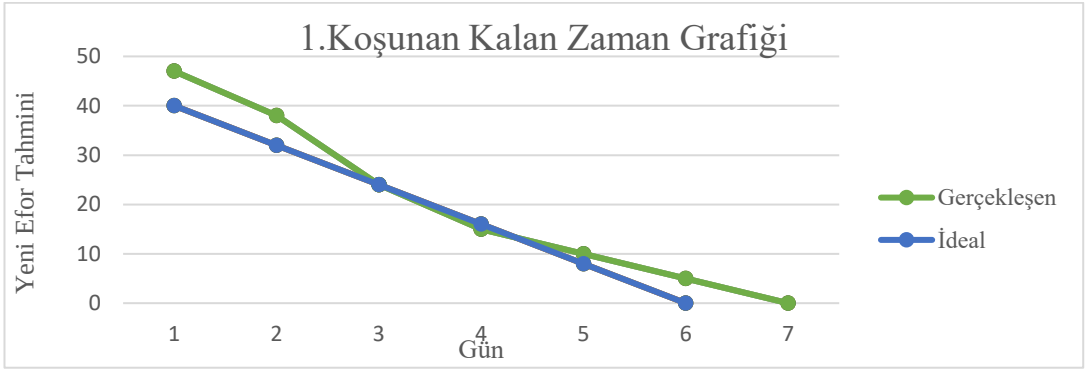
Ürün gereksinim listesi	**Koşu İş Listesi**	**Gönüllü**	**Başlangıç Efor Tahmini (saat)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
İl Faaliyet Sisteminden Verilerin alınarak raporlanması	~~Harita alt yapısının oluşturulması~~		5	4	2	0				
	~~Kullanılacak arayüzün tasarlanması~~		5	4	0					
	~~Altyapı Kensel Dönüşüm Ekranlarının rapora aktarılması (2 Ekran)~~		2	2	2	1	0			
	~~Çevre Yönetimi Ekranlarının rapora aktarılması (7 Ekran)~~		7	7	7	4	2	1	0	
	~~ÇED Ekranlarının rapora aktarılması (5 Ekran)~~		5	5	5	3	2	0		
	~~Mesleki Hizmetlerin Ekranlarının rapora aktarılması (3 ekran)~~		3	3	3	3	3	2	0	
	~~Bütün Ekranlarının analizinin yapılması (17 Ekran)~~		10	5	0					
	~~Veri Tabanın oluşturulması ve Tabloların belirlenmesi~~		3	2	0					
	~~Grafik raporlarının oluşturulması verilerin incelenmesi~~		3	3	3	2	1	0		
	~~Grafik türlerinin belirlenmesi~~		2	1	0					
	~~Oluşan raporların verilerle uyumlu olduğunun incelenmesi~~		2	2	2	2	2	2	0	
	Toplam		**47**	**38**	**24**	**15**	**10**	**5**	**0**	
Hava Kalitesi Sisteminden Veriler alınarak raporlanması	~~Harita alt yapısının oluşturulması~~		3	2	0					
	~~Hava Kalitesi uygulamasının incelenmesi~~		3	1	0					
	~~Veri Tabanı Tabloların belirlenmesi~~		2	1	0					
	~~Veri entegrasyonun sağlanması~~		5	3	1	0				
	~~Verilerin gösterim şeklinin belirlenmesi~~		2	2	0					
	~~Harita üzerinde oluşan verilerin Hava Kalitesi sistemi ile uyumluluğunun incelenmesi~~		3	3	2	0				
	Toplam		**18**	**12**	**3**	**0**				

