T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

CMMI MODELİ VE ÇEVİK YAZILIM GELİŞTİRME YÖNTEMLERİNİN ENTEGRASYONU

Tezi Hazırlayan
İbrahim Halil SARAÇOĞLU

Tezi Yöneten
Doç.Dr.Erkan BEŞDOK

Bilgisayar Mühendisliği Ana Bilim Dalı
Yüksek Lisans Tezi

Eylül 2009
KAYSERİ

T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

CMMI MODELİ VE ÇEVİK YAZILIM GELİŞTİRME YÖNTEMLERİNİN ENTEGRASYONU

Tezi Hazırlayan
İbrahim Halil SARAÇOĞLU

Tezi Yöneten
Doç.Dr.Erkan BEŞDOK

Bilgisayar Mühendisliği Ana Bilim Dalı
Yüksek Lisans Tezi

Eylül 2009
KAYSERİ

Doç.Dr.Erkan BEŞDOK danışmanlığında **İbrahim Halil SARAÇOĞLU** tarafından hazırlanan **"CMMI Modeli ve Çevik Yazılım Geliştirme Yöntemlerinin Entegrasyonu"** adlı bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında **Yüksek Lisans** tezi olarak kabul edilmiştir.

31.08.2009

JÜRİ:

Başkan : Doç.Dr.Erkan BEŞDOK

Üye : Prof.Dr.Derviş KARABOĞA

Üye : Doç.Dr.Coşkun ÖZKAN

ONAY:

Bu tezin kabulü, Enstitü Yönetim Kurulunun 29/09/2009 tarih ve 2009/33-11... sayılı kararı ile onaylanmıştır.

29/09/2009

Prof. Dr. Nusret AYYILDIZ
Enstitü Müdürü

TEŞEKKÜR

“CMMI ve Çevik Süreç Entegrasyonu” konulu tez çalışmamın gerçekleşmesinde, sonuçlandırılmasında ve sonuçlarının değerlendirilmesinde maddi ve manevi destek ve yardımlarını esirgemeyen değerli hocam sayın Doç. Dr. Erkan BEŞDOK’a çok teşekkür ederim.

Tez çalışması boyunca benim için uygun şartları sağlayan ve tez sürecinin benim açımdan daha az stresli geçmesi için elinden geleni yapan, emeklerini hiçbir şekilde ödeyemeyeceğim annem Dilber SARAÇOĞLU’na teşekkür ederim.

CMMI MODELİ VE ÇEVİK YAZILIM GELİŞTİRME YÖNTEMLERİNİN ENTEGRASYONU

İbrahim Halil SARAÇOĞLU
Erciyes Üniversitesi, Fen Bilimleri Enstitiisii
Yiiksek Lisans Tezi, Eylül 2009
Tez Danışmanı: Doç. Dr. Erkan BEŞDOK

ÖZET

Günümüzde yazılım endüstrisi, yazılım geliştirme süreçleri ve süreç değerlendirme modelleri giderek önem kazanıyor. Bu kapsamda geliştirilen süreç değerlendirme standartlarından CMMI, yazılım geliştirme süreç modellerinden ise Çevik geliştirme modeli giderek yaygın olarak kullanılıyor.

Carnegie Mellon Üniversitesi'nin Software Engineering Institute akademisyenleri tarafından geliştirilmiş bir standart olan CMMI yazılım projelerinde "ne"lerin yapılmasını söyleyerek projelerin başarı oranını yükseltecek bir olgunluk ve iyileştirme modeli sunuyor. Çevik geliştirme ise süreçlerin "nasıl" gerçekleştirilmesi konusunda pratikler ihtiva ederek, hızlı ve müşteri memnuniyeti yüksek yazılım oluşturulmasını sağlıyor.

Bu tez çalışmasında, birbirinden çok farklı olduğu düşünülen CMMI ve Çevik Süreçlerin birlikte kullanılabilirliği ispatlanmaya çalışıldı. Aralarındaki benzerlik ve farklar irdelendi. Benzer amaçları yerine getiren pratikler ortaya çıkarıldı. Az sayıdaki farklılıkların da giderilmesi için de çözümler sunuldu.

Anahtar Kelimeler: CMMI; Süreç değerlendirme; Süreç İyileştirme; Agile; Çevik Geliştirme; Yazılım Geliştirme; Çevik Yöntemler; Scrum; XP.

CMMI MODEL AND AGILE SOFTWARE DEVELOPMENT METHODS INTEGRATION

İbrahim Halil SARAÇOĞLU
Erciyes University, Graduate School of Natural and Applied Sciences
M.Sc. Thesis, September 2009
Thesis Supervisor: AP Dr. Erkan BEŞDOK

ABSTRACT

Nowadays, software industry, software development process and process assesment are becoming more important. CMMI is one of the process assesment Standard *that tells* you "what to do". Agile is also one of the software development models.

CMMI is developed as *maturity and optimization model* by Carnegie Mellon University Software Engineering Institute academicians to improve software project success. But Agile is a collection of the methods *tells* "how to do" about software development. Agile gives some practices for rapid delivery of high-quality software which satisfies customer needs.

In this work, I tried to present both approach are not different or in conflict. Many supporting sides and a few differences have been found by comparing the CMMI requirements with Agile practices. Some alternative solutions offered to minimize differences between CMMI and Agile.

Keywords: CMMI; Process assesment; Process optimization; Agile; Agile Development; Software Development; Agile Methods; Scrum; XP.

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

1.BÖLÜM

GİRİŞ

Bilişim, hayatımızın her alanında karşımıza çıkan, dev bir endüstri halini almıştır. Üstelik bu endüstri, kapsamı her geçen gün büyüyerek gelişiyor. Yazılım da bu gelişen endüstrinin en hızlı gelişen alt alanlarından biri olduğundan, yazılım ve buna bağlı süreçler bilişim endüstrisi için hayati önem taşır hale geldi.

Ülkeler her yıl binlerce gereksinim duydukları yazılım projesini hayata geçiriyor. Fakat doğru proje süreç yönetimi ve çeşitli yazılım geliştirme pratiklerden yoksun geliştirilen yazılım projelerinin çoğu büyük oranda başarısızlıkla sonuçlanıyor. The Standish Group'un düzenli aralıklarla yayınladığı "The Chaos Report" adlı araştırma bu konuda çarpıcı sonuçlar sunmaktadır. The Chaos Report'un 2004 yılı rakamlarına göre ABD'de farklı sektörlerde farklı ölçeklerden firmalarda gerçekleştirilen 30,000 yazılım projesinin sadece %34'ü başarılı; %15'i başladıktan sonra iptal edilmiş; %51'i ise tartışmalı, yani zamanında bitirilememiş veya gereksinimleri karşılamamış projeler. Yazılım ve inşaat sektörleri arasında bir analoji kurarsak, yazılım sektöründe yapılan binaların %15'i yıkılıyor gibi bir sonuca ulaşıyoruz. Ayrıca ABD Ulusal Standartlar ve Teknoloji Enstitüsü'nün raporuna göre yazılım hataları ABD ekonomisine yılda 60 milyar dolara mal oluyor. Bu rakam bazı yerlerde 75 milyar dolar olarak geçiyor. Yani ABD yazılım sektöründe başarısız projeler nedeniyle Türkiye'nin 2008 yılı sonu itibariyle hesaplanan dış borcunun dörtte birinden bile fazla bir maddi zarar oluşuyor. Gartner Enstitüsü'nün 1999–2002 yılları arasında yaptığı kapsamlı araştırmaya göre bilişim teknolojisi projelerinin %74'ü başarısız ya da maliyet/zaman hedeflerini aşıyor. %51'i bütçesini %200 oranında aşarken, hedeflenen özelliklerin ancak %75'ini karşılayabiliyor.

Başarılı olduğu düşünülen bazı yazılım projelerinin ise sonradan hataları ortaya çıkıyor. Bu hataların bazıları telafi edilemez zararlara yol açabiliyor.

ABD'nin Denver kentinde inşa edilen, dünyanın en büyük uluslararası havaalanının otomatik bagaj sisteminin 230 milyon dolarlık bir yazılımla yönetilerek 1994 yılında açılması planlanıyordu. Ancak bagaj sistemindeki yazılım hataları nedeniyle havaalanının açılması 16 ay gecikti. Bu gecikmenin günlük maliyetinin 1 milyon dolara yakın olduğu ve gecikme nedeniyle oluşan toplam zararın 340 milyon doları bulduğu hesaplanıyor. Nihayetinde 70 milyon dolarlık yedek bir proje devreye sokuldu. O zamandan beri çeşitli sorunlarla çalıştırılan bu yazılımın da 2005 yılında artık iş göremeyeceği belirlenerek yenilenme kararı alındı.

Avrupa Uzay Ajansı ve Havacılık Dairesi(NASA) 1996 yılında uydu taşıma amaçlı 7 milyar Euro'luk Arianne II isimli bir roket geliştirdi. Roket fırlatıldıktan 37 saniye sonra kontrol yazılımındaki bir hata nedeniyle infilak etti ve bu olay en pahalı yazılım hatalarından biri olarak tarihe geçti. Problem 16 bit 64 bit çevrimi sırasında oluşan bir "overflow" hatasından kaynaklanıyordu ve aslında bilgisayar programcılığında öğrenilen temel konulardan biriydi.

Ünlü spor malzemeleri üretim şirketi Nike, i2 isimli tedarik zinciri yönetim yazılımını kullandığı 400 milyon dolarlık tedarik zinciri yönetim sisteminin 2001 yılında yazılım sorunları nedeniyle iş göremez hale geldiğini duyurdu. Nike 2001 yılında 3. çeyrekte yaşadığı yazılım sorunları nedeniyle beklediği satışın 100 milyon dolar altında kaldı ve hisseleri %21 düştü.

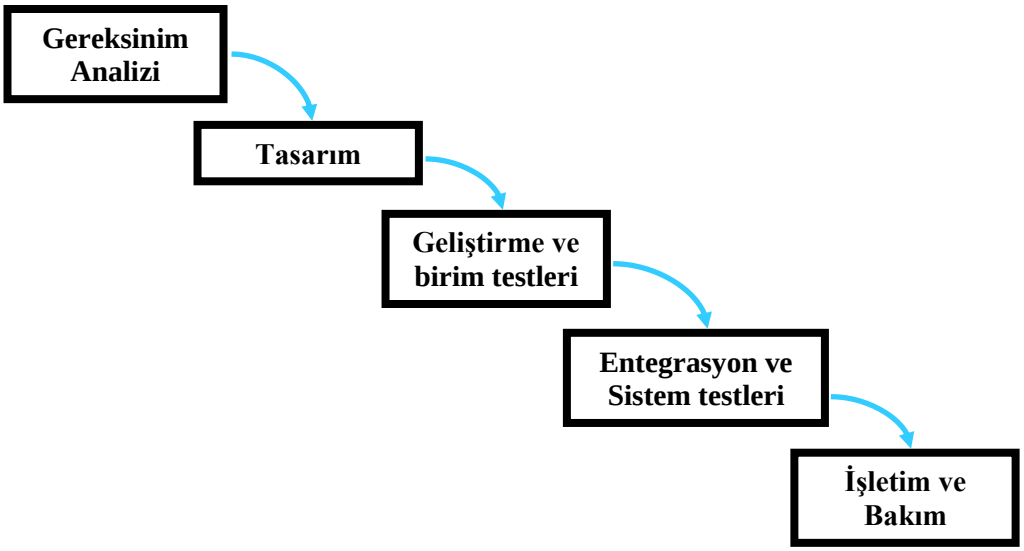
14 Ağustos 2003'te, ABD'nin 8 eyaletinde ve Kanada'da 50 milyon kişinin elektriksiz kalmasına neden olan bir elektrik kesintisi gerçekleşti. Kesintinin nedeni elektrik şirketinin Ohio bölgesinde elektrik tellerine değerek arızaya neden olan ağaçları budamayı ihmal etmesi olmakla birlikte, erken önlem alınamamasının temel nedeninin, şirketin kontrol sisteminin alarm vermesini önleyen yazılım hataları olduğu belirlendi. Ancak 16 Ağustos'ta tamamen giderilebilen elektrik kesintisi iki ülkenin ekonomisine yaklaşık 6 milyar dolara mal oldu ve büyük güvenlik riskleri oluştu.

Yazılım projelerindeki ya da oluşan yazılımlardaki bu gibi başarısızlıklar nedeniyle oluşan zararlar, diğer endüstriyel alanlardaki süreç standartlaştırma ve iyileştirme ihtiyacını yazılım alanında da gerekli kıldı. Uzun yıllar boyunca yazılım geliştirme gibi karmaşık bir süreci disipline etmek ve belirli bir sistematiğe oturtmak amacıyla değişik yazılım mühendisliği metotları ve pratikleri ortaya konuldu ve hala konuluyor. Yazılımın üç önemli temel faktörü olan süreç, teknoloji ve insanı dikkate alarak "kim", "ne", "nasıl" sorularına cevap aranarak kaliteli, maliyeti düşük ve müşteri ihtiyaçlarını karşılayan başarılı yazılım elde etmenin yolları aranıyor. Bu çalışmaların ürünü olarak bugün hala günümüzde kullanılan bazı metotlar, modeller ve standartlar var.

Bu süreci modellerinin en önemli ve yaygın olarak kullanılanları şunlardır.

- Şelale (Waterfall)
- Evrimsel Geliştirme (Evolutionary Development)
- Yinelemeli ve Artımlı (Iteratif & Incremental)

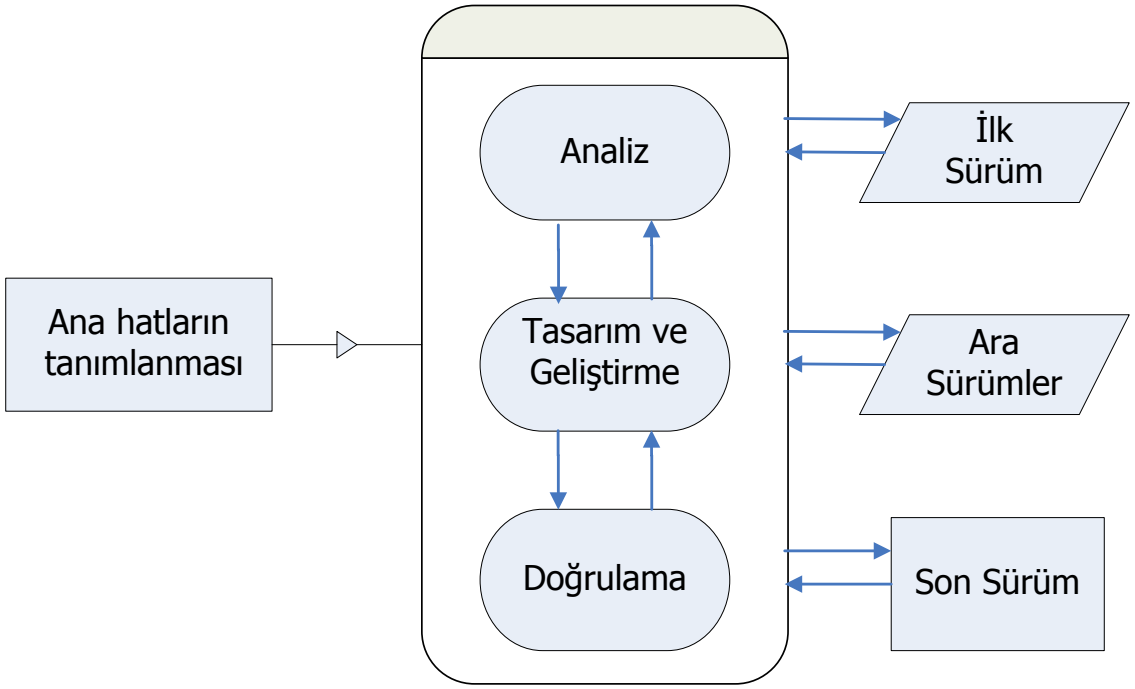
Şelale modeli, yazılımın belirli bir geliştirme yöntemi olmadan baştan sona kodlanıp test edilerek oluşturulmasının büyük projeler için işe yaramamasına çözüm olarak ortaya çıkmıştır. "Şelale modeli" terimi ilk kez Winston Royce tarafından 1970 yılında kullanılmıştır. Bu model bir projedeki hiçbir sürecin bir önceki süreç tamamlanmadan başlayamayacağı fikrine dayanır. Proje süreçleri (planlama, gereksinim geliştirme, tasarım, kodlama ve doğrulama) bir sıra halinde bir önceki süreç bittikten sonra başlar ve bir sonraki süreç başlamadan önce biter. Bir süreç bir önceki süreç bitmeden başlayamaz.



Şekil 1.1 - Şelale Modeli

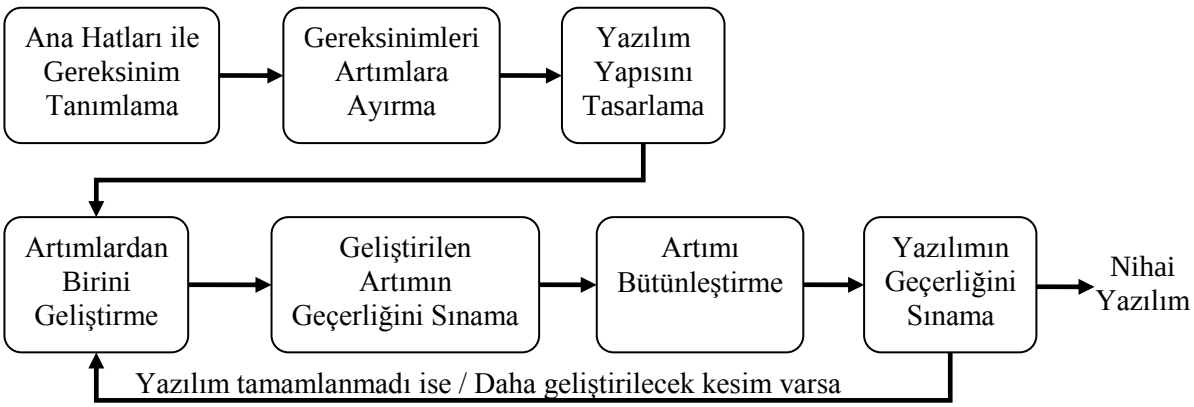
Modelin sıralı bir akış takip etmesi, modelin proje açısından yönetimini kolaylaştırır. Aynı sebepten ötürü, yazılımın standartlara uygunluğunun kontrolü de kolaydır Bütün bu avantajların yanında şelale modelinin çok önemli bir dezavantajı vardır. Projenin geç safhalarında gelen değişiklikler çok pahalıya mal olur. Şelale modeli sıralı bir akış izlediğinden, doğrulama safhasında gereksinimlerde oluşan bir değişiklik akış içersindeki tüm süreçlerden geçmek zorundadır (tasarım, kodlama ve doğrulama). Bu yüzden gereksinimlerdeki bu değişiklik çok pahalıya mal olur.