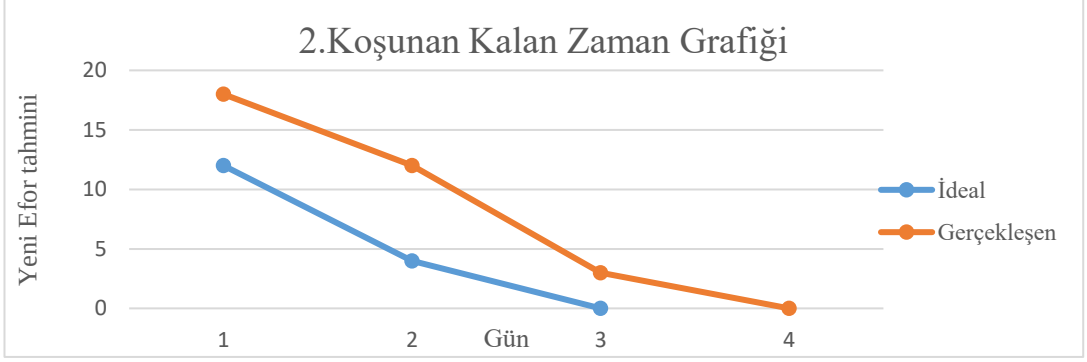
Çevre izin Lisans Uygulamasından veriler alınarak raporlanması	Çevre izin Lisans uygulamasının incelenmesi		3	2	0					
	Veri Tabanı Tablolarının belirlenmesi		3	1	0					
	Veri entegrasyonun sağlanması		2	1	0					
	Verilerin gösterim şeklinin belirlenmesi		5	3	1	0				
	Oluşan raporların verilerle uyumlu olduğunun incelenmesi		2	2	0					
	Toplam		**15**	**9**	**1**	**0**				
İl Faaliyet Sisteminden Verilerin alınarak raporlanması için iyileştirme çalışmaları	Grafik raporlarının iyileştirilme çalışmaları		4	3	2	1	0			
	Tutarsız verilerinin analizinin yapılması		4	2	0					
	Harita altyapısının iyileştirilmesi		4	2	2	0				
	Toplam		**12**	**7**	**4**	**1**	**0**			

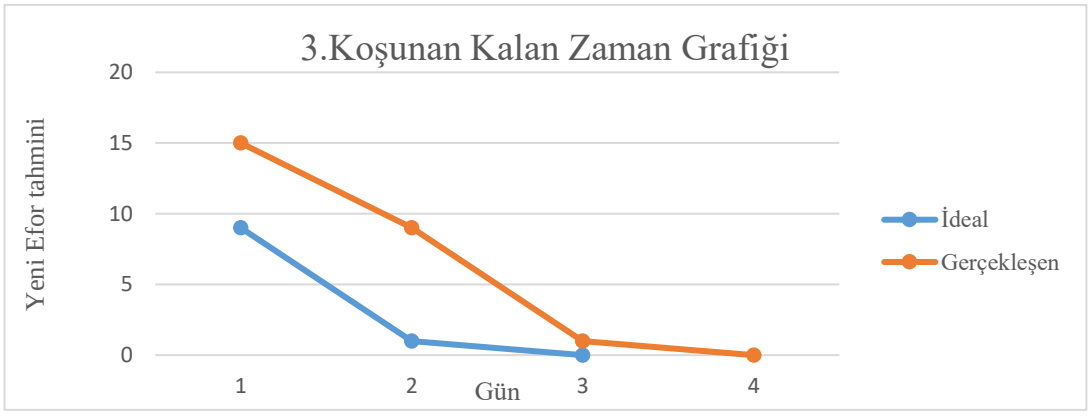
Ürün Gereksinim listesindeki tüm işler için oluşturulan koşu iş listesi bittikten sonra tüm koşular için gözden geçirme toplantısı yapılır.

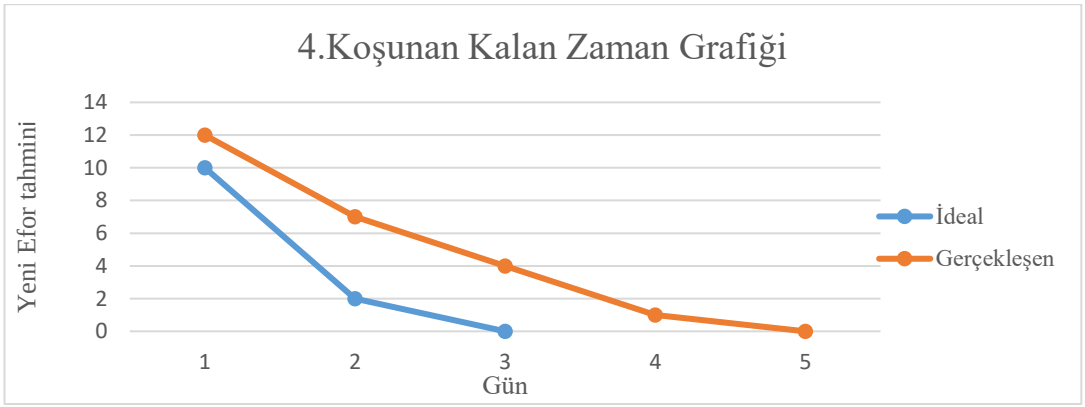
Tüm koşular için gözden geçirme toplantısı

Sonuç olarak aşağıdaki gibi ideal ve gerçekleşen zaman grafikleri oluşur.