Evrimsel geliştirme ise tanımlama, analiz, tasarım, geliştirme ve doğrulama faaliyetlerinin ilk olarak hızlı bir şekilde yapılması, doğrulama sağlanana kadar aşamaların tekrarlanması mantığını temel alan gelişmiş bir modeldir. Şelale modeline göre biraz daha karmaşık bir yönetim mekanizması gerektirse de, değişiklikleri çok daha az maliyetle karşılayabilmesi güçlü bir özelliğidir.



Şekil 1.2 - Evrimsel Geliştirme

Yinelemeli ve artımlı metodu ise şelale modelindeki eksiliklerden yola çıkılarak geliştirilmiştir ve yazılımın geliştirilmesi sırasında bir tekrar (döngü) ile yazılımın daha iyi hale getirilmesi hedeflenmiştir. Yazılım kalitesini arttıran, erken geribildirim sayesinde riskin azaltılmasını sağlayan, yeni ve değişen gereksinimleri gerçekleştirmek için esnek bir yapı sunan bir metottur. Yazılımı küçük adımlarla geliştirilir. Daha sonra sürüm serileri geliştirilerek ürün son haline getirilir.



Şekil 1.3 - Yinelemeli ve Artımlı

Günümüzde giderek fazla bir kullanım alanı bulan Çevik Geliştirme (Agile Development) modeli bu metodu temel alarak geliştirilmiştir.

Bunların dışında; bileşen tabanlı yazılım mühendisliği, biçimsel yazılım geliştirme modeli (formal), yeniden kullanıma dayalı geliştirme modeli (reuse-oriented) gibi yazılım süreci modelleri çeşitlendirilebilir.

Üstte saydığımız modeller, yazılımın geliştirilmesinin "nasıl" yapılması gerektiği sorusuna cevap veriyordu. Ama aynı yazılım geliştirme sürecinde kalitenin devamlı sağlanması için daha çok "ne" yapılması gerektiği sorusuna cevap veren bazı standartlar da geliştirilmiştir. ISO 9000–3, AQAP 150/160, TICKIT, TRILLIUM, ISO 12207, ISO 15504 (SPICE), BOOTSTRAP, CMM, CMMI, MIL-STD 498 bunlardan bazılarıdır.

Tüm bu süreç yönetim/iyileştirme metot ve standartları, yazılımda kalite kontrol ve yönetim işinin öneminin bazı ülkeler tarafından anlaşıldığı 1960'lardan günümüze kadar birçok yazılım projesinde uygulandı ve kullanıldı. Böylece geliştirilen yazılımların başarı oranları arttırıldı.

Yaptığım tez çalışmasının amacı, bu standartlardan biri olan CMMI modeli ile süreç yönetim modellerinden biri olan Çevik geliştirmenin bazı uyarlamalar yapılarak birlikte kullanılabilirliğini göstermek ve bu birlikteliğin yazılım geliştirme sürecine sağladığı faydaya dikkat çekmektir. Pek çok akademisyen ve bilim adamının katkısı ile geliştirilen CMMI modeli ve konusunda yıllarca deneyimli yazılım uzmanları tarafından geliştirilen Çevik yazılım geliştirme yöntemleri ile ilgili pek çok ayrı çalışma yapılmış ve genel itibari ile iki yöntem arasındaki farklılıkların, avantajların ve dezavantajların neler olduğu kişisel tecrübelerle saptanmıştır. Bu iki yöntemin bir arada kullanılması dünyada giderek artan bir eğilim gösterirken, her iki kavramında henüz yeni olduğu Türkiye'de bu konuda çok fazla çalışma yapılmamıştır. Bu tez çalışması ile Türkiye gibi yazılım sektörü hızla gelişen ve bu alanda dünyada önemli bir yer kapmak isteyen ülkemizde yazılım üreten şirket yöneticilerinin ve çalışanlarının bu yöntemlerin birlikteliğinde elde edecekleri kazançları daha iyi görebilmeleri hedeflenmektedir.

Tezin kapsamına bakacak olursak; çalışmanın 2.bölümünde CMMI süreç ve seviyeleri hakkında detaylı bilgi verilecek, 3.bölümde ise Çevik yöntemler ve pratikleri ayrıntılı şekilde incelenecektir. 4.bölümde Scrum, 5.bölümde XP çevik yöntemleri detaylandırılacak ve 6.bölümde bu iki çevik yöntemin birlikte kullanımı ele alınacaktır. 7.bölümde CMMI ve Çevik Geliştirme arasındaki benzerlik ve farklılıklar irdelenecek, 8.bölümde bu yapıların pratikleri arasında eşlemeler yapılarak birbirine uyumu tartışılacak. Son bölümde ise bu birlikteliğin sağlayacağı sonuçlar ve faydalar anlatılacaktır.

2.BÖLÜM

CMMI

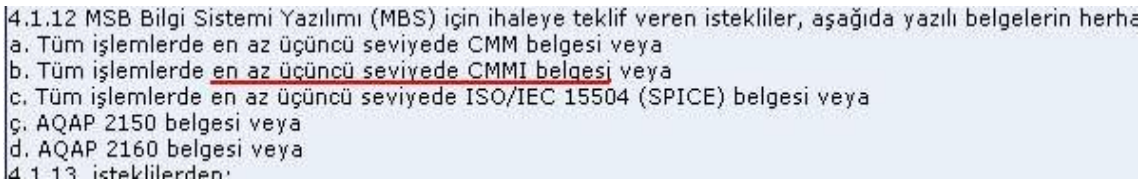
2.1. CMMI Nedir?

CMMI, Türkçe karşılığı "Bütünleşik Yetenek Olgunluk Modeli" anlamına gelen, İngilizce "Capability Maturity Model Integration" kelimelerinin baş harflerinden oluşmuş bir kısaltmadır.

CMMI, bir süreç modeli olup, kurumların yazılım süreçlerinin (yazılım planlama, geliştirme, yapılandırma vb.) olgunluğunu değerlendirmelerini sağlar ve bu süreçlerin iyileştirmesi konusunda gereken temel adımları gösterir. Bir veya birden çok disiplin için etkili süreçlerin gerekli elemanlarını içerir. Geçici, olgunlaşmamış süreçleri; gelişmiş kalite ve yararlılık ile disiplinli ve olgun süreçler haline getirmek için evrimsel gelişim yolu tanımlar. Süreç alanları bazında nelerin eksik olduğunu ve nelerin olması gerektiğini anlatır. Model, sorunlara ait çözümlerin kurumlara; hatta projelere ait olduğunu düşünerek, eksikliklerin nasıl giderileceğine dair yöntemler tanımlamaz. Yani ne yapılması gerektiğini açıklar, nasıl sorusuna kesin bir cevap vermez. Bu yaklaşım bir projeye, bir kuruma ait birime ya da büyük bir organizasyona, süreçlerini iyileştirme ve geliştirme konusunda rehberlik eder. Olgun olmayan bir süreçten olgun ve disiplinli bir sürece giden evrimsel bir yol çizer. Ortaya konulan seviyelere göre olgunluk gelişimi sağlanır. Her olgunluk modelinin kendine özel süreçleri bulunur. Böylece sürekli bir iyileşme söz konusu olur. CMMI, geleneksel yapıda ayrı olan kurumsal fonksiyonların bütünleştirilmesine, süreç iyileştirme için gerekli hedef ve önceliklerinin belirlenmesine, kalite süreçleri için kılavuz oluşturulmasına ve mevcut süreçlerin değerlendirilmesi için bir referans noktası oluşturulmasına yardım eder.

CMMI, dünyada ve Türkiye'de daha çok IT sektöründe ve özellikle yazılım geliştirme yapılan kurum, departman ve projelerde, yazılım kalitesini arttırmak ve süreç

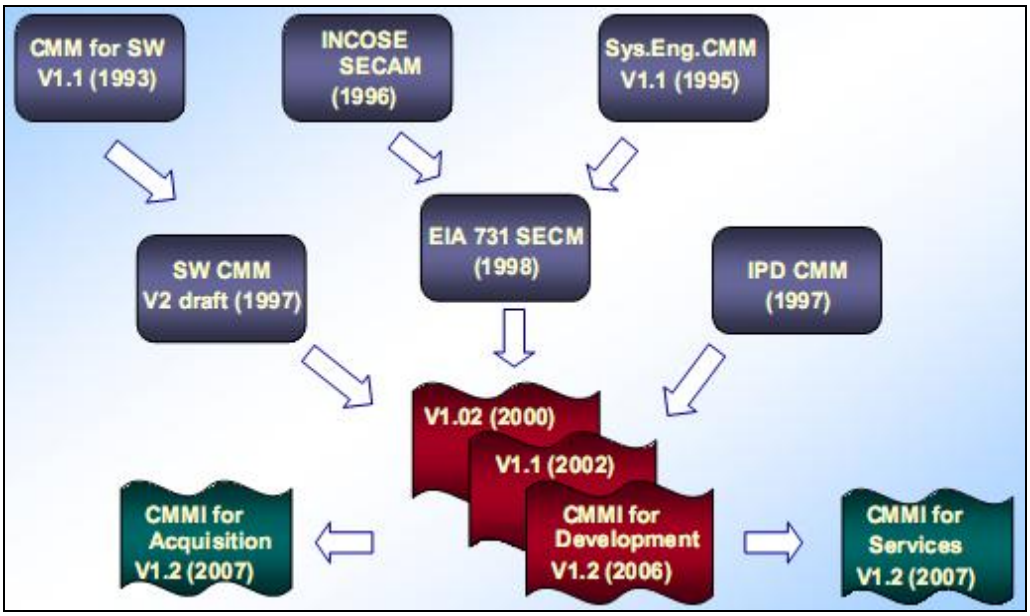
iyileştirmek için kullanılan bir referans model olarak uygulanmaktadır. Donanım ve network alanlarında da uygulanan model, özellikle savunma sanayi başta olmak üzere; Ar-Ge ve yeni ürün geliştirme konusunda hizmet veren kurumların aradıkları bir standarttır. Ayrıca yazılım işlerini yapmak üzere mukavele yapılacak nitelikli/ehliyetli şirketlerin belirlenmesinde de kullanılmaktadır.


4.1.12 MSB Bilgi Sistemi Yazılımı (MBS) için ihaleye teklif veren istekliler, aşağıda yazılı belgelerin herha
a. Tüm işlemlerde en az üçüncü seviyede CMM belgesi veya
b. Tüm işlemlerde en az üçüncü seviyede CMMI belgesi veya
c. Tüm işlemlerde en az üçüncü seviyede ISO/IEC 15504 (SPICE) belgesi veya
ç. AQAP 2150 belgesi veya
d. AQAP 2160 belgesi veya
4.1.13. isteklilerden:

Şekil 2.1 - CMMI belgesi isteyen ihale ilanı örneği

2.2. CMMI Gelişimi

1970'lerde Amerikan Savunma Bakanlığı çok sayıda yazılım ihalesi açıyor. Bu ihalelerin çoğunun geç tamamlanması ya da hiç tamamlanamaması ve maliyet veya süre tahminlerinin tutmaması üzerine Amerikan Savunma Bakanlığı (Department of Defense, DoD) Carnegie Mellon Üniversitesi'nden, az sayıda olan başarılı firmaları analiz ederek, ihtiyaç duyduğu yazılım standartlarını karşılayabilmek için diğer firmaların da kullanabileceği bir model oluşturmasını ister. Böylece Carnegie Mellon Üniversitesi'ne bağlı Yazılım Mühendisliği Enstitüsü (SEI), CMM süreç iyileştirme yaklaşımını geliştirir. İlk olarak yalnızca yazılım üzerine, SW-CMM olarak 1991 yılında yayınlanan model, 2002 yılında sektör bağımsız bir şekil alarak "CMMI" olmuş, 2006 yılında ise v1.2 olan son sürümüne ulaşmıştır. CMMI, açık bir standart/model olmayıp tüm hakları SEI'e aittir.



Şekil 2.2 - CMMI'ın tarihi gelişimi

CMMI v1.2 ile beraber, CMMI-SW, CMMI-SE, CMMI-SS ve CMMI-IPPD modelleri kaldırılmış ve onların yerine üç yeni oluşum getirilmiştir. Bu yeni oluşumlar şunlardır:

- **CMMI-DEV** (Development – Ürün ya da hizmet oluşturma)
  CMMI-DEV, hizmet ya da ürün geliştiren kuruluşlar için tasarlanmıştır. Projeler gerçekleştiren, bu projelerinin sonunda yeni bir ürün ya da hizmet oluşturan kuruluşlar, bu modelden faydalanabilirler.

- **CMMI-SVC** (Services – Hizmet verme)
  CMMI-SVC, geliştirilmesi tamamlanmış, müşteriye teslim edilmiş bir hizmetin ya da ürünün, yürütülmesi, bakımının yapılması ya da işletilmesi ile ilgili hizmetler için kullanılmaktadır.

- **CMMI-ACQ** (Acquisition - Satın alma)
  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.

2.3. 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(goals) tanımlamaktadır. Bu tanımlamayı yaparken hedefleri özel (specific goals) ve genel (generic goals) 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[14]. Özel hedeflerin her biri ise belirli bir süreç alanına karşılık gelir. Yani o süreç alanının gereğidirler. Örneğin "Proje Planlama" süreç alanına ait "Proje Planını Geliştir" hedefi özel bir hedeftir. CMMI her bir hedefe ulaşılması için bir takım pratikler atar. Bu hedef için atanan pratiklere "Bütçe ve takvimi belirle", "Proje risklerini belirle", "Proje kaynaklarını planla" örnek verilebilir. CMMI'ın resmi yayınlarında bu özel pratiklerin, bilinen ve başarısı ispatlanmış uygulamalar olduğu ifade edilmektedir. Ayrıca CMMI, hedeflere ulaşıldığı sürece genel ve özel pratiklerin dışında alternatif pratiklerin de uygulanmasına müsaade etmektedir.

Aşağıda CMMI alanları ve her bir alan için geçerli olan özel hedefler ve bu hedefleri sağlayan özel pratikler sıralanmıştır[5,9].
(SA: Süreç Alanı, ÖH: Özel Hedef, ÖP: Özel Pratik)

SA1. Gereksinim Yönetimi

Bu süreç alanının amacı, proje ürünleri ve ürün bilişenleri ile ilgili gereksinimleri yönetmek ve bu gereksinimler ile proje ürünleri ve ürün bileşenleri arasındaki tutarsızlıkları tespit etmektir.

ÖH1. Gereksinimlerin Yönetilmesi

- ÖP1.1. Gereksinimlerin anlaşılmasını sağla
- ÖP1.2. Gereksinimler için taahhütleri al
- ÖP1.3. Gereksinimlerdeki değişiklikleri yönet
- ÖP1.4. Gereksinimler üzerinde çift yönlü izlenebilirliğini sağla
- ÖP1.5. Gereksinimler ile iş ürünleri arasındaki tutarsızlıkları belirle

SA2. Proje Planlama

Proje Planlamanın amacı, proje faaliyetlerini tanımlayan planları oluşturmak ve güncel tutmaktır.

ÖH1. Tahminlerin Oluşturulması

- ÖP1.1. Proje kapsamının belirle
- ÖP1.2. İş ürünleri ve görev özelliklerini tahminle
- ÖP1.3. Proje hayat döngüsü tanımla
- ÖP1.4. İşgücü ve maliyet tahminle

ÖH2. Proje planının geliştirilmesi

- ÖP2.1. Bütçe ve takvimi belirle
- ÖP2.2. Proje risklerini belirle
- ÖP2.3. Veri yöntemini planla
- ÖP2.4. Proje kaynaklarını planla
- ÖP2.5. Gerekli bilgi ve yetenekleri sapta
- ÖP2.6. Paydaşların katılımını planla
- ÖP2.7. Proje planını oluştur

ÖH3. Plana olan taahhütlerin sağlanması

- ÖP3.1. Proje planlarını gözden geçir
- ÖP3.2. İş ve kaynak seviyelerini uzlaştır
- ÖP3.3. Proje plan taahhütlerini sağla

SA3. Proje İzleme ve Kontrol

Proje izleme ve kontrolün amacı, projenin gidişatı hakkında sürekli bilgi toplanması ve böylece projenin performansı planlardan önemli derecede saparsa bu durumun bir an önce fark edilmesi ve düzeltici hareketlerin yapılabilmesidir.

ÖH1. Plana göre projenin izlenmesi

- ÖP1.1. Proje planlama parametrelerini izle
- ÖP1.2. Taahhütlerin izle
- ÖP1.3. Proje risklerini izle
- ÖP1.4. Veri yönetimini izle
- ÖP1.5. Proje paydaşlarının katılımlarını izle
- ÖP1.6. Gelişme gözden geçirme toplantıları yap
- ÖP1.7. Kilometretaşı gözden geçirme toplantıları yap

ÖH2. Düzeltici faaliyetlerin kapanıncaya kadar yönetilmesi

- ÖP2.1. Sorunların analizi
- ÖP2.2. Düzeltici faaliyetleri gerçekleştir
- ÖP2.3. Düzeltici faaliyetleri yönet

SA4. Tedarikçi Sözleşme Yönetimi

Tedarikçi sözleşme yönetiminin amacı, tedarikçiden ürün satın işlemini taraflar arasında imzalanmış bir sözleşme ile yönetmektir.

ÖH1. Tedarikçi Sözleşmesinin oluşturulması

- ÖP1.1. Satın alma yöntemini belirle
- ÖP1.2. Tedarikçi seç
- ÖP1.3. Tedarikçi Sözleşmesini oluştur

ÖH2. Tedarikçi Sözleşmesinin gereklerinin yerine getirilmesi

- ÖP2.1. Tedarikçi sözleşmesini işlet
- ÖP2.2. Belirlenen tedarikçi süreçlerini izle
- ÖP2.3. Belirlenen tedarikçi iş ürünlerini değerlendir
- ÖP2.4. Satın alınan ürünü kabul et
- ÖP2.5. Ürünün kullanıma geçişini gerçekleştir

SA5. Konfigürasyon Yönetimi

Konfigürasyon yönetiminin amacı, konfigürasyon tanımlama, konfigürasyon kontrol, konfigürasyon durum değerlendirme ve konfigürasyon denetleme faaliyetlerini kullanarak iş ürünlerinin tutarlılığını ve bütünlüğünü oluşturmak ve korumaktır.