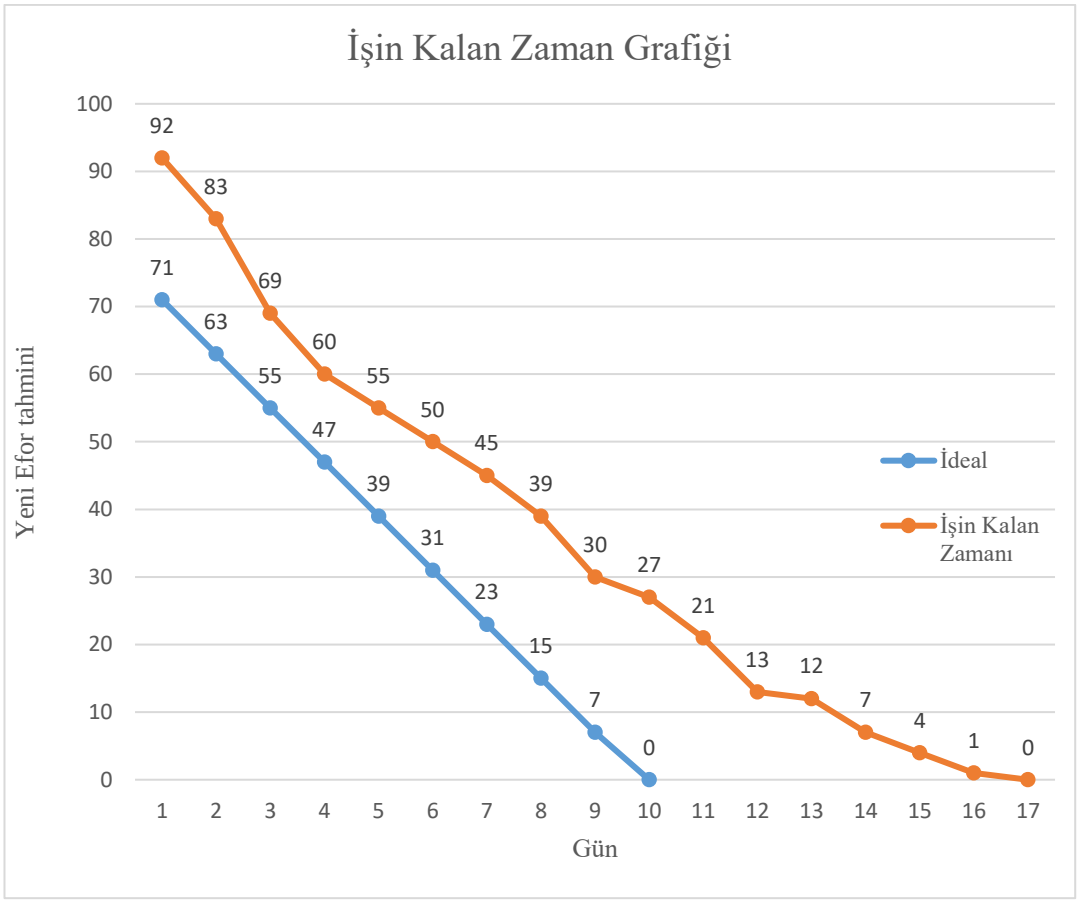






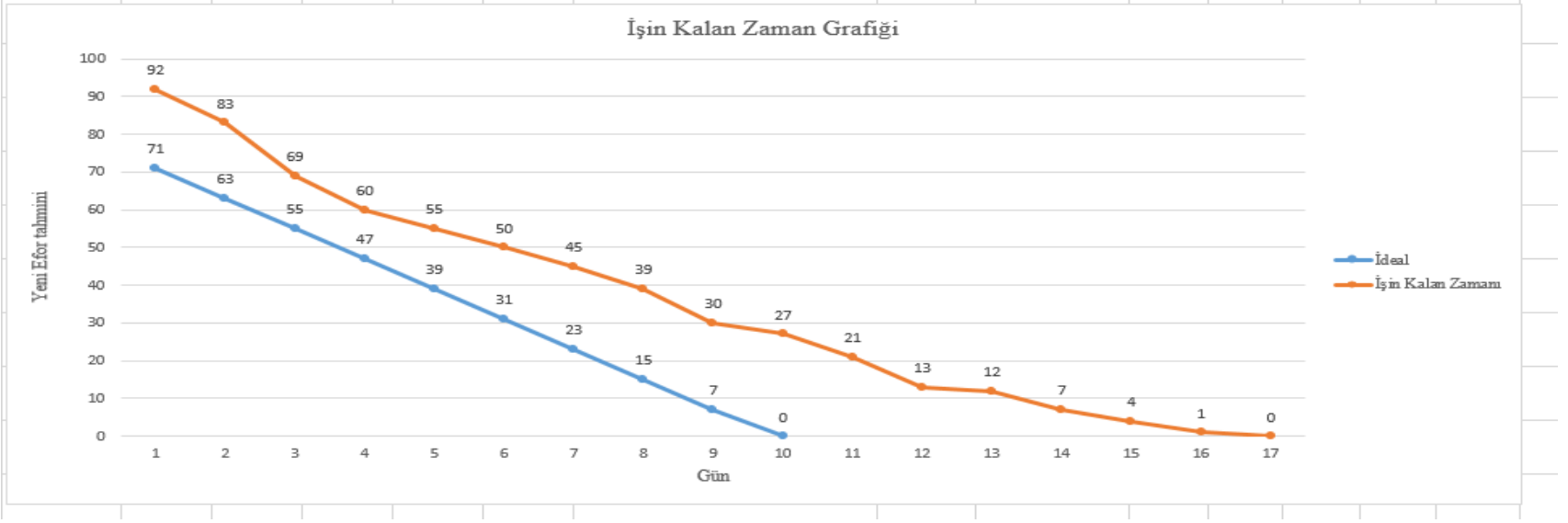
En son olarak ürün gereksinim listesindeki işlerin tümü ve gerçekleşen işlerin zaman grafiği de incelenir.

	Ürün gereksinim listesi	Öncelik	Başlangıç Efor Tahmini (saat)	1	2	3	4	5
1	İl Faaliyet Sisteminden Verilerin alınarak raporlanması	1	40	32	24	16	8	0
2	Hava Kalitesi Sisteminden Veriler alınarak raporlanması	2	12	4	0			
3	Çevre izin Lisans Uygulamasından veriler alınarak raporlanması	3	9	1	0			
4	İl Faaliyet Sisteminden Verilerin alınarak raporlanması için iyileştirme çalışmaları	4	10	2	0			
	Toplam		**71**	**39**	**24**	**16**	**8**	**0**

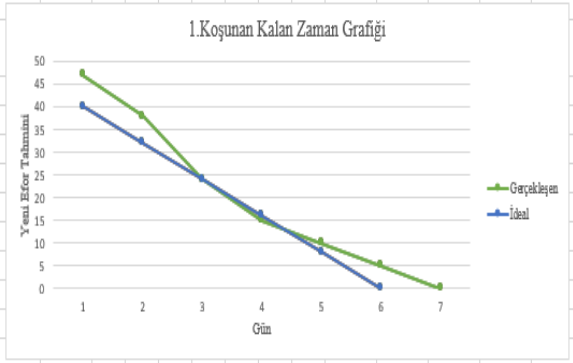

İşin Kalan Zaman Grafiği
Yeni Efor tahmini
Gün
İdeal
İşin Kalan Zamanı
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
92 83 69 60 55 50 45 39 30 27 21 13 12 7 4 1 0
71 63 55 47 39 31 23 15 7 0

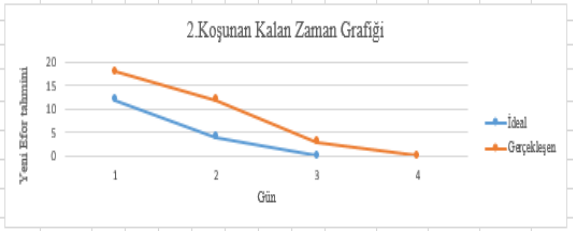
Faaliyet İzleme Sisteminin Çıktıları

	Ürün gereksinim listesi	Öncelik	Başlangıç Efor Tahmini (saat)	1	2	3	4	5
1	İl Faaliyet Sisteminden Verilerin alınarak raporlanması	1	40	32	24	16	8	0
2	Hava Kalitesi Sisteminden Veriler alınarak raporlanması	2	12	4	0			
3	Çevre izin Lisans Uygulamasından veriler alınarak raporlanması	3	9	1	0			
4	İl Faaliyet Sisteminden Verilerin alınarak raporlanması için iyileştirme çalışmaları	4	10	2	0			
	Toplam		**71**	**39**	**24**	**16**	**8**	**0**