ÖH1. Dayanakların oluşturulması

- ÖP1.1. Konfigürasyon birimlerini belirle
- ÖP1.2. Konfigürasyon Yönetim Sistemini oluştur
- ÖP1.3. Dayanakları oluştur ve yayınla

ÖH2. Değişikliklerin izlenmesi ve kontrol edilmesi

- ÖP2.1. Değişiklik isteklerini takip et
- ÖP2.2. Konfigürasyon birimlerini kontrol et

ÖH3. Bütünlüğün sağlanması

- ÖP3.1. Konfigürasyon yönetim kayıtlarını oluştur
- ÖP3.2. Konfigürasyon denetimlerini oluştur

SA6. Süreç ve Ürün Kalite Güvencesi

Süreç ve ürün kalite güvencesinin amacı, çalışanlara ve yöneticilere, süreçler ve ilgili iş ürünleri ile ilgili, tarafsız gözlemler sunmaktır.

ÖH1. Süreçlerin ve iş ürünlerinin tarafsız olarak değerlendirilmesi

- ÖP1.1. Süreçlerin tarafsız olarak değerlendir
- ÖP1.2. İş ürünleri ve hizmetleri tarafsız olarak değerlendir

ÖH2. Tarafsız görüş sağlanması

- ÖP2.1. Uygunsuzlukları ilet ve çözümlendiğinden emin ol
- ÖP2.2. Kayıtları oluştur

SA7. Ölçme ve Analiz

Ölçme ve analizin amacı, yönetimin bilgi ihtiyacına destek olmak için uygun ölçme yeteneklerini geliştirmek ve devamlılığını sağlamaktır.

ÖH1.Ölçme ve analiz çalışmalarının uyumlu hale getirilmesi

- ÖP1.1. Ölçme hedeflerini belirle
- ÖP1.2. Ölçümleri belirle
- ÖP1.3. Veri toplama ve saklama prosedürlerini belirle
- ÖP1.4. Analiz prosedürlerini belirle

ÖH2. Ölçme sonuçlarının sağlanması

- ÖP2.1. Ölçme verilerini topla
- ÖP2.2. Ölçme verilerini analiz et
- ÖP2.3. Veri ve analiz sonuçlarının sakla
- ÖP2.4. Sonuçların ilet

SA8. Gereksinim Geliştirme

Gereksinim geliştirmenin amacı müşteri, ürün ve ürün bileşenlerinin gereksinimlerini oluşturmak ve analiz etmektir.

ÖH1. Müşteri gereksinimlerinin geliştirilmesi

- ÖP1.1. İhtiyaçları ortaya çıkar
- ÖP1.2. Müşteri gereksinimlerini geliştir

ÖH2. Ürün gereksinimlerinin geliştirilmesi

- ÖP2.1. Ürün ve ürün bileşenlerinin gereksinimlerini oluştur
- ÖP2.2. Ürün bileşen gereksinimlerini ata
- ÖP2.3. Arayüz gereksinimlerini belirle

ÖH3. Gereksinimlerin analiz edilmesi ve onaylanması

- ÖP3.1. Kullanıcı senaryoları ve kavramlarını oluştur
- ÖP3.2. Gereksinim duyulan fonksiyonalite tanımını oluştur
- ÖP3.3. Gereksinimleri analiz et
- ÖP3.4. Dengenin sağlanması için gereksinimleri analiz et
- ÖP3.5. Gereksinimleri onayla

SA9. Teknik Çözüm

Teknik çözümün amacı, müşteri ihtiyaçlarına uygun çözümler tasarlamak, oluşturmak ve üretmektir. Çözümler, ürünleri, ürün bileşenleri ve ürün hayat döngüsü ile ilgili süreçlerin tamamını kapsar.

ÖH1. Ürün bileşen çözümlerinin seçilmesi

- ÖP1.1. Alternatif çözümleri ve seçim kriterlerini geliştir
- ÖP1.2. Ürün bileşen çözümlerini seç

ÖH2. Tasarımın Geliştirilmesi

- ÖP2.1. Ürün ve ürün bileşenini tasarla
- ÖP2.2. Teknik veri paketi oluştur
- ÖP2.3. Kriterlere uygun olarak arayüzleri tasarla
- ÖP2.4. Geliştirme, Satın alma ya da tekrar kullanma analizlerini yap

ÖH3. Ürün tasarımının gerçekleştirilmesi

- ÖP3.1. Tasarımı gerçekleştir
- ÖP3.2. Ürün destek dokümantasyonunu geliştir

SA10. Ürün Bütünleştirme

Amacı, ürün bileşenlerini bir araya getirerek ürünü oluşturmak, oluşan ürünün ihtiyaçlarına uygun olarak çalıştığını görmek ve ürünü teslim etmektir.

ÖH1. Ürün bütünleştirme için hazırlıkların yapılması

- ÖP1.1. Bütünleştirme sırasının belirle
- ÖP1.2. Ürün birleştirme ortamını oluştur
- ÖP1.3. Ürün birleştirme prosedürlerini ve kriterlerini geliştir

ÖH2: Arayüz uyumluluğundan emin olma

- ÖP2.1. Arayüz tanımlarının eksiksiz olduğunun gözden geçir
- ÖP2.2. Arayüzleri yönet

ÖH3. Ürün bileşenlerinin bir araya getirilmesi ve ürünün teslim edilmesi

- ÖP3.1. Ürün bileşenlerinin bütünleştirme için hazır olduğunu kontrol et
- ÖP3.2. Ürün bileşenlerinin birleştir
- ÖP3.3. Birleştirilen ürün bileşenleri değerlendir
- ÖP3.4. Ürün ve ürün bileşenini paketle ve teslim et

SA11. Doğrulama

Doğrulamanın amacı, ürün ya da ürün bileşenlerinin hedeflenen ortama yerleştirildiğinde hedeflenen kullanımını sağladığının gösterilmesidir

ÖH1. Doğrulama için hazırlıkların yapılması

- ÖP1.1. Doğrulanacak iş ürünlerinin seçilmesi
- ÖP1.2. Doğrulama ortamlarının oluşturulması
- ÖP1.3. Doğrulama prosedürlerinin ve kriterlerinin oluşturulması

ÖH2. Eşdeğer gözden geçirmelerin gerçekleştirilmesi

- ÖP2.1 Gözden geçirme için hazırlan
- ÖP2.2. Gözden geçirme yap
- ÖP2.2. Gözden geçirme verilerini analiz et

ÖH3. Seçilen iş ürünlerinin doğrulanması

- ÖP3.1 Doğrulamayı gerçekleştir
- ÖP3.2. Doğrulama sonuçlarını analiz et

SA12. Geçerleme

Geçerlemenin amacı, seçilen iş ürünlerinin belirlenen isterleri karşıladığından emin olmaktır.

ÖH1. Geçerleme için hazırlık yapılması

- ÖP1.1. Geçerlenecek ürünleri seç
- ÖP1.2. Geçerleme ortamını oluştur
- ÖP1.3. Geçerleme prosedürlerini ve kriterlerini oluştur

ÖH2. Ürün veya ürün bileşenlerinin geçerlenmesi

- ÖP2.1. Geçerlemeyi gerçekleştir
- ÖP2.2. Geçerleme sonuçlarını analiz et

SA13. Risk Yönetimi

Olası problemler oluşmadan tespit ederek, bu problemlerin proje hedeflerine ulaşılmasına engel olmasını engellemek için, önlemlerin planlanması ve hayata geçirilmesidir.

ÖH1. Risk yönetimi için hazırlıkların yapılması

- ÖP1.1. Risk kaynak ve kategorilerini belirle
- ÖP1.2. Risk parametrelerini tanımla
- ÖP1.3. Risk yönetim stratejisini oluştur

ÖH2. Risklerin tespit edilmesi ve analizi

- ÖP2.1. Risklerin tespit et

- ÖP2.2. Risklerin değerlendir, kategorize et ve önceliklendir

ÖH3. Risk olasılıklarının azaltılması

- ÖP3.1. Risk olasılıklarının azaltma planlarını geliştir
- ÖP3.2. Risk olasılık azaltma planlarını uygula

SA14. Bütünleşik Proje Yönetimi

Projenin ve ilgililerin kurumun standart süreçlerinden özelleştirilmiş tanımlı ve bütünleşik bir süreç ile yönetilmesidir.

ÖH1. Projenin tanımlı sürecinin kullanılması

- ÖP1.1. Projenin tanımlı sürecini oluştur
- ÖP1.2. Proje aktiviteleri planlaması için kurumsal süreç varlıklarını kullan
- ÖP1.3. Projenin çalışma ortamını oluştur
- ÖP1.4. Planların bütünleştirilmesi
- ÖP1.5. Projenin bütünleşik planlar ile yönet
- ÖP1.6. Kurumsal süreç varlıklarına katkıda bulun

ÖH2. İlgili paydaşlar ile işbirliği ve koordinasyon

- ÖP2.1. Paydaşların katılımını yönet
- ÖP2.2. Bağımlılıkları yönet

SA15. Kurumsal Eğitim

Çalışanların rollerini etkin ve verimli bir şekilde gerçekleştirebilmeleri için sahip olmaları gereken yetkinliklerin ve bilginin çalışanlar üzerinde oluşturulmasını amaçlar.

ÖH1. Kurumsal eğitim yeteneğinin oluşturulması

- ÖP1.1. Stratejik eğitim ihtiyaçlarını oluştur
- ÖP1.2. Kurumun sorumluluğunda olan eğitim ihtiyaçlarını belirle

- ÖP1.3. Kurumsal eğitim taktik planını oluştur
- ÖP1.4. Eğitim yeteneğini oluştur

ÖH2. Gerekli eğitimlerinin sağlanması

- ÖP2.1. Eğitimi sağla
- ÖP2.2. Eğitim kayıtlarını oluştur
- ÖP2.3. Eğitimin etkinliğini değerlendir

SA16. Kurumsal Süreç Tanımlama

Amacı, kurumsal süreç varlıklarının kolay kullanıma uygun olarak oluşturulması ve güncel tutulmasıdır.

ÖH1. Kurumsal süreç varlıklarının oluşması

- ÖP1.1. Standart süreçleri oluştur
- ÖP1.2. Yaşam döngüsü model tanımlarını oluştur
- ÖP1.3. Uyarlama kriterlerini ve ana esaslarını oluştur
- ÖP1.4. Kurumsal ölçme deposunu oluştur
- ÖP1.5. Kurumsal süreç varlık kütüphanesini oluştur
- ÖP1.6. Çalışma ortamı standartlarını oluştur

SA17. Kurumsal Süreç Odaklılık

Amacı, kurumun süreç ve süreç varlıklarında hali hazırda bilinen güçlü ve zayıf yönlerine uygun olarak gerçekleştirilmesi gereken süreç iyileştirme çalışmalarını planlamak ve gerçekleştirmektir.

ÖH1. Süreç iyileştirme fırsatlarının belirlenmesi

- ÖP1.1. Kurumsal süreç ihtiyaçlarını belirle
- ÖP1.2. Kurumsal süreçlerini değerlendir
- ÖP1.3. Kurumun süreç iyileştirmelerini belirle

ÖH2. Süreç iyileştirme aktivitelerinin planlanması ve gerçekleştirilmesi

- ÖP2.1. Süreç hareket planlarını oluştur
- ÖP2.2. Süreç hareket planlarını gerçekleştir

ÖH3. Kurumsal süreç varlıklarını yaygınlaştır ve alınan dersleri ekle

- ÖP3.1. Kurumsal süreç varlıklarını yaygınlaştır
- ÖP3.2. Standart süreçleri yaygınlaştır
- ÖP3.3. Uygulamayı izle
- ÖP3.4. Süreçler ile ilgili tecrübeleri kurumsal süreç varlıkları arasına yerleştir

SA18. Karar Analiz ve Çözümleme

Olası kararların, belirlenmiş kriterlere göre değerlendirilmesi ve resmi bir değerlendirme süreci kullanılarak, analiz edilmesidir.

ÖH1. Alternatiflerin değerlendirilmesi

- ÖP1.1. Karar analiz için ana esasları belirle
- ÖP1.2. Değerlendirme kriterlerini belirle
- ÖP1.3. Alternatif çözümleri oluştur
- ÖP1.4. Değerlendirme yöntemlerini seç
- ÖP1.5. Alternatifleri değerlendir
- ÖP1.6. Çözümü seç

SA19. Nicel Proje Yönetimi

Belirlenmiş kalite ve süreç performans hedeflerine ulaşması için, projenin tanımlanmış süreçlerinin sayısal olarak yönetilmesidir.

ÖH1. Projenin nicel olarak yönetilmesi

- ÖP1.1. Projenin hedeflerini oluştur
- ÖP1.2. Tanımlı süreci birleştir
- ÖP1.3. İstatistiksel olarak yönetilecek alt süreci seç

- ÖP1.3. Proje performansını yönet

ÖH2. Alt süreçlerin performansının istatistiksel olarak yönetilmesi

- ÖP2.1. Ölçme ve analitik tekniklerini seç
- ÖP2.2. Değişkenliği anlamak için istatiksel yöntemlerin kullanılması
- ÖP2.3. Seçilen alt süreçlerin performansının gözetlenmesi
- ÖP2.4. İstatiksel yönetim verilerin kayıt altına alınması

SA20. Kurumsal Süreç Performansı

Süreç performans ve kalite hedeflerine desteklemek için, kurumun standart süreçleri hakkında veriler toplamak ve toplanan verileri güncel tutmak. Projenin sayısal yöntemi için ihtiyaç duyulacak olan süreç performans verilerini, temel sürümleri ve modellerini sağlamaktır.

ÖH1. Performans dayanakları ve modellerinin oluşturulması

- ÖP1.1. Süreçleri seç
- ÖP1.2. Süreç performans ölçümlerinin oluştur
- ÖP1.3. Kalite ve süreç performans hedeflerini belirle
- ÖP1.4. Süreç performans dayanaklarını oluştur
- ÖP1.5. Süreç performans modellerini oluştur

SA21. Kurumsal Yenilik ve Yayma

Kurumsal süreçler ve teknolojide ölçülebilir iyileştirme sağlamak için yaratıcı iyileştirme önerilerinin seçilmesi ve seçilen önerilerin kurum geneline yaygınlaştırılmasıdır. İyileştirmeler, şirketin iş hedeflerine uygun olarak oluşturulmuş olan kurumsal kalite ve süreç performans hedeflerine ulaşmalarında destek olmaktadır.

ÖH1. İyileştirmelerin seçilmesi

- ÖP1.1. İyileştirme önerilerini topla ve analiz et
- ÖP1.2. Yenilikleri belirle ve analiz et

- ÖP1.3. Pilot iyileştirme çalışmaları yap
- ÖP1.4. Yaygınlaştırılacak iyileştirmeleri seç

ÖH2. İyileştirmelerin yaygınlaştırılması

- ÖP2.1. Yaygınlaştırılmayı planla
- ÖP2.2. Yaygınlaştırılmayı yönet
- ÖP2.3. İyileştirmenin etkilerini ölç

SA22. Nedensel Analiz ve Çözüm

Hataların ve problemlerin sebeplerini tespit ederek gelecekte tekrar oluşmalarını engellemek için gerekli işlemlerin yapılmasını sağlamaktır.

ÖH1. Hataların sebeplerinin saptanması

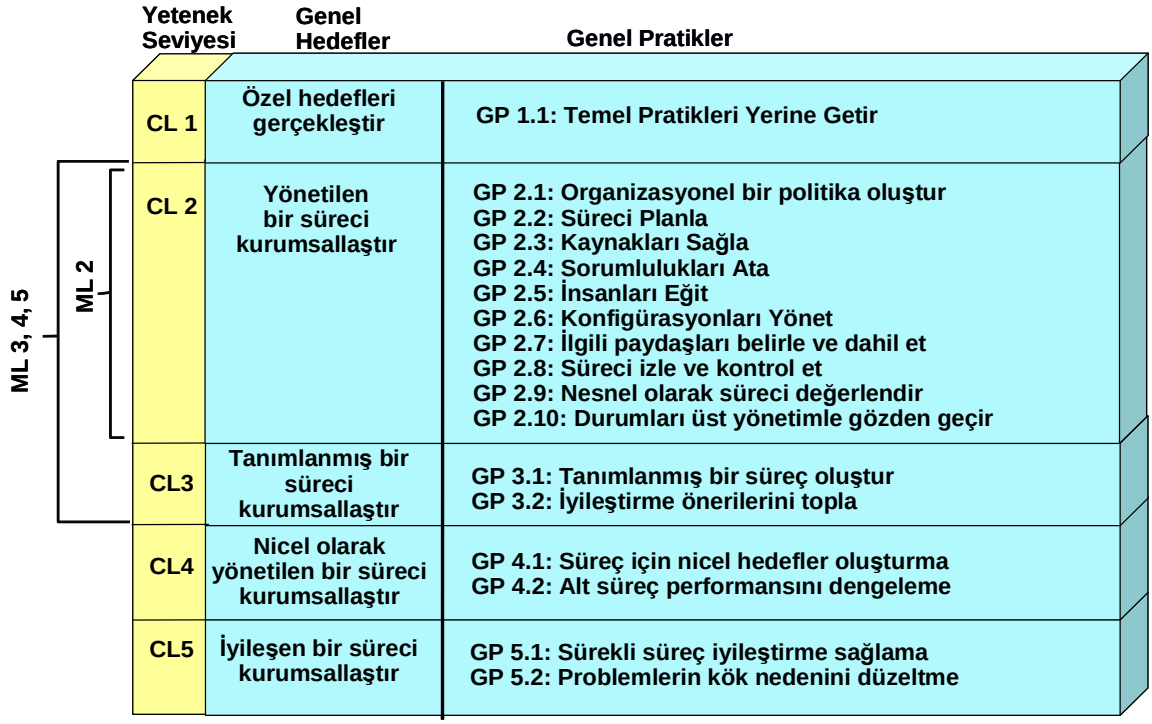
- ÖP1.1. Analiz edilecek hata verilerini seç
- ÖP1.2. Sebepleri analiz et

ÖH2. Hataların sebeplerinin ele alınması

- ÖP2.1. Eylem önerilerini gerçekleştir
- ÖP2.2. Değişikliklerin etkilerini değerlendir
- ÖP2.3. Verileri kaydet

Yukarıda süreç alanları ve bu süreç alanlarını gerçekleştirmek için geliştirilmiş özel hedef ve özel pratikler açıklandı. Aşağıdaki tabloda da tüm ya da birden çok süreç alanları için geçerli olan genel hedeflerin ve bu hedefleri sağlayan genel pratiklerin listesi sıralanmıştır.
(ML: Olgunluk Seviyesi, CL: Yetenek Seviyesi, GP: Genel Pratik)

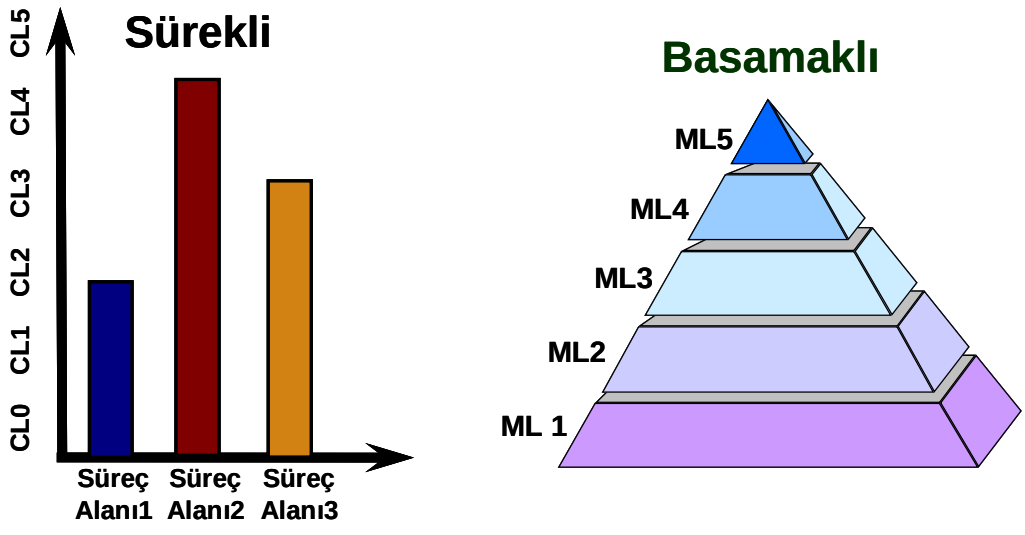
Tablo 2.1 - CMMI genel hedefleri ve pratikleri



2.4. CMMI Gösterimleri ve Seviyeleri

CMMI modelinin iki şekilde gösterimi vardır.

1. Basamaklı (staged) gösterim
2. Sürekli (continuous) gösterim

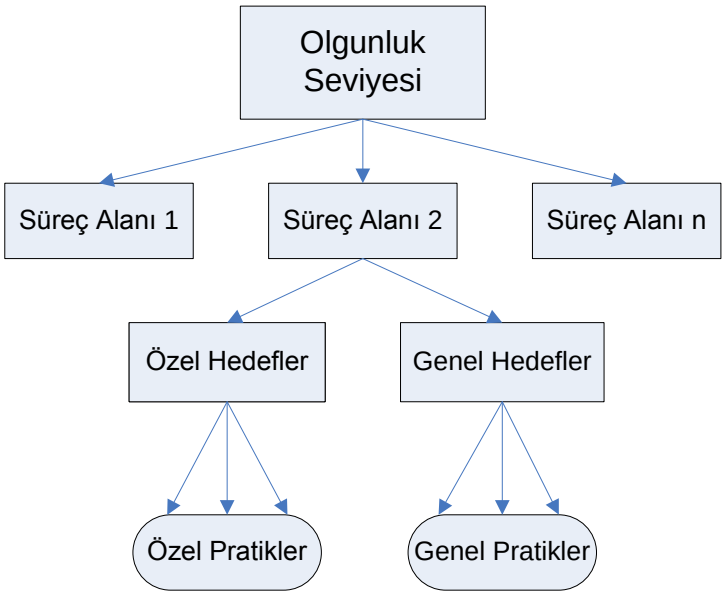


Şekil 2.3 - CMMI gösterimleri

Her iki gösterimdeki içerik aynı olmasına karşın bu verinin gösterimi ve düzenlenmesi birbirinden farklıdır. Farklı gösterimlerin amacı kurumların farklı iyileştirme hedeflerini takip edebilmelerini olanak sağlamaktır. Bu nedenle CMMI sertifikasını almak için niyetlenen bir kurum, belirlenmiş ihtiyaçları için en uygun olan CMMI gösterimini seçmelidir.

2.4.1. Basamaklı Gösterim

CMMI modeli süreç alanlarını "olgunluk seviyesi" adı altında beş gruba ayırmıştır. Olgunluk seviyeleri, kurumun "Hangi süreçler benim için en uygun? Nereden başlamalıyım? İzlemem gereken doğru sıralama nedir?" sorularına cevap bulmasını kolaylaştırır. Kurum bir olgunluk seviyesini hedeflemek ile hangi süreç alanlarını hedefleyeceğini seçmiş olur. Kurumun hedeflemesi gereken seviye şu anda sağladığı en düşük olgunluk seviyesinden bir fazlasıdır. Yani her bir olgunluk seviyesi bir sonraki seviyedeki süreçlerin etkin bir şekilde gerçekleştirilmesi için gerekli olan alt yapıyı sağlar.



Şekil 2.4 – CMMI basamaklı gösterim

Bütün kurumlar kesin olarak en azından birinci olgunluk seviyesini karşılarlar. Birinci olgunluk seviyesinde olmak için hiç bir şey yapmanız gerekmez. Kurumun var olması yeterlidir. Dolayısı ile hedeflenmesi gereken en düşük olgunluk seviyesi ikinci olgunluk seviyesidir. Aşağıdaki tabloda hangi olgunluk seviyesinde hangi süreç alanlarının yer aldığı gösterilmiştir.

Tablo 2.2 - CMMI basamaklı gösterim süreç alanları

Olgunluk Seviyesi	**Süreç Alanları**
1. Olgunluk Seviyesi - Başlangıç	-
2. Olgunluk Seviyesi – Yönetilen	Gereksinim Yönetimi Proje Planlama Proje İzleme ve Takip Tedarikçi Sözleşme Yönetimi Konfigürasyon Yönetimi Ölçme ve Analiz Süreç ve Ürün Kalite Güvencesi
3. Olgunluk Seviyesi - Tanımlı	Gereksinim Geliştirme Teknik Çözüm Ürün Bütünleştirme Doğrulama Geçerleme Kurumsal Süreç Odaklılık Kurumsal Süreç Tanımlama Kurumsal Eğitim Bütünleşik Proje Yönetimi Risk Yönetimi Karar Analizi ve Çözümleme
4. Olgunluk Seviyesi – Nicel Yönetilen	Kurumsal Süreç Performansı Nicel Proje Yönetimi
5. Olgunluk Seviyesi – İyileşen	Kurumsal Yenilik ve Yayma Nedensel Analiz ve Çözümleme

Seviye 1: Başlangıç (Initial)

- Yazılım süreci gelişigüzel ve kaotiktir.
- Başarı kişisel gayretlere bağlıdır.
- Kriz durumunda var olan planlar terk edilir.
- Yazılım süreç yetenekleri önceden kestirilemez.

Seviye 2: Yönetilen (Managed)

- Proje yönetimi kontrolleri kuruludur, projeler belgelenen planlarına uygun şekilde gerçekleştirilir ve yönetilir.
- Yazılım gereksinimleri yönetilir, gereksinimler ile ilişkili ürünler oluşturulur ve denetlenir.
- Proje yönetim sistemi proje sürecini etkin bir şekilde kontrol eder, projeler planlı, başarılmış, ölçülmüş ve kontrol edilmiştir.
- Gereksinimler, süreçler, çıktı iş ürünleri ve hizmetler yönetilir. İş ürünlerinin ve sağlanacak hizmetlerin durumu tanımlandığı noktada yönetimin görüşüne açıktır.
- Taahhütler, ilgili paydaşların istek ve revizyonlarına uygun şekilde tesis edilmiştir.
- İş ürünleri taraflarla gözden geçirilir ve kontrol edilir.
- İş ürünü ve hizmetler; onlara özel gereksinim, standart ve hedefleri sağlar.

Seviye 3: Tanımlı (Defined)

- Süreçler iyi tanımlanmış ve iyi anlaşılmıştır, standartlar, prosedürler, araçlar ve metotlar ile anlatılmıştır.
- Standart süreç tanımları ile kurumsal boyutta tutarlılık sağlanır.
- Projeler standart süreci uyarlama rehberlerine uygun olarak uyarlayarak, kendi süreçlerini oluşturur. Tüm projelerde standart yazılım geliştirme sürecinin uyarlanmış hali kullanılır.
- Süreçler 2. seviyeye göre daha özenli ve detaylı olarak tanımlanmıştır.

Seviye 4: Nicel Yönetilen (Quantitatively Managed)

- Sürecin başarımı için önemli alt süreçler seçilir ve bu alt süreçler istatistiksel ve diğer nicel ölçüm teknikleri kullanılarak kontrol edilir. (Yazılım sürecinin ve ürün kalitesinin ölçümleri toplanır ve kullanılır).
- Kalite ve süreç performansının nicel hedefleri tesis edilir ve süreçlerin yönetiminde bir kriter olarak kullanılır. Nicel hedefler müşterinin

ihtiyaçları/istekleri, son kullanıcı, organizasyon ve süreçlerini geliştirenler baz alınarak belirlenir.

- Süreçler ölçülür ve ölçülebilen sınırlar içinde işler gerçek veriler baz alınarak tahmin edilebilir. 3.seviye ile temel faklılık süreç performansının öngörülebilmesidir. 4.seviyede süreçlerin performansı istatistiksel ve diğer nicel teknikler ile kontrol edilebilir ve nicel olarak öngörülebilir. 3.seviyede süreçler sadece niteleyici olarak öngörülebilir.

Seviye 5: İyileşen (Optimizing)

- Süreçten gelen sayısal geri beslemeler ve teknolojilerden yararlanılarak sürekli süreç iyileştirilmesi yapılır. Tüm organizasyon süreç iyileştirmeye odaklanmıştır.
- Yazılım süreç yeteneği sürekli iyileşen olarak tanımlanabilir
- 5. ve 4.seviye arasındaki önemli bir ayrım, adreslenen 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 (süreç başarımından istatistiksel öngörüye) adreslemekle ilgilenir.

2.4.2. Sürekli Gösterim

Tek bir süreç alanındaki (Process Area) iyileştirmeyi gösterir. Bu gösterimim cevap verdiği temel soru: Belirli bir X süreç alanında iyileştirme sağlayabilmek için izlemem gereken doğru sıralama nedir? Bu gösterim, kurumların bir veya bir grup süreç alanı seçerek süreçlerini bu doğrultuda geliştirmelerine imkan sağlar. Yani bir süreç alanındaki iyileştirmeyi gösterir. “Belirli bir X süreç alanında iyileştirme sağlayabilmek için izlemem gereken doğru sıralama nedir?” sorusuna cevap verir.

Sürekli gösterimde bir süreç alanındaki gelişme, yetenek seviyeleriyle ifade edilir. Kurum hangi süreçlerinin geliştirilmesine ihtiyaç duyduğunu belirleyebiliyor ve bu süreç alanları arasındaki ilişkiyi biliyorsa bu gösterim, basamaklı gösterime göre kurum için daha iyi bir seçimdir.

Bu gösterimde 6 Adet Yetenek /Ehliyet (Capability) Seviyesi vardır. Bir süreç alanının alabileceği en düşük yetenek seviyesi 0(sıfir)'dır. Bu, o süreç alanının kurumda olmadığını ifade eder. Süreçlerin ulaşabileceği tüm yetenek seviyeleri şunlardır:

- Seviye 0: Eksik (Incomplete)
- Seviye 1: Gerçekleştirilen (Performed)
- Seviye 2: Yönetilen (Managed)
- Seviye 3: Tanımlı (Defined)
- Seviye 4: Nicel Yönetilen (Quantitatively Managed)
- Seviye 5: İyileşen (Optimizing)

Sürekli gösterimde süreç alanlarının kategorilere göre listelenişi aşağıdaki Tablo 2.3'te verilmiştir.

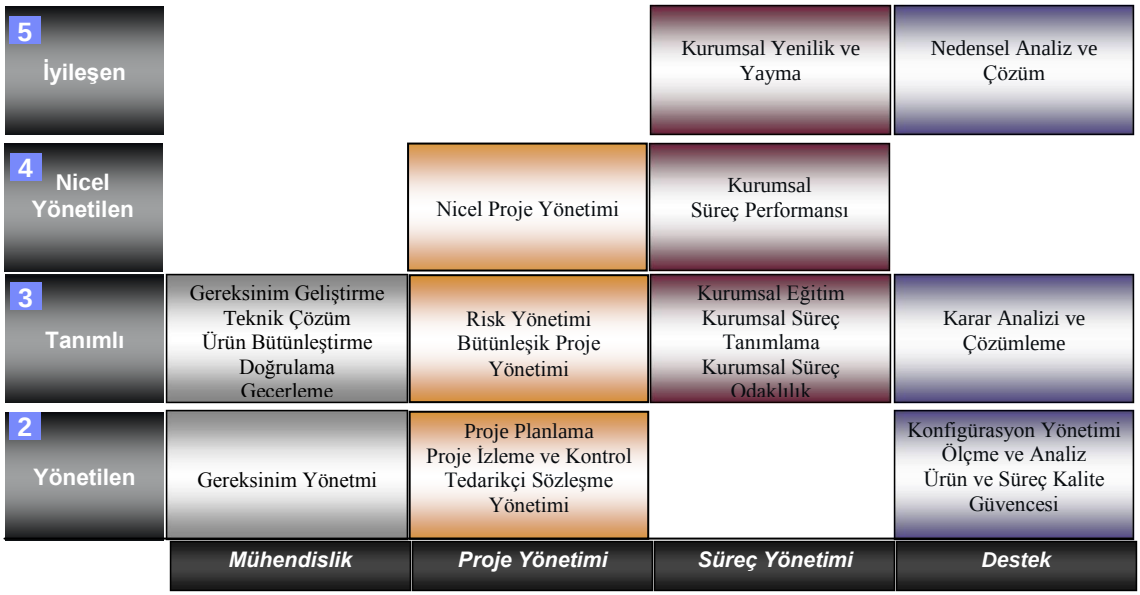
Tablo 2.3 - Sürekli gösterim süreç alanları

CMMI Kategorisi	**Süreç Alanları**
Proje Yönetimi	Proje Planlama Proje İzleme ve Kontrol Tedarikçi Sözleşme Yönetimi Bütünleşik Proje Yönetimi Risk Yönetimi Nicel Proje Yönetimi
Destek	Konfigürasyon Yönetimi Süreç ve Ürün Kalite Güvencesi Ölçme ve Analiz Karar Analizi ve Çözümleme Nedensel Analiz ve Çözümleme
Süreç Yönetimi	Kurumsal Süreç Tanımlama Kurumsal Süreç Odaklılık Kurumsal Eğitim Kurumsal Süreç Performansı Kurumsal Yenilik ve Yayma

Mühendislik	Gereksinim Yönetimi Gereksinimlerin Geliştirilmesi Teknik çözüm Ürün bütünleştirme Doğrulama Geçerleme

Sürekli gösterim kullanıldığında, süreç alanları için ayrı ayrı yetkinlik seviyeleri tespit edilir. Sürekli gösterim, süreç iyileştirme için süreç alanları içinden iyileştirmek istediğiniz süreç alanlarını tek tek seçmenize izin veren bir yaklaşımdır. Bu gösterimi kullanmak için hem kurumunuzu hem süreç alanlarını ve hem de süreç alanları arasındaki ilişkileri çok iyi bilmeniz gereklidir. Aksi takdirde hazır olmadığınız bir süreç alanını seçtiğiniz ya da o süreç alanını destekleyecek diğer süreç alanlarını birlikte iyileştirmek için seçmediğiniz için uzun ve pahalı bir çalışmayı sonuçsuz bırakma riskiniz vardır[6].

Her iki gösterim de organizasyonun süreçlerini ne kadar iyi geliştirebildiğinin ölçülmesine olanak sağlar. Sadece, süreç iyileştirmek için yaklaşım farklıdır. Aşağıdaki şekilde süreç alanlarını kapsayacak bir gösterimle bu yaklaşımların benzerliğini farkını göstermektedir.



Şekil 2.5 - CMMI gösterimleri ve süreç alanları

2.5. CMMI'ın Faydaları

SEI'in Aralık-2005 rakamlarına göre 25 büyük organizasyondan elde ettiği faydaların listesi şöyledir:

Tablo 2.4 - CMMI'ın faydaları

Performans Kategorisi	**Ortalama**	**Veri kaynağı sayısı**	**En az**	**En çok**
Maliyet	%20	21	%3	%87
Proje Takvimi	%37	19	%2	%90
Üretkenlik	%62	17	%9	%255
Kalite	%50	20	%7	%132
Müşteri Tatmini	%14	6	%-4	%55
ROI (Geri dönen fayda)	4.7 : 1	16	2 : 1	27.7 : 1

SEI'nin 2006 yılında 30 organizasyona yaptırdığı aynı araştırmada Maliyet kazancı ortalaması %34'lere, Proje Takvimi kazancı ortalaması da %50'lere çıkmıştır. Diğer değerlerde çarpıcı bir gelişme gözlenmemiştir.

.

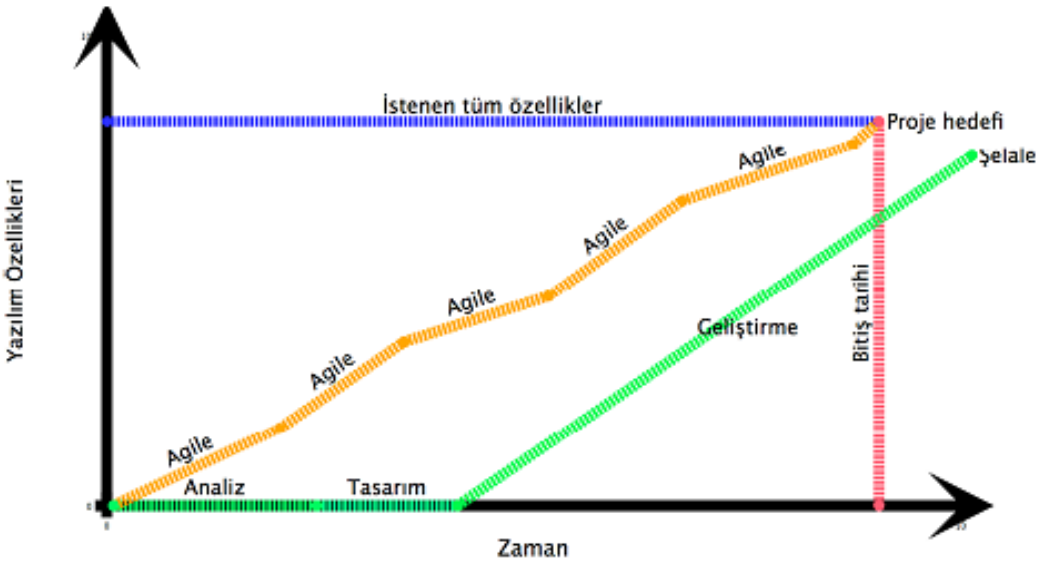
3.BÖLÜM

ÇEVİK GELİŞTİRME

3.1. Çevik (Agile) Geliştirme Nedir?