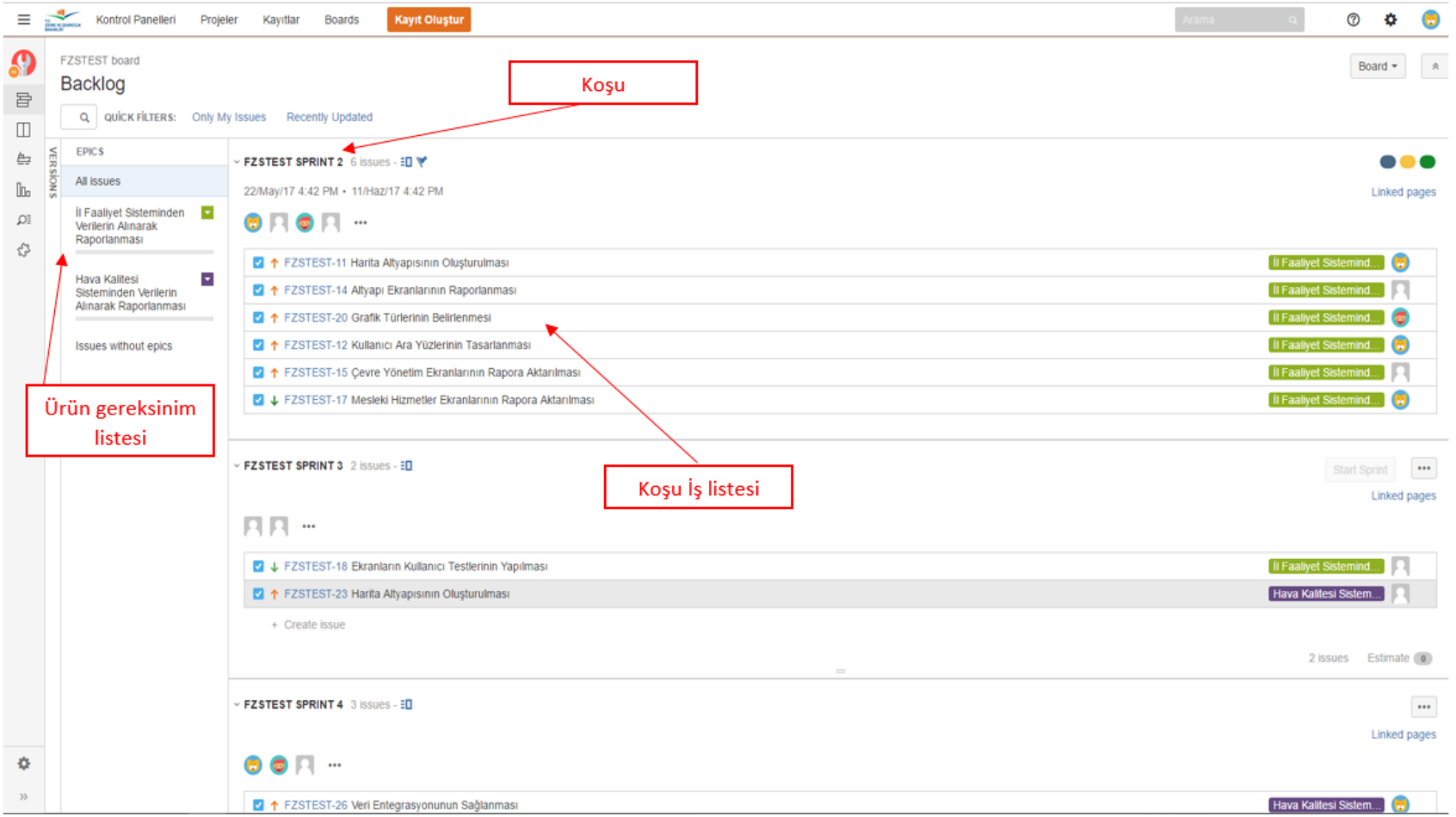


Ürün gereksinim listesi	Koşu İş Listesi	Gönüllü	Başlangıç Efor Tahmini (saat)	... Gün itibariyle Kalan Yeni Efor Tahmini						
				1	2	3	4	5	6	7
İl Faaliyet Sisteminden Verilerin alınarak raporlanması	Harita alt yapısının oluşturulması		5	4	2	0				
	Kullanılacak arayüzün tasarlanması		5	4	0					
	Altyapı Kensel Dönüşüm Ekranlarının rapora aktarılması (2 Ekran)		2	2	2	1	0			
	Çevre Yönetimi Ekranlarının rapora aktarılması (7 Ekran)		7	7	7	4	2	1	0	
	ÇED Ekranlarının rapora aktarılması (5 Ekran)		5	5	5	3	2	0		
	Mesleki Hizmetlerin Ekranlarının rapora aktarılması (3 ekran)		3	3	3	3	3	2	0	
	Bütün Ekranlarının analizinin yapılması (17 Ekran)		10	5	0					
	Veri Tabanın oluşturulması ve Tabloların belirlenmesi		3	2	0					
	Grafik raporlarının oluşturulması verilerin incelenmesi		3	3	3	2	1	0		
	Grafik türlerinin belirlenmesi		2	1	0					
	Oluşan raporların verilerle uyumlu olduğunun incelenmesi		2	2	2	2	2	2	0	
	Toplam		**47**	**38**	**24**	**15**	**10**	**5**	**0**	
Hava Kalitesi Sisteminden Veriler alınarak raporlanması	Harita alt yapısının oluşturulması		3	2	0					
	Hava Kalitesi uygulamasının incelenmesi		3	1	0					
	Veri Tabanı Tabloların belirlenmesi		2	1	0					
	Veri entegrasyonun sağlanması		5	3	1	0				
	Verilerin gösterim şeklinin belirlenmesi		2	2	0					
	Harita üzerinde oluşan verilerin Hava Kalitesi sistemi ile uyumluluğunun incelenmesi		3	3	2	0				
	Toplam		**18**	**12**	**3**	**0**				
Çevre izin Lisans Uygulamasından veriler alınarak raporlanması	Çevre izin Lisans uygulamasının incelenmesi		3	2	0					
	Veri Tabanı Tablolarının belirlenmesi		3	1	0					
	Veri entegrasyonun sağlanması		2	1	0					
	Verilerin gösterim şeklinin belirlenmesi		5	3	1	0				
	Oluşan raporların verilerle uyumlu olduğunun incelenmesi		2	2	0					
	Toplam		**15**	**9**	**1**	**0**				