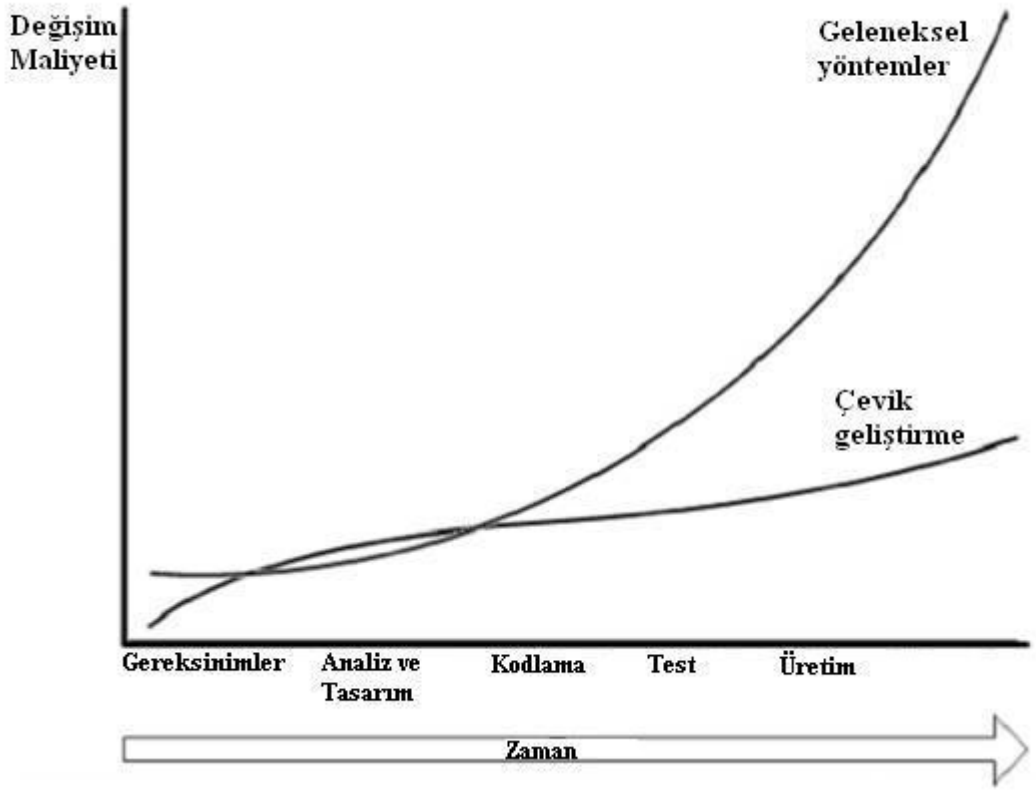
Agile kelime olarak hızlı ve atik düşünmek, akıllı bir yolu tercih etmek demektir. Yazılım geliştirmede agile(çevik) kelimesi ise değişen ihtiyaçlara hızlı cevap veren pratikleri ifade eder. Çevik yazılım süreçleri, 1950'lerdeki üretim alanında verimliliğin artırılması için geliştirilen yalın yaklaşımların yazılım sektöründe bir uzantısı olarak ortaya çıkmıştır. Yazılım dünyasında çeşitli çevik yaklaşımlara 1970'lerden itibaren rastlanabilmekle birlikte, çevik yazılım geliştirme yöntemlerinin kullanımı 1990'larda hız kazanmış ve geçtiğimiz son 7–8 yıl içerisinde de tüm dünyada başarılarını kanıtlayarak popülaritesini arttırmıştır. Şu anda, dünyadaki birçok yazılım şirketinde ve birçok yazılım projesinde yazılımlar, çevik yaklaşımlarla geliştirilmektedir. Çevik yaklaşım, özellikle şu an bilişim sektörünün en güçlü olduğu Amerika ve İngiltere gibi ülkelerde altın zamanını yaşıyor. Çok büyük şirketler bile çevik yazılım geliştirme yöntemlerinden yarar sağlayarak çalışmalarını yürütüyorlar. Şirketlerin ilgisindeki artış en büyük nedeni artan rekabet koşulları, şirketlerin IT projelerine yaptıkları yatırımların sonucu daha hızlı görmek istemeleri, çevik çözümlerin bu ihtiyaçlara cevap verebilmesi ve müşteriye katma değer sağlamaya odaklı olmasından kaynaklanıyor.

Bilindiği gibi geleneksel yazılım geliştirme süreçleri, projelerde değişen gereksinimleri karşılamak ve beklenen kalite faktörlerine erişmek açısından çok katı ve ağırdır. Mesela Şelale modelinde hedefe doğru ilerlemeden önce uzun ve detaylı analiz, tasarım, gözden geçirmeler, onay süreçleri bulunur. Proje başlangıcından bir süre sonra çalışan yazılım özellikleri geliştirilmeye ve hedefe doğru ilerlemeye başlanır. Bu modelde proje başlangıcında detaylı çalışmalarda çok zaman kaybedilir ve hedefe ulaşılamayacağı projenin ancak sonuna doğru belli olur.



Şekil 3.1 - Çevik geliştirmenin Şelale sürecinden farkı

Detaylı çalışmalar sonunda oluşturulan yapıda ilerleyen aşamalarda değişikliğe gidilmesi büyük maliyetler oluşturur. Çevik yazılım geliştirme ise süreçlerin katı kurallarla yönetilmesi yerine amaca yönelik verimli ve etkin pratiklerin yapılmasını esas alır, yani hem yazılım geliştirenlerin hem de müşterilerin ana hedeflerini karşılar. Değişimin maliyetini olabildiğince düşüren pratikler içerir.



Şekil 3.2 - Çevik geliştirmenin değişim maliyeti

Çevik yazılım geliştirme yöntemleri, tekrarlanan yazılım geliştirme metodu taban alınarak geliştirilmiştir. Sık aralıklarla parça yazılım teslimatını, değişimi, takım içerisindeki iletişimin arttırılmasını, test odaklı yazılım geliştirilmesini ve uyumlu planlamayı teşvik eder. Bu özellikleri nedeniyle kendine her geçen gün artan bir kullanım alanı bulmaktadır.

3.2. Çevik Manifesto

Kent Beck ve dünyanın önde gelen çevik yazılım geliştiricisi 16 arkadaşı (XP, Scrum gibi metodolojilerin yaratıcıları) ortak bir zeminde buluşabilmek adına 13 Şubat 2001'de Utah'da bir araya gelerek "çevik yazılım geliştirme manifestosu"nu yayınlamışlardır. Böylelikle çevik yöntemlerin projelere genel bakış açıları ifade edilmiştir. Bu manifestoda çevik geliştirmenin öncelikleri ve prensipleri sıralanmıştır.

Şekil 3.3 - Çevik manifesto

3.2.1. Çevik Geliştirme Öncelikleri

Çevik manifesto ile çevik sürecin bir değer sistemine sahip olması ve geleneksel yazılım metotlarından elde edilen tecrübeler doğrultusunda daha pratik ve çevik bir yapıda olması gerektiği dile getirilmiştir. Öncelikler şöyle sıralanmıştır.

- **Kişiler ve iletişim**, *süreç ve araçlardan* önce gelir.

  İnsan başarının en önemli anahtarıdır. Güçlü oyunculardan oluşmayan takımların süreçlerin başarılı olması mümkün değildir. Aynı şekilde kötü bir süreç, mükemmel oyuncuları faydasız hale getirebilmektedir. Burada güçlü takım oyuncusundan kasıt çok iyi bir programcı değil, takımın diğer üyeleri ile başarılı bir şekilde iletişim kurabilen ve çalışabilen ortalama bir programcıdır. Kişi ve iletişim öncelikli olsa da süreçlerin yapılandırılması ve süreçlerde kullanılan araçların etkinliği de önemlidir. Çünkü geliştirme sürecinde kullanılan derleyiciler, geliştirme ortamları (IDE - Integrated Development Environment), kaynak kod kontrol sistemleri gibi araçlar geliştirme takımına uygun fonksiyonlar sağlayabilir.

- **Çalışır durumda olan yazılım**, *detaylı dokümantasyondan* daha önceliklidir.

  Dokümantasyonu olmayan bir yazılım felakettir. Çünkü program kodları iletişim için ideal bir ortam değildir. Proje takımı sistemi tanımlayan veya tarif eden belgeleri oluşturmaya mutlaka ihtiyaç duyar. Fakat bu dokümantasyon faaliyetleri çoğaldıkça yazılım geliştirme süresi azalır. Bunun dengesinin sağlanması önemlidir. Ayrıca müşterinin asıl ihtiyaç duyduğu şey çalışan yazılım olduğuna göre dokümantasyon çalışmaları gerektikçe gerçekleştirilmeli ve asıl amaçtan uzaklaştıran bir duruma dönüşmemelidir.

- **Müşteri ile işbirliği ve beraber çalışma**, *sözleşmelerden ve anlaşmalardan* daha önceliklidir.

  Başarılı projeler düzenli ve sık aralıklarla müşteri geri beslemesine ihtiyaç duyar. Müşterinin geliştirme takımına yakın çalışması bunun sağlanması açısından çok önemlidir.

- **Değişikliklere uyum sağlayabilmek**, *mevcut planı takip etmekten* daha önemlidir.

  Değişikliklere cevap verebilme yeteneği, yazılım projesinin başarılı ya da başarısız olmasını belirleyen çok önemli bir özelliktir. Bu nedenle planların

esnek olmasına, süreç boyunca teknolojik ve iş alanındaki değişikliklere uyum sağlayabileceğinden emin olmak gereklidir. Bunu sağlamak için planlamanın yakın zamanda gerçekleşecek planları detaylı, sonraki zamanlarda gerçekleşecek planları ise daha yüzeysel ve detaysız oluşturulmalıdır. Böylelikle zamanla belirsizliği azalıp netleşecek gereksinimlerin ortaya çıkaracağı yeni durumlara adaptasyon sağlanabilir.

3.2.2. Çevik Geliştirme Prensipleri

Çevik manifestoda yer alan ifadeler on iki prensip ile somut hale getirilmekte ve açıklanmaktadır. Bunlar:

- *En büyük öncelik, sürekli, kaliteli yazılım teslimatıyla müşteri memnuniyetini sağlamaktır.* Bu projenin ilk aşamalarından itibaren ara teslimler yapılır ve müşterinin yazılımı çok önceden kullanmaya başlayarak değer sağlamasına olanak sağlanır. Günümüzde çevik süreçlerin popülaritesinin başlıca nedenlerden biri, yapılan yatırımların hızlı geri dönüş sağlamasıdır.
- *Proje ne kadar ilerlemiş olursa olsun değişiklikler kabul edilir. Çevik yazılım süreçleri değişiklikleri müşteri avantajına dönüştürürler.* Eğer amaç müşteri memnuniyeti ise bu değişiklik isteklerinin yerine getirilmesi yazılımın kalitesi ve müşteri memnuniyeti açısından çok önemlidir. Test güdümlü geliştirme, otomatik testler, sürekli entegrasyon, basit tasarım, yeniden yapılandırma gibi pratikler sayesinde değişikliklerin getireceği maliyetler minimuma indirilir ve süreç değişikliklere hızla uyumlu hale getirilir. Bu nedenle çevik süreç katılımcıları değişikliklerden korkmazlar, aksine projenin sonunda oluşacak yazılımın daha "iyi" olması için fırsat olarak görürler.
- *Mümkün olduğunca kısa zaman aralıklarıyla (2-8 hafta arası [4]) çalışan, kaliteli yazılım teslimatı yapılır.* Projenin ilk zamanlarından itibaren müşteriye çalışan yazılım teslimi yapılır. Bu sayede sürekli geri besleme sağlanır ve müşterinin istekleri doğrultusunda yazılım evrimleşerek gelişir.
- *Analistler, alan uzmanları, yazılımcılar, testçiler vs. tüm ekip elemanları bire bir iletişim halinde, günlük olarak birlikte çalışırlar.* Rol bazlı ekipler yerine

yazılım özelliklerine göre ekipler oluşturulur ve bu ekipler içinde koordinasyon ve işbirliği sağlanır.

- *Projeler motivasyonu yüksek bireyler etrafında kurulur ve ekip üyelerine kendileri ile ilgili alacakları kararlar konusunda güvenilir.* Ekip elemanlarına gerekli destek verilir, ihtiyaçları karşılanarak kendi kendine organize olacak yetkiye sahip olması sağlanır.
- *Ekip içerisinde kaliteli bilgi akışı için yüz yüze iletişim önemlidir.* Çevik projelerde iletişimin en tercih edilen şekli diyalogdur. Projede dokümantasyon önemlidir ama her şeyin de kaydedilmesi gerekmez.
- *Çalışan yazılım, projenin en önemli gelişim ölçütüdür.* Çevik projeler hangi aşamada neler olduğunu ya da ne kadar dokümantasyon yapıldığını sorgulamaz. İstenen fonksiyonelliğin ne kadarının gerçekleştirildiğini sorgular.
- *Çevik süreçler mümkün olduğunca sabit hızlı, sürdürülebilir geliştirmeye önem verir.* Planlamaların sağlıklı olması için ekibin iş teslim hızının güvenilir olması gerekir. Fazla mesailer gibi yöntemlerle ekibin hızını geçici olarak arttırmak tercih edilen yöntemler değildir.
- *Teknik üstünlük ve iyi tasarım çevikliği arttırır.* Bu nedenle teknik açıdan mükemmel, sade çözümler oluşturulmasına özen gösterilir. Çünkü en iyi tasarım çabuk genişleyebilen tasarımdır.
- *Basitlik, sadelik önemlidir.* Sadelik, anlaşılması ve sonradan değiştirilmesi kolay, maliyeti düşük ve o anki gereksinimleri karşılayan çözümü kullanmaktır. Akla gelen ilk baştan savma çözümü uygulamak olarak algılanmamalıdır.
- *En iyi mimariler, gereksinimler ve tasarımlar kendini organize edebilen ekipler tarafından yaratılır.* Takım kendi çalışma yöntemlerini sorgulamakta ve gerekli değişiklikleri yapmakta özgürdür. Sorumluluklardan bireyler değil, takımın tamamı sorumludur. Sorumlulukların nasıl yerine getirileceğine takım kendi içinde karar verir, dışarıdan dikte edilmez.
- *Takımlar düzenli aralıklarla kendi yöntemlerini gözden geçirerek verimliliği arttırmak için gerekli iyileştirmeleri yaparlar.* Çevik takım kendi organizasyonunu, kurallarını, anlaşmalarını, ilişkilerini proje süresince yeniden uyarlayabilir.

3.3. Çevik Süreç Modelinin Faydaları

Çevik geliştirme ile elde edilen faydaların en önemlileri şunlardır:

- Çevik projelerde müşteri gereksinimleri (requirement) ve bu gereksinimlerin karşılığı olan program paralel olarak gelişir. Müşteri projenin gidişatına her zaman müdahale etme yetkisine sahiptir. Bu uzun süren projelerde önem taşımaktadır, çünkü çoğu zaman proje başlangıcında müşteri kendi gereksinimlerini tam olarak bilmeyebilir. Zaman içinde oluşan ilk prototipler müşterinin kafasında nasıl bir sistem istediği hakkında daha net bir resmin oluşmasını sağlar. Projedeki geri besleme (feedback) mekanizmalarıyla müşteri gereksinimleri her zaman değişikliğe uğrayabilir.
- Bu modelde geri beslenme merkezi bir rol oynamaktadır. Çeşitli safhalarda sağlanan geri beslenme ile projenin hangi durumda olduğu saptanır. Programcılar hazırladıkları testler aracılığıyla geri beslenme sağlayarak, geliştirdikleri bileşenlerin hangi durumda olduklarını tespit ederler.
- Sürekli entegrasyon yapılarak programın hangi durumda olduğu geri beslenme olarak elde edilir. Müşteriye kısa aralıklarla programın yeni sürümü sunularak, geri beslenme sağlanır.
- Proje başlangıcında detaylı dokümantasyon ve tasarım oluşturulmaz.
- Programcılar test güdümlü (Test Driven Developlement - TDD) çalışır ve oluşan tasarımı her yeni testle gözden geçirirler. Eğer tasarım yetersiz kalırsa, gerekli tasarım değişikliklerini, ilerideki testlerin çıkmaza girmesini engellemek için uygularlar.
- Her iterasyon başlangıcında müşteri tarafından dile getirilen gereksinimler analiz edilir ve geliştirilecek olanlar seçilir. Müşteri her gereksinim için bir öncelik sırası belirler. Öncelik sırası yüksek olan gereksinimler öncelikli olarak gerçekleştirilir. Her iterasyon sonunda gerekli değerlendirmeler yapılarak, oluşan problemler tartışılır ve tekrar meydana gelmelerini engellemek için gerekli önlemler alınır.
- Test güdümlü çalışıldığı için kod kalitesi çok yüksek olur. Gün boyunca programcılar kendilerine değişik bir programcıyı takım arkadaşı olarak seçerek (Pair Programming), implementasyonu gerçekleştirirler. Kısa bir zaman sonra

programcılar arasındaki teknik bilgi aynı seviyeye gelir ve her programcı programın herhangi bir bölümünde çalışacak hale gelir. Bu şekilde sadece bir programcının kod hakkında bilgi sahip olması engellenir.