İl Faaliyet Sisteminden Verilerin alınarak raporlanması için iyileştirme çalışmaları	Grafik raporlarının iyileştirilme çalışmaları		4	3	2	1	0			
	Tutarsız verilerinin analizinin yapılması		4	2	0					
	Harita altyapısının iyileştirilmesi		4	2	2	0				
	Toplam		**12**	**7**	**4**	**1**	**0**			





EK-4: Örnek Projenin Jira Programı Üzerinde Takip Edilmesi



ÖZGEÇMİŞ

1985 yılında Trabzonda doğdu. Lise öğrenimini İstanbul Mehmet Niyazi Altuğ Lisesi'nde(Y.D.A) tamamladı. 2009 yılında Elazığ Fırat Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nden mezun oldu. 2015 yılında Afyon Kocatepe Üniversitesi Fen Bilimleri Enstitüsü İnternet ve Bilişim Teknolojileri Yönetimi yüksek lisans(Tezli) eğitimine başlamış olup halen devam etmektedir. 2010 yılında Uyumsoft Bilgi Sistemleri ve Teknolojileri AŞ'de yaklaşık bir yıl süreyle yazılım geliştirici olarak görev yaptı. 2010-2011 yılları arasında Sinerji Bilişim firmasında yaklaşık bir yıl süreyle yazılım geliştirici olarak görev yaptı. 2012-2013 yılları arasında Türkiye Demiryolu Makinaları Sanayi AŞ(TÜDEMSAŞ) Bilgi işlem şubesinde mühendis olarak çalıştı. 2013-2014 yılları arasında İstanbul Üniversitesi Akademik Geliştirme Kalite Geliştirme Kurulu Biriminde mühendis olarak görev yaptı. 2014 yılında Çevre ve Şehircilik Bakanlığı'nda Çevre ve Şehircilik Uzman Yardımcısı olarak göreve başladı.

ETİK KURALLARA UYGUNLUK BEYANI

Uzmanlık tezi olarak sunduğum bu çalışmayı, bilimsel ahlak ve geleneklere aykırı düşecek bir yol ve yardıma başvurmaksızın yazdığımı, yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu, bunlardan her seferinde değinme yaparak yararlandığımı ve Çevre ve Şehircilik Uzmanlığı Yönetmeliğine uygun olarak hazırladığımı belirtir, bunu onurumla doğrularım. Çevre ve Şehircilik Bakanlığı tarafından belli bir zamana bağlı olmaksızın, tezimle ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçlara katlanacağımı bildiririm.

29.05.2017

Salih SOYLU