- Programcılar aktif olarak proje planlamasında yer alırlar. Onlar gereksinimlerin tespitinde müşteriye yardımcı olurlar ve zaman tahminlerinde bulunarak, proje planlaması için gerekli verilerin oluşturulmasını sağlarlar. Bu programcılara belirli bir sorumluluk yükler. Kendisine güvenildiğini bilen ve sorumluluk sahibi bir programcının öz güveni ve motivasyonu artar.
- Çevik projelerde iyi bir çalışma ortamının ve temposunun oluşturulması fazla mesai yapılmasını engeller. Fazla mesai yapılmayacak diye bir kural yoktur. Fakat fazla mesai bir kural haline gelmemelidir. Bu durum tüm ekibin motivasyonunu negatif etkiler. Hatalar genelde isteksizce yer alınan fazla mesailerde meydana gelir. Proje çalışanları sekiz saat olan ve fazla mesai yapılmayan iş günlerinde daha verimli olurlar.
- Müşteriye kısa aralıklarla çalışabileceği bir sürüm sunulur. Program tamamlanmamış olsa bile, müşteri hazır bölümleri kullanarak, yaptığı yatırımın hızlı şekilde geri dönmesini sağlanır.
- Programcılar ve müşteri arasında devamlı iletişim vardır. Programcılar soru ve sorunları müşteri ile paylaşarak, kısa sürede çözüm üretebilirler.
- Müşterinin piyasadaki değişikliklere ve bununla birlikte rekabet ortamına ayak uydurabilmesi önemlidir. Çevik süreç hızlı reaksiyon göstererek, bu değişikliklere ayak uydurulmasını sağlar. Rekabete ayak uydurabilmek için hızlı reaksiyon gösterebilmek hayatı bir önem taşımaktadır.

3.4. Çevik Geliştirme Yöntemleri

Çevik süreçler yukarıda sayılan öncelik ve prensipleri içeren manifestoyu kabul eden ve çalışma yöntemlerini çeviklik bakış açısıyla oluşturmuş süreçlerdir. Zaman içinde, bir meta model olarak kabul edebileceğimiz çevik süreci implemente eden değişik çevik süreç türleri oluşmuştur. Bunların çoğu çevik manifestodan sonra oluşmuştur. Ama bazıları çevik manifesto öncesi de mevcuttu ve kullanılmaktaydı. Şuanda yaygın olarak kullanılan en önemli çevik süreç türleri şunlardır:

- **Scrum**

  Scrum seksenli yıllarda Jeff Sutherland tarafından uygulanmış ve Kent Schwaber ve tarafından da detaylandırılmış bir çevik süreçtir[32]. Sutherland ve Schwaber daha sonra birlikte çalışmış, Microsoft, Borland ve Hewlett-Packard'da oluşan yöntemsel deneyimlerle birlikte endüstriyel kontrol yöntemlerini uygulayarak Scrum'ı yeniden şekillendirmişlerdir. Aslında Scrum, Rugby oyununda kullanılan bir terimdir. Oyuncular kısa bir süre için bir araya gelerek, bir sonraki oyun hamlesi hakkında fikir alış verişinde bulunurlar, yani kısa bir toplantı yaparlar. Scrum daha çok proje yönetim ve planlama ile ilgili değerlere ve pratiklere konsantre olmaktadır. Yazılımın nasıl yapılması gerektiği hakkında detay ihtiva etmez. Bağımsız bir çevik yöntem olarak algılanmamalıdır. Birçok projede Scrum, Extreme Programming (XP) ya da Unified Process (UP) ile desteklenir[33].

- **Extreme Programming (XP)**

  XP, doksanlı yılların sonunda Kent Beck, Ron Jeffries ve Ward Cunningham tarafından[31] Chrysler için yapılan bir proje sonrasında oluşmuş bir çevik süreçtir. XP Scrum'dan esinlenilerek geliştirilmiştir. Ama XP Scrum'ın aksine daha çok yazılım geliştirme pratiklerine konsantre olmaktadır. Bu sebepten dolayı Scrum ve XP projelerde birlikte kullanılabilir. Özellikle gereksinimlerin belirsiz, değişimin hızlı olduğu küçük ve orta büyüklükte yazılım geliştirme takımları için çözümler sunar. Ana amacı müşterinin ihtiyaçlarını karşılayan yüksek kalitede ürünü sağlayarak müşterinin güveni kazanmaktır. Genellikle test pratikleri ile dikkat çeker. Test Güdümlü Geliştirme (Test Driven Development), Sürekli Entegrasyon (Continuous Integration), Otomatik Test (Test Automation) konusundaki uygulamalar XP'nin çığır açan pratikleridir. Bu pratiklerin çevik model karakterinde olmayan projelerde bile kullanılması yararlar getirecektir.

- **Agile Modeling (AM)**

  AM, yazılım sistemlerinin modelleme ve etkili dokümantasyon işlerini kolaylaştırmak üzere bu işleri iş ürünü olarak fazla dikkate almayan ve önemsemeyen diğer çevik yöntemleri tamamlayan bir yapıdır. Yazılım

mühendisleri tarafından uygulanacak bazı prensip ve değerlerin belirlediği pratikler kümesidir. Tamamen detaylandırılmış özel bir süreç modelleme yöntemi tanımlamaz. Sadece modellemenin daha etkin olması için tavsiyelerde bulunur. AM gereksinimler, mimari ve tasarım modelleme işlerinde kullanılabilir [34].

- **IXP**

  XP'den doğan IXP'nin (Industrial XP5) amacı XP yi geliştirmek ve XP'de yer alan metot ve tekniklerin daha büyük organizasyonlar için adapte etmektir.

- **Evo**

  Tom Gilb tarafından oluşturulmuş en eski çevik yöntemdir[35]. Gilb 1976'da yinelemeli geliştirme ve evrimsel yönetim konularıyla ilgilenmeye başladı ve 1981'de bu konuları detaylı inceleyen "Evolutionary Development(Evrimsel Geliştirme)" kitabını yayınladı. Doksanlarda Gilb kısaca "Evo" denilen ve XP, Scrum hatta UP gibi yöntemleri etkileyen bu çevik yöntemi geliştirmeye devam etti. Bugün bu modelin 5 ana elemanı vardır: hedefler, değerler ve maliyetler, çözümler, güçlü tahminler, evrimsel plan ve fonksiyonlar.

- **Dynamic Systems Development Model (DSDM)**

  İngiltere kaynaklı hızlı uygulama geliştirme(RAD) için geliştirilmiş bir çerçevedir[38]. XP, Scrum gibi süreçlerin işleyişi ile karşılaştırıldığında daha fazla detay içerir ve genelde devlet ihaleleri gibi yazılım projelerinde sıklıkla kullanılır. Katı zaman kuralları gerektiren projelerde iş çözümleri sunar. XP, UP ya da diğer çevik yöntem kombinasyonları ile tamamlanabilir.

- **Crystal Metholodogies**

  Alistair Cockburn tarafından geliştirilmiş bir yöntemler topluluğudur[36]. Bu yöntemler yazılım geliştirme ile diğer mühendislik süreçleri arasında karşılaştırma yaparak yazılımın özellikleri ve modelleri, tamamlanması, doğruluğu ve çalışması konularına yoğunlaşır. Crystal Methodologies'de 4 farklı

ana yöntem vardır. Biri en fazla 8 kişilik takımlara uygulanabilen Crystal Clear'dir. CC en çok dokümante edilmiş Crystal yöntemidir. Sarı yöntem 8 ile 20, Turuncu yöntem 20 ile 50, Kırmızı yöntem ise 50 ile 100 kişi arasındaki takımlara uygulanır. Bu yöntemler kahverengi, mavi ve mor diye devam eder. XP gibi süreçlere nazaran daha esnektir ve geliştirme pratikleri konusunda kısıtlar koymaz. Sabit fiyat (Fixed price) projeler ile ilgili çözümler sunar. Orta düzeyde kritik küçük projelerde kullanılabildiği gibi bazı eklemelerle kritik projelerde de uygulanabilir.

- **Feature Driven Development (FDD)**

  Jeff DeLuca tarafından doksanlı yılların sonunda geliştirilmiş bir çevik süreç türüdür. Diğer çevik yöntemlerden farklı olarak tüm yazılım hayat döngüsünü kapsamaz. Tasarım ve geliştirme bölümleri için çözümler sunar. FDD, yazılım özelliği (feature, function) güdümlü çalışır. Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur. Bu yüzden FDD daha çok tasarım odaklı işleyen bir çevik süreçtir. Alan modellemesi, yazılım özellikleri çevresinde ekiplerin oluşturulması, tanımlı kilometre taşları gibi daha detaylı çözümler içerir. Önemli kritik projelerde uygulanabilir[37].

- **Adaptive Software Development (ASD)**

  Büyük ve karmaşık sistemlerin geliştirmesi konusunda oluşan problemlere yoğunlaşır. Bu artırımlı ve yinelemeli yöntem, prototip geliştirme şeklinde uygulanır.

Bu çevik yöntemlerden en yaygın kullanılanları birbirini tamamlayan Scrum ve XP'dir ve genelde de birlikte kullanılırlar.

4.BÖLÜM

SCRUM

4.1. Scrum Nedir?

Scrum Jeff Sutherland ve Ken Schwaber tarafından 1990'ların ortalarında geliştirilmiştir. Yüksek seviyede belirsizlik arz eden projelerde son yıllarda yoğun biçimde uygulanan ve başarılı sonuçlar üreten bir tür metodolojidir. Scrum, küçük takımlarda (<20) uygulanması kolay olan bir proje yönetimi yöntemi olup, büyük projelerde takımların küçük parçalara ayrılması ile etkinleştirilir. Değişimi bir istisna olarak görmeyen, adaptif sistem geliştirme hayat döngüsü perspektiflerinin genel özelliğine paralel olarak proje kapsamının proje boyunca sürekli değişeceğini en baştan kabul eden bir yöntemdir. Bu kabul doğrultusunda Scrum, fazla dokümantasyon yapmadan, kısa periyotlarda geliştirme yapılıp, sonuçların müşteriye sıkça gösterildiği ve geri beslemeler doğrultusunda geliştirmelerin yönünün sürekli değiştiği, gereksiz geliştirmelerin minimuma indirilmeye çalışıldığı bir sürecin yönetimini üstlenir.

Scrum, süreçlerini zeki çözümlerle tasarlamaları için geliştiricileri özgür bırakır. Temel olarak nesne tabanlı teknolojiyi benimser. Böylece birbirinden ayrık ve yönetilebilir ortamlar sağlar. Fakat basit arayüzlerle ve güçlü veri oryantasyonu ile CMMI gibi disiplinli süreç değerlendirme sistemlerine de uyarlanabilir.

Scrum'da 4 tür toplantı vardır.

- Sprint planlama toplantısı (Sprint Planning Meeting)
- Sprint gözden geçirme toplantısı (Sprint Review Meeting)
- Geriye Bakış toplantısı (Retrospective Metin)
- Günlük Scrum toplantısı (Daily Scrum Meeting)

Ayrıca Scrum için çok önemli üç kavram daha vardır.

- Ürün gereksinim dokümanı (Product Backlog)
- Sprint dokümanı (Sprint Backlog)
- Aksaklık dokümanı (Impediment Backlog)
- Sprint kalan zaman grafiği (Burndown Chart)

Şimdi bu kavramlar ve Scrum rolleriyle şekillenen proje sürecini inceleyelim.

4.2. Scrum Rolleri

Scrum süreçlerinde ana roller(pig roles) ve yardımcı roller(chicken roles) olmak üzere iki tür rol vardır. Ana roller; Ürün Sahibi, Scrum Yöneticisi (Scrum Master), Takım Üyesidir. Yardımcı roller ise Paydaşlar (müşteriler, satıcılar) ve Yöneticilerdir.

- **Ürün Sahibi (Product Owner)**

  Alan bilgisine sahip, müşteri isteklerini anlayabilen, gereksinimleri belirleyen ve belgeleyen kişidir. Müşterinin takımdaki yansımasıdır. Asıl görev ve sorumluluğu sürekli takım ve müşteriyle iç içe çalışmak ve bunun sonucunda takımı yönlendirmektir. Ne yapılması gerektiğine ve sürümün ne zaman teslim edilmesi gerektiğine karar verir. Ürün Sahibi, klasik proje yönetimi yaklaşımındaki Proje Yöneticisi'nin sorumluluklarının bir kısmını üzerine alır. Ürünün karlılığından (ROI) doğrudan sorumludur. Ürün gereksinimlerini, ihtiyacın müşteri gözündeki ve pazardaki kıymetine göre önceliklendirir. Belirli periyotlarla yapılan toplantılarda ürün gereksinimlerinin önceliklerini değiştirebilir. Projenin çıktılarını kabul veya reddeder. En kritik nokta olan projenin başarısından veya başarısızlığından tamamen Ürün Sahibi sorumludur. Henüz belirli bir müşterisi olmayan projelerde ise alan bilgisine sahip kurum içinden biri veya destek alınan bir danışman ürün sahibi olabilir.

- **Scrum Yöneticisi (Scrum Master)**

  Scrum süreçlerinin istenildiği gibi gidip gitmediğinden sorumludur. Scrum yöneticisi takımın verimliliğini en üst düzeye çıkarmaya çalışır ve proje

sırasında karşılaşılan engelleri ortadan kaldırmaya çalışır. Takımın lideri değildir ama takımla diğer dış etkiler arasında bir tampon görevi görür. Takım üyelerini kurallara zorlayan kişidir. Proje süreci hakkındaki bilginin tüm takım üyelerine açık olması ve bu bilginin güncellenmesinden sorumludur. Sprint toplantılarını organize eder, yönetir ve toplantıların amacına ulaşması için gerekli önlemleri alır.

- **Takım üyesi (Team Member)**

  Takım üyeleri ürünün yetiştirilmesinden sorumludur. Genel farklı yeteneklere(tasarım, geliştirme, test, teknik iletişim gibi) sahip 5-9 kişiden oluşur.

- **Yönetici (Manager)**

  Ürünün geliştirildiği organizasyonda ürünün geliştirilmesi için çevresel şartları sağlayan kişilerdir.

- **Diğer Paydaşlar (Other Stakeholders)**

  Müşteri (customers), satıcı (vendors) ve tedarikçilerden (supplier) oluşur. Bu kişiler projenin sonunda ortaya çıkacak ürünü kullanacak olan ve faydalarını doğrulayabilecek kimselerdir. Sprint gözden geçirme toplantılarına katılırlar.

4.3. Scrum Safhaları

Scrum şu safhalardan oluşur[5]:

- **Planlama**

  Scrum için en önemli safhadır. İlk olarak ürün gereksinim dökümanı geliştirilir. Sürüm ya da sürümlerin teslim tarihleri ve sürümlerin sağlayacakları yeni fonksiyonlar tanımlanır. Hemen geliştirilmesi en uygun sürüm bu safhada seçilir ve seçilen sürümdeki sürüm dokümanındaki maddelerle ürün paketleri eşleştirilir. Sürümü gerçekleştirecek takım tasarlanır, riskler değerlendirilir ve uygun risk kontrolleri sağlanır. Geliştirme araçları ve altyapı seçilir. Geliştirme,

pazarlama, eğitim gibi konular göz önünde bulundurularak sürüm maliyeti tahmini yapılır. Yönetim onayı ve mali kaynak sağlanır.

- **Mimari/Yüksek Seviye Tasarım**

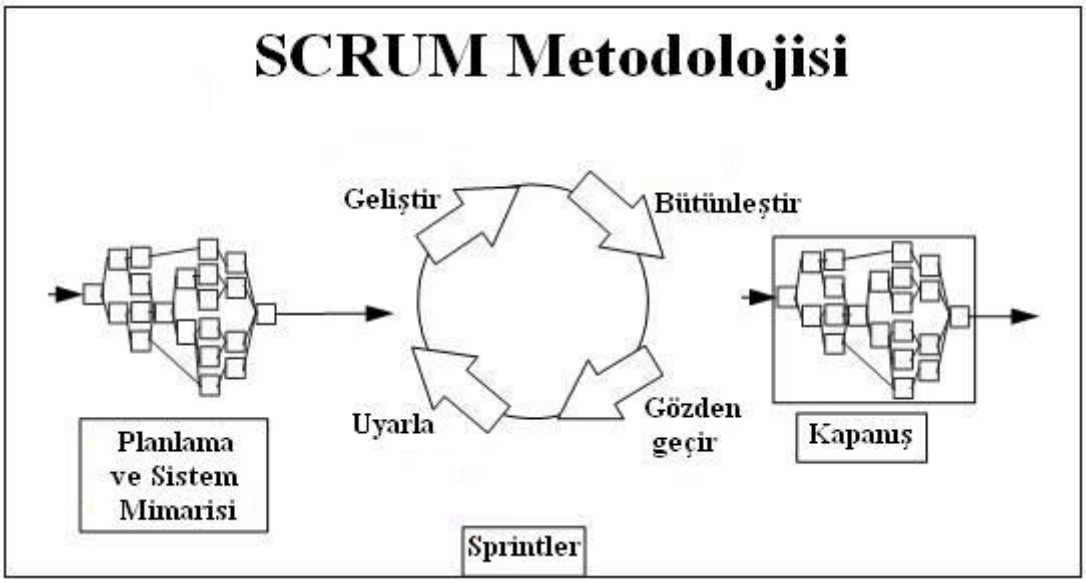
  Gereksinim dokümanında atanan işler gözden geçirilir. Böylece değişiklik ihtiyacı varsa belirlenir. Sistem mimarisi yeni içerikleri ve gereksinimleri destekleyecek şekilde düzenlenir. Değişikliklerin geliştirilmesinde ortaya çıkabilecek problemler tanımlanır. Gerekli ise işlerde yeniden atamalar yapılır.

- **Geliştirme (Sprint)**

  Geliştirme safhası, iteratif (yinelemeli) iş geliştirme döngüsü şeklindedir. Bu safhada takım toplantıları yapılarak sürüm planları gözden geçirilir. Ürünün uygun geliştirileceği standartlar için uyarlamalar yapılır. Ürün dağıtıma hazır olduğu düşünülene kadar iteratif sprintler devam eder. Yönetim ise ürünün süre, kalite ve fonksiyonalite açısından istenenleri karşılayıp karşılamadığını, iterasyonların tamamlandığını ve kapanış safhasının geldiğini belirler. Bu yaklaşıma Eşzamanlı Mühendislik (Concurrent Engineering) de denilir.

- **Kapanış**

  Yönetim geliştirme safhasının bittiğine karar verdiği zaman başlar. Bu safhada daha genel bir ürün sürümü oluşturulur. Entegrasyon, sistem testi, kullanıcı dokümanı, eğitim malzemeleri ve pazarlama malzemelerinin hazırlanması kapanış görevleridir.



Şekil 4.1 - Scrum safhaları

4.4. Scrum Süreci

Tüm rollerin aktif olarak görev aldıkları genel bir Scrum süreci şöyledir:

Projenin başlangıç adımı olarak yazılım gereksinimlerinin (requirements, features) ürün sahibi tarafından ürün gereksinim dokümanına yazılmasını düşünebiliriz. Ürün gereksinim dokümanı, Scrum'ın kalbidir[3]. Bu dokümanın sahibi ürün sahibidir ve gereksinimleri önceliklerine (priority) göre sıralar. Daha sonra takımla yapılan toplantılarda henüz net olmayan gereksinimler için geliştirme süresi tahminleri yapılır. Ayrıca ürün sahibi bu dokümana yazılım geliştirme süresince eklemeler ve çıkarmalar yapıp, öncelikleri değiştirme hakkına sahiptir. Böylece ürün sahibi değişen ihtiyaçlarına uygun olarak bir yazılıma sahip olma şansını yakalamış olur. Ama bunun öncesinde Scrum yöneticisi takım üyeleri ile birlikte takımın hızını ve Sprint uzunluğunu belirler. Sprint süresi en az 2 en fazla 4 hafta olarak belirlenir. İdeal olan proje boyunca tüm Sprint'ler aynı süreli olmasıdır. Scrum yöneticisi ayrıca takımın karşılaşacağı engel ve problemlerle ilgili Aksaklık Dokümanı'nı hazırlar.

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Şekil 4.2 - Ürün gereksinim dokümanı örneği (Eclipse)

Gereksinimler belirlendikten sonra yazılım geliştirme takımı Sprint planlama toplantısında bir sonraki Sprint'de geliştirilmek üzere ürün gereksinim dokümanından ürün sahibinin belirlediği yüksek öncelikli gereksinimleri seçerek Sprint dokümanına aktarırlar. Bu toplantıya Scrum yöneticisi, ürün sahibi ve takım üyeleri katılırlar.

Sprint planlama toplantılarını Scrum yöneticisi yönetir. Scrum yöneticisinin asıl görevi Scrum'ın temel prensiplerinin projeye uygulanmasını, bu prensiplerin takım üyelerince doğru şekilde anlaşılmasını sağlamaktır. En önemli görevi ise Sprint süresince takımı dışardan gelebilecek etkilere karşı korumak ve takımın ihtiyaçlarını karşılamaktır.

Scrum her Sprint'in sonunda mutlaka ürün sahibine kullanabileceği bir yazılım sağlamayı hedefler, bundan dolayı planlanan Sprint süresi (2-4 hafta) asla uzatılmaz. Fakat eğer bir gereksinim belirlenen Sprint süresi içerisinde gerçekleştirilemeyecekse

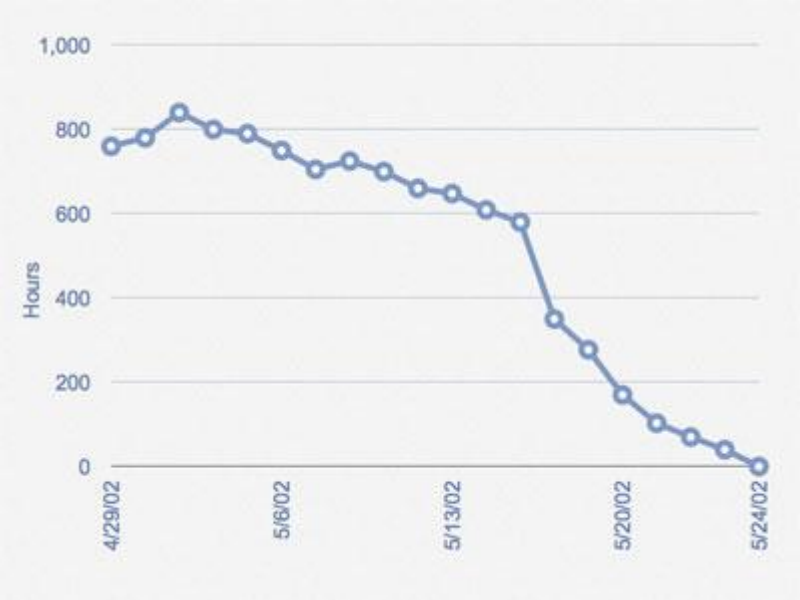
bir sonraki Sprint'e aktarılabilir. Ve aynı şekilde eğer Sprint süresi bitmeden Sprint dokümanındaki gereksinimlerin hepsi tamamlanmışsa ürün gereksinim dokümanından yeni gereksinimler Sprint dokümanına aktarılabilir.

Sprint planlama toplantısında belirlenen gereksinimler takım üyelerince küçük görevlere (tasks) bölünerek takım üyelerine geliştirilmek üzere atanır. Scrum takımı geleneksel yazılım geliştirme süreçlerinden farklı olarak kesin rollere (architect, tester, developer, designer vb.) sahip değildir. Scrum takımındaki bütün üyeler çapraz görevlerde yer alabilirler, böylece kodun tek bir kişiye bağımlılığı riski ortadan kaldırılmış olur. Sprint dokümanının sahibi bu sefer ürün sahibi değil yazılım geliştirme takımıdır, dolayısıyla bu dokümana ürün sahibi değil takım üyeleri katkıda bulunurlar.

Tasks	Mon	Tues	Wed	Thurs	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

Şekil 4.3 - Sprint dokümanı örneği

Sprint dokümanına aktarılan gereksinimlerin tahmini geliştirme süresi saat bazında takım üyelerince belirlenir ve Sprint boyunca sürekli olarak tahmini bu zamanlar güncellenerek Sprint kalan zaman grafikleri (burndown chart) oluşturulur. Böylece Sprint süresince ürün sahibi ve Scrum yöneticisi Sprint'in genel gidişi hakkında bilgi sahibi olur, aynı zamanda takım elemanları da kalan iş sürelerini ve harcadıkları zamanı takip edebilirler.



Şekil 4.4 - Sprint kalan zaman grafiği örneği

Scrum'ın belki de verimliliği artıran en önemli kavramlarından biri de Günlük Scrum toplantılarıdır. Bu toplantılar her gün belirli saatlerde farklı bir takım üyesinin liderliğinde ayaküstü yapılır ve en fazla 15 dakika sürer. Bu toplantılarda her takım üyesi şu 3 soruya cevap verir;

- Dün ne yaptım?
- Bugün ne yapacağım?
- Önümde olan engeller ve karşılaştığım sorunlar neler?

Bu toplantılara herkesin zamanında ve davet edilmeden katılması ve uzun sürmemesi çok önemlidir. Bu toplantılar sayesinde takım üyelerinin her biri diğer üyelerin nelerle uğraştığını öğrenme fırsatını edinirler ve çalışacakları işleri diğerleriyle paylaştıkları için işlerine daha iyi konsantre olabilirler. Scrum yöneticisi bu konuda takımı desteklemek ve diğer dış etkenlerle arayüz olmaktan sorumludur.

Şekil 4.5 - Günlük Scrum Toplantısı

Her Sprint'in bitiminde ortaya konulan ürün hakkında geri besleme alabilmek için yazılımla alakalı her türlü kişiye (ürün sahibi, pazarlama, diğer takımlar vs.) açık Sprint Gözden Geçirme Toplantısı yapılır. Bu toplantının amacı yazılımın ürün sahibinin gereksinimlerine uygun olarak geliştirildiğinden emin olmaktır. Bu sayede müşterinin gereksinimleri bir şekilde yanlış anlaşılmış ise bu fark edilir ve bir sonraki Sprint'de bu hataların önüne geçilir.

Daha sonra Scrum yöneticisi ve takım üyeleri bir araya gelerek Geriye Bakış Toplantısı düzenlerler. Bu toplantıda Sprint'te karşılaşılan problemler tartışılır. Sprint'in eksikleri ve iyi yönleri konusunda fikirler paylaşılır. Üretkenliği ve kaliteyi olumsuz etkileyen durumlar aksaklık olarak tanımlanır. Bu durumlar süreç ile ilgili olabileceği gibi (yetersiz toplantı süreleri, Sprint süresinin kısa olması, dokümantasyon yetersizlikleri vs.) teknik sorunlar da olabilir (yavaş internet bağlantısı, uzaktan erişim hakları vs). Tüm aksaklıklar önceliklendirilir ve çözülmek üzere Aksaklık Dokümanı'na eklenir.

Bu adımlar ürün sahibinin ürün gereksinim dokümanına yazdığı, zaman içinde geliştirip, değiştirdiği gereksinimler bitene kadar tekrarlanır. Böylece Scrum yönetimi ile ürün geliştirme sağlanır.

4.5. Scrum'ın Faydaları

Çevik süreçlerin planlama ve yönetimi konusunda etkili çözümler sunan Scrum'ın yazılım projelerine sağladığı en önemli katkılar şunlardır:

- Hızlı hayata geçirilen projeler
- Esnek alt yapı sayesinde kaliteli yazılımlar
- Maliyetlerin azaltılması
- Sürekli teslimat ile yüksek müşteri memnuniyeti
- Motivasyonu yüksek çalışanlar
- Kendi kendine organize olabilen takımlar

5.BÖLÜM

XP

5.1. XP (Extreme Programming) Nedir?

En popüler çevik süreçlerden birisi XP olarak bilinen Extreme Programming'dir. Kent Beck ve arkadaşları tarafından 1996 yılında Chrysler firmasında yapılan bir proje bünyesinde oluşan XP, içerdiği basit ama bir o kadar etkili yöntemlerle yazılım sektöründe yeni bir rüzgarın esmesini sağlamıştır.

XP ile oluşturulan çevik süreçte, müşteri ve gereksinimleri merkezi bir rol oynamaktadır. Yazılım esnasında XP ile tam belirli olmayan ve çabuk değişikliğe uğrayan müşteri gereksinimlerine ayak uydurulabilir. Bu geleneksel yazılım metotlarda mümkün değildir, çünkü proje öncesi müşteri gereksinimleri en son detayına kadar kâğıda dökülmesi gerekir. Bu şekilde oluşturulan bir dokümantasyon temel alınarak, yazılım gerçekleştirilir. Böyle bir projede zaman geçtikçe müşteri tarafından yapılması istenen değişikliklerin maliyeti daha yüksek olacaktır, çünkü mevcut yapı (tasarım) istenilen değişiklerin yapılmasını engelleyebilir ya da yeni bir yapılanmaya gidilmesini gerektirebilir. XP, kullanıldığı projelerde formalite ve bürokrasinin mümkün en az seviyeye çekilmesine önem verir. Çevik olabilmek için "az yükle yola çıkılması" gerekmektedir. Bu yüzden proje öncesi geniş çapta tasarım ve dokümantasyon oluşturulmasına izin verilmez. Bu kesinlikle dokümantasyon ve tasarım oluşturulmadan çalışıldığı anlamına gelmemektedir[23].

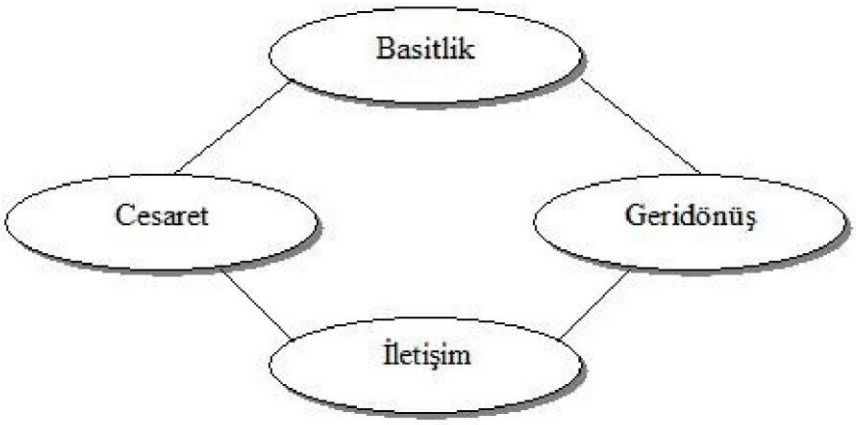
5.2. XP Değerleri

XP dört değer üzerine kuruludur:

- Basitlik (Simplicity)

- İletişim (Communication)
- Geri dönüş (Feedback)
- Cesaret (Courage).

XP'nin temel mantığı bu dört değer içinde saklıdır. Bu değerler uygulandığı takdirde XP'nin öğrenimi ve kullanımı kolaylaşır. Bu değerlerin geçerlilik bulmadığı ortamlarda XP'nin uygulandığı söylenemez. XP ile verimli bir çevik süreç oluşturabilmek için bu değerlerin hepsinin kabul görmesi ve uygulanması gerekmektedir.



Şekil 5.1 - XP'nin dört değeri

XP basit yöntemler aracılığıyla sonuca ulaşmak ister, çünkü sadece bu şekilde hızlı ve düşük maliyetli projeler gerçekleştirilebilir. Bunun yanı sıra basit çözümlerle oluşturulan programın bakımı ve geliştirilmesi de kolaydır. Basit çözümler kolay anlatılır ve adapte edilir. Bu, zaman kazanılması anlamına gelmektedir.

Yazılımda en önemli konulardan birisi kalite kontrolüdür. XP projelerinde kalite kontrolü daha çok geri besleme (feedback) üzerinden sağlanır. Programcılar, yazdıkları testlerden geri besleme alarak kaliteyi sağlarlar. Kısa zaman aralıklarıyla yeni sürüm oluşturularak müşteriye verilir, müşteri ve kullanıcılardan geri besleme aracılığıyla programın gereksinimleri karşılayıp, karşılamadığı kontrol edilir. Yazılım esnasında sürekli entegrasyon (bütünleştirme) yapılarak, programın en son durumu hakkında da geri besleme sağlanır. Bu nedenle XP'nin uygulanabilmesi için değişik katmanlarda geri besleme mekanizmalarının oluşturulmalıdır.

Tüm proje çalışanlarının aralarında sürekli olarak iletişim kurmaları gerekmektedir. Bireyler arası yüz yüze iletişim projenin başarısı için büyük önem teşkil etmektedir. Sadece bu sayede sağlıklı bilgi transferi gerçekleşebilir. Böylece yanlış anlaşılmalar ve bilinmeyenler ortadan kaldırılır. Eğer takım içinde iletişim ve işbirliği güçlü ise, dokümantasyon oluşturma ve kullanma gereksiz hale gelebilir. Bu zaman kaybını önler. Dokümantasyon yazılımı ve kullanımı başka sebeplerden dolayı gerekli olabilir, ama projenin başarısı için dokümantasyon öncelikli rol oynamamaktadır.

Basit çözümler, geri besleme ve iletişim için cesaret gereklidir. Bu saydığımız değerler bireyler arası işbirliği ve iletişimi artıracağı için, bireylerin kişisel olarak hareket etmeleri yerine takımın bir parçası olmalarını kolaylaştırırlar. Bu kişisel gelişimi de sağlar ve temelinde kişisel cesaret yatar.

5.3. XP Prensipleri

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

- **Hızlı geri besleme (Rapid Feedback)**

  Sık ve hızlı geri besleme edinmek, projenin gidişatını olumlu etkiler. Geri besleme sayesinde yanlış anlaşılmalar ve hatalar ortadan kaldırılır.

- **Basitliği tercih etmek** (**Assume Simplicity**)

  Basit çözümler kolayca geliştirilir ve kısa zamanda oluşturulur. Bu geri beslemenin de hızlı bir şekilde gerçekleşmesini sağlar. Basit çözümlerin kavranması ve anlatılması daha kolaydır. XP programcılardan o anki gereksinimi tatmin etmek için basit çözümü bekler. Programcı gelecekte oluşabilecek eklemeleri ve değişiklikleri düşünmemeli, sadece ve sadece kendisinden o an için bekleneni en basit haliyle geliştirmelidir.

- **Artırımlı değişiklik (Incremental Change**)

  Basit çözümler uygulasak bile, yazılım sistemleri zaman içinde karmaşık bir yapıya dönüşebilir. Yapılan en ufak bir değişiklik bile, sistemin düşünmediğimiz bir

bölümü üzerinde hata oluşmasına sebep olabilir. Oluşabilecek bu hataları kontrol altında tutabilmek için değişikliklerin ufak çapta olması gerekmektedir. Büyük değişiklikler beraberinde büyük sorunları getirebilir. Bu sebepten dolayı değişikliklerin ufak çapta ve sıklıkla yapılması gerekmektedir.

- **Değişimi istemek (Embracing Change)**

İlerleyebilmek için kendimize bir yön tayin etmemiz ve yeniliklere açık olmamız gerekiyor. Yeniliklere açık olmak cesaret gerektirir. Bilinmeyenle uğraşmak, rahatsız edici olabilir, ama başarıyı elde edebilmek için değişimi istemek gerekir.

- **Kaliteli iş (Quality Work)**

XP projelerinde kaliteli işin ortaya konabileceği bir ortamın oluşturulması gerekmektedir. Hiçbir programcı hatalı program yazmak istemez. Çalışma ortamının da etkisiyle yüksek kalitede yazılım yapmak hem programcının özgüvenini artırır, hem de müşteriyi tatmin edici ürünlerin ortaya konmasını sağlar.

- **Öğrenmeyi öğret (Teach Learning)**

XP programcı takımlarında tertipçilik ve kıdem farkı yoktur. Tecrübeli programcılar bilgilerini daha az tecrübeli programcılarla paylaşarak, hem bilginin çoğalmasını sağlarlar, hem de takım arkadaşları ile teknik olarak aynı seviyeye gelirler. Programcılara komutlar vererek iş yaptırmak yerine, kendiliğinden bazı şeyleri öğrenerek, görevlerini yerine getirmeleri sağlanmalıdır.

- **Az başlangıç yatırımı (Small Initial Investment)**

XP en modern ve pahalı araç gereçlerle projeye başlanmasını beklemez. Başlangıç giderleri ne kadar düşük tutulabilirse, projenin iptali durumunda kayıplar o oranda az olacaktır. Başlangıçta tüm takımın dar bir finansman korsasını giymesi sağlanarak, proje için daha önemli görevlere odaklanmaları sağlanır. Bunu bir örnekle açıklayabiliriz. Proje başlangıcında en son Oracle veritabanı ve kullanıcı araçları (Toad gibi bir client program) satın alınarak, tüm ekibe bu bilgi bankasını ve araçları nasıl kullanacaklarına dair seminer verilebilir. Bu çok masraflı ve bir o kadar

da gereksiz bir şeydir. Projeye açık kaynak (open source) ve ücretsiz olan mySQL, HSQL ya da PostgreSQL veritabanı ile başlanabilir. Programcı ekip böylece ne bir hafta tanımadıkları bir ürün için harcamış olurlar, ne de büyük masraflar yapılarak şu an için gerek olmayan bir altyapı bileşeni satın alınmış olur. Gerekli araçlar zamanı geldiğinde edinilmelidir.

- **Kazanmak için oyna (Play to win)**

XP takımları kazanmak için oynar. Her zaman gözlerinin önünde nihai sonuç vardır; programı tamamlamak ve müşteriye teslim etmek. XP programcı takıma tünelin sonundaki ışığı görmek için gerekli tüm imkanları sunar.

- **Somut denemeler (Concrete Experiments)**

Verdiğimiz kararların sonuçlarını kontrol edebilmek için denemeler yaparız, çünkü alınan kararlar her zaman doğru olmayabilir. Bir kontrol mekanizmasına ihtiyacımız vardır. Böylelikle nerede olduğumuzu tespit edebiliriz. Benzer somut denemeler yazılım sistemleri için de geçerlidir. Örneğin testler hazırlayarak, oluşturduğumuz mimari ve tasarımı kontrol edebiliriz.

- **Açık ve samimi iletişim (Open, honest Communication)**

Projenin başarılı olabilmesi için bireyler arasında açık ve samimi türde bir iletişimin olması gerekmektedir. Birçok projede bu böyle değildir. Çoğu zaman bireylerin korkuları, deneyimsiz olmaları yada kendilerini çok beğenmeleri ve diğerlerini kendilerinden alt safhada görmeleri, açık ve samimi bir iletişim ortamının oluşmasını engeller.

- **Takımın içgüdülerini kullan, onlara karşı koyma (Work with people's instincts, not against them)**

Bireysel içgüdünün yanı sıra, bireylerin oluşturduğu takımların da içgüdüsü vardır. Eğer takım bir şeylerin doğru gitmediği hissine sahipse ve bunu dile getiriyorsa, o zaman bir şeyler yolunda gitmiyor demektir. Takımın içgüdüsüne kulak verilmelidir. Bunun göz ardı edilmesi, proje için olumsuz sonuçlar doğurabilir.

- **Sorumluluk üstlenmek (Accepted Responsibility)**

Sorumluluk birilerine verilmemeli, bireyler kendileri sorumluluk üstlenmeliler. Eğer bir bireye ya da bir takıma yapılması zor bir projenin sorumluluğu yüklenirse, bu birey ya da takım için motivasyonun düşmesini ve kaybetme korkusunun pekişmesini hızlandırır. Eğer bireyler ya da takımlar kendi sorumluluklarını kendileri seçerlerse, hem yaptıkları işte kendilerini iyi hissederler, hem de yüksek motivasyon ile üstlendikleri işi başarıyla tamamlarlar.

- **Sürecin ortam şartlarına adapte edilmesi (Local Adaptations)**

Büyük bir ihtimalle her takımın XP'yi Kent Beck'in anlattığı tarzda harfiyen uygulaması mümkün değildir. Amaç XP'yi harfiyen uygulamak değildir, amaç kısa bir zamanda projeyi başarılı bir sonuca ulaştırmaktır. Eğer proje XP'de yapılacak modifikasyonlarla başarıya ulaşacaksa, o zaman süreç üzerinde bu modifikasyonlar yapılmalı ve uygulanmalıdır. Bunda bir sakınca yoktur.

- **Az yükle yolculuk yapmak (Travel light)**

Projede hızlı ilerleyebilmek için fazla bir yükle yola çıkılmaması gerekmektedir. Beraber çalışmayı kolaylaştırmak için kullanımı kolay araç ve gereçler seçilmelidir. Formalitelerden uzak durulmalıdır.

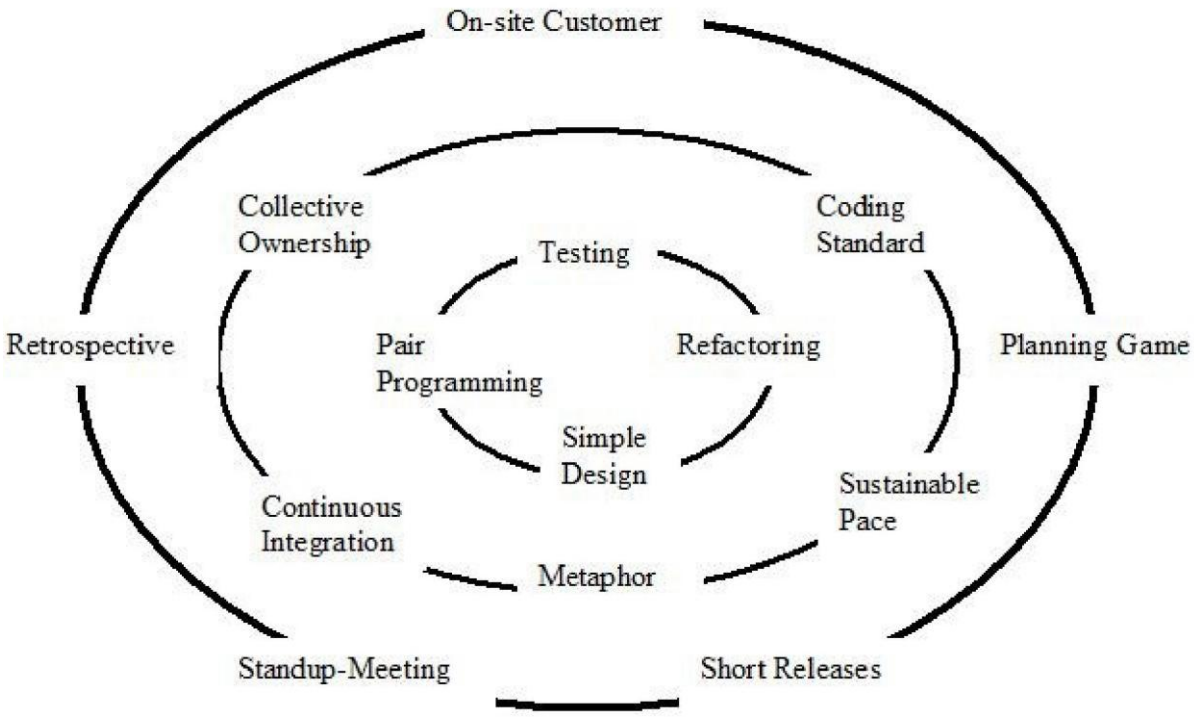
- **Doğru ölçme (Honest Measurement)**

Proje gidişatını kontrol edebilmek için değişik türde ölçümlerin yapılması gerekmektedir. Örneğin hazırlanan birim (unit) testleri ile sınıfların işlevleri kontrol edilir. Yapılan ölçümler doğru ve samimi yapıldığı takdirde kontrol mekanizması olarak kullanılan ölçümlerin bir anlamı vardır. Programcılar tarafından samimi ve doğru yapılmayan ölçümler projenin gidişatını olumsuz yönde etkiler.

5.4. XP Pratikleri

Dört XP değeri ve on beş XP prensibi, on dört XP pratiği ile desteklenmektedir. XP pratikleri programcıların XP değer ve prensiplerini uygulamada yardımcı olur. Kent

Beck tarafından hazırlanan ilk XP versiyonunda on iki teknik yer almaktaydı. Diğer çevik süreçlerin de etkisiyle ayaküstü ve retrospektif toplantılar XP pratikleri arasına katıldı.



Şekil 5.2 - XP pratikleri

- **Programcıya yakın müşteri (On-site Customer)**

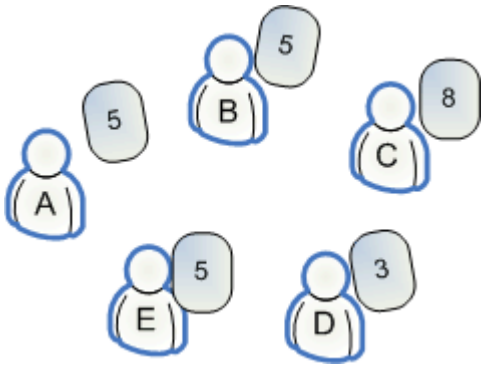
XP projeleri müşteri gereksinimlerine odaklı ilerler. Bu yüzden müşteri ve sistem kullanıcılarının projeye dahil edilmeleri gerekmektedir. Müşteri gereksinimlerini ekibe bildirir. Programcıların geliştirmeyi gerçekleştirebilmesi için müşteri tarafından dile getirilen gereksinimleri anlamaları gerekmektedir. Yanlış anlaşılmaları ve hataları gidermek için programcıların müşteri ve sistem kullanıcıları ile diyalog halinde olabilmesi gerekmektedir. Bu sebepten dolayı müşteri veya sistem kullanıcılarının programcıların erişebileceği bir uzaklıkta olmaları gerekir. Tipik XP projelerinde müşteri ve programcılar aynı odada beraber çalışırlar. Müşteri ekibin sorularını zaman kaybı olmadan cevaplar ve projenin ilerlemesine katkıda bulunur.

- **Ayakta toplantı (Standup-Meeting)**

Proje çalışanları her gün 15 dakikayı aşmayan ve ayakta yapılan toplantılarda bir araya gelirler. Bu toplantının amacı, projenin gidişatı hakkında bilgi alışverişinde bulunmaktır.

- **Planlama oyunu (Planning Game)**

XP projeleri yinelemeli ve artırımlı yol alır. Bir sonraki iterasyonda yapılması gereken işleri planlama oyununda görüşülür ve sürüm ve iterasyonun içeriği tespit edilir. Planlama oyununa müşteri, kullanıcılar ve programcılar katılır. Müşteri ve kullanıcılar daha önce kullanıcı hikayesine (user story) dönüştürdükleri isteklerine öncelik sırası verirler. Bu hikayelerden ortaya ilgili görevler çıkartılır. Görevler tüm takım üyeleri tarafından anlaşıldıktan sonra programcılar her görev için gerekli zamanı tahmin ederler. Görevlerin öncelik sırası bu tahmine bağımlı olarak değişebilir. Herkes tahminlerini boş kartlara yazdıktan sonra aynı anda açarak birbirlerine gösterirler.



Şekil 5.3 - Planlama oyunu (Poker)

En düşük ve en yüksek tahminde bulunanlar tahminlerini açıklarlar. Böylece diğer takım üyelerinin düşünemediği noktalar açığa çıkabilir. Bu noktada takım birkaç dakika tartışır. Müşteri ya da ürün sahibi net olmayan yerleri açıklar. Hatta yeni kullanıcı hikayeleri ortaya çıkabilir. Daha sonra tekrar takım üyeleri tahmin kartlarına yeni tahminlerini yazarlar ve tekrar birbirlerine gösterirler. Genellikle 2. tekrarda tahminler aynı çıkar ama yine de değişik tahminler varsa bir önceki tahminlemede yapıldığı gibi en yüksek ve en düşük tahmin edenler tekrar açıklama yapar. Bu işlemler takım üyeleri hemfikir olana kadar tekrar eder. Nadiren 3

tekrardan fazla olur. Örneğin, 2. Tekrarda 5 5 5 3 gibi tahminler yapılmışsa 3 tahmin edene 5 olmasının uygun olup olmadığı sorularak zaman kazanılabilir. Planlama oyununda kullanılan kartlar boş olabileceği gibi tahminlemelerdeki farklılıkları aza indirmek için üzerinde Fibonacci serisinin sayıları (0,1,1,2,3,5,8,13,21…) bulunan kartlarda kullanılabilir. Planlama oyunları sonunda sürüm ve iterasyon planları oluşur.

- **Kısa aralıklarla yeni sürüm (Short Releases)**

XP projelerinde yeni geliştirilen ve değişikliğe uğrayan bileşenler yeni sürümler oluşturularak müşteri ve kullanıcının beğenisine sunulur. Bu sayede hem müşteriler çalışır durumda olan programdan faydalanabilir hem de yeni sürümü inceleyerek, gereksinimleri ile örtüşüp, örtüşmediğini kontrol edebilirler. Eğer yeni sürüm müşteriyi tatmin edecek durumda değilse, gereksinimler değişikliğe uğrayabilir. Bu değişiklikler bir sonraki iterasyonda göz önünde bulundurularak, müşteri istekleri ile yüksek derecede örtüşen bir programın oluşturulması sağlanır.

- **Geriye bakış (Retrospective)**

Proje çalışanları düzenli aralıklarla geçmişe bakarak, meydana gelen sorunları gözden geçirirler. Buradaki amaç gelecekte bu sorunların tekrarını önlemektir. Geriye bakış bir ila altı zaman birimleri için tüm proje çalışanları yada seçilen bireyler tarafından yapılır. Geriye bakış toplantıları yarım gün ila üç gün arasında sürebilir.

- **Mecaz (Metaphor)**

XP projelerinde hazırlanan program için bir veya birden fazla, programın nasıl bir işlevi olacağını ekibin gözünde canlandırmalarını sağlayacak mecazi isim, öğe yada resimler kullanılır. Bunlar proje çalışanlarının ortak bir payda da buluşarak, ne yapılması gerektiği hakkında bir fikir sahibi olmalarını kolaylaştırır. Örneğin bir alışveriş sistemi yazılımı yapılacak. Burada metafor olarak alışveriş sepeti kullanılabilir. Alışveriş sepetini duyan her programcının aklında, bir shop sisteminin programlanması gerektiği fikri doğar.

- **Ortak sorumluluk (Collective Ownership)**

XP projelerinde programcılar ortak sorumluluk taşırlar. Bu her kod parçasının herhangi bir programcı tarafından gerekli durumlarda değiştirilebileceği anlamına

gelir. Böylece yapılması gereken işler aksamaz, çünkü belli kod bölümlerinden belli programcılar sorunlu değildir. Aksine her programcı programın her bölümü üzerinde çalışma hakkına sahiptir. Bir programcının işe gelmemesi durumunda, başka bir programcı kolaylıkla onun görevlerini üstlenebilir.

- **Sürekli entegrasyon (Continuous Integration)**

Sistem değişiklikleri ve yeni bileşenler hemen sisteme entegre edilerek test edilir. Sürekli entegrasyon sayesinde yapılan tüm değişiklikler her programcının sistem üzerinde yapılan değişiklikleri görmesini sağlar. Ayrıca sistem entegrasyonu için gerekli zaman azaltılır, çünkü oluşabilecek hatalar erken teşhis edilerek, ortadan kaldırılır.

- **Kodlama standartları (Coding Standards)**

Programcılar tarafından aynı kalitede kod yazılımı yapılabilmesi için, kod yazarken kullanılacak kuralların oluşturulması gerekmektedir. Kodun nasıl formatlanacağı, sınıfların, metot isimlerinin ve değişkenlerin nasıl isimlendirileceği kod standartlarında yer alır.

- **Kalıcı tempo (Sustainable Pace)**

XP projelerinde programcılar haftalık belirli mesai saatlerini aşmazlar. Gereğinden fazla çalıştırılan ve yorulan bir programcıdan verimli iş yapması beklenemez. Programcıların motivasyonun ve çalışma enerjilerinin yüksek olması için günde sekiz saatten fazla çalışmalarına izin verilmemelidir. Bazen fazla mesai saatlerine ihtiyaç olabilir. Eğer durum devamlı böyle ise, bu proje gidişatında bazı olumsuzlukların göstergesi olabilir.

- **Test etmek (Testing)**

Oluşturulan programların kalite kontrolünden geçmesi gerekmektedir. Bu yazılım geliştirme esnasında oluşturulan testlerle yapılır. Programcılar bileşenler için birim(unit) testleri hazırlar. Sınıf bazında yapılan bu testlerle bileşenlerin işlevleri kontrol edilir. Müşteri gereksinimlerini test etmek için kabul testleri hazırlanır. Bileşenlerin entegrasyonunu test etmek için ise entegrasyon testleri hazırlanır. Bunun yanı sıra performans testleri ve stabilite testleri ile yazılımın gücü test edilebilir.

- **Sade tasarım (Simple Design)**

Programcılar üstlendikleri görevleri (task) en basit haliyle geliştirirler. Bu programın basit bir yapıda kalmasını ve ilerde değiştirilebilir ve genişletilebilir olmasını sağlar. Sade bir tasarım yazılım sisteminin kompleks bir yapıda olmasını önler. Bunun yanısıra basit tasarımlar daha kolay ve daha hızlı geliştirilebilir. Basit bir geliştirmeyi anlamak ve anlatmak daha kolaydır.

- **Yeniden yapılandırma (Refactoring)**

Tasarım hataları yazılım sisteminin daha ilerde tamir edilemeyecek bir hale dönüşmesine sebep olabilir. Bu yüzden bu hatalar hemen giderilir. Bu yeniden yapılandırma işlemine "refactoring" ismi verilir. Hazırlanan birim testleri ile yapılan değişikliklerin yan etkileri kontrol edilir. Bu açıdan bakıldığında birim testi olmayan bir sistem üzerinde yeniden yapılandırma işlemi hemen hemen mümkün değildir, çünkü değişikliklerin doğurduğu yan etkileri tespit etme mekanizması bulunmamaktadır.

- **Eşli programlama (Pair Programming)**

Eşli programlama, bir bilgisayarda iki programcının birlikte çalışmasını benimseyen bir yazılım geliştirme tekniğidir. Biri(sürücü) kodlama yaparken diğeri(gözlemci) yazılan kodun her satırının anında gözden geçirmesini yapar. Kod gözden geçirilirken, gözlemci işin gidişatını dikkate alır, ona göre uygun iyileştirmeler ve olası problemleri belirler. Gözlemci konumundaki programcı bir emniyet şeridi veya rehber gibi hareket ederek, sürücü konumundaki programcının yapılan işin sadece taktik tarafına yoğunlaşmasını sağlar. Bu yapılırken iki programcı sık sık rolleri değişirler. Bu sayede programcıların kısa bir zaman içinde aynı seviyeye gelmesi sağlanır. Bu teknik ayrıca yazılımın kalitesinin de yükselmesini sağlar. Ama bazı tekniklere bağlı kalınarak uygulanması gereken ve yapılmadığı zaman tam anlamıyla uyulama geliştiricisi için bir kâbusa dönüşebilen bir tekniktir. Eşli programlama ile yazılım geliştirmek isteyenler ilk başlarda birbirlerini anlamakta zorlanabilirler. Bu da yazılım gereksinimlerin hızlı gerçekleştirilememesine ve sürenin uzamasına sebep olur. Zamanın ilerlemesiyle birlikte yazılımcılar birbirlerini daha iyi anlamaya başlarlar ve işlerin vaktinden önce yapılmasını da sağlayabilirler.

Eşli programlama projenin tamamında ya da yazılımdaki hataların giderimi, yazılım güvenliği, modül birleşim kısımları gibi bazı önemli alt bölümlerinde kullanılabilir. Kolay anlaşılır, ezbere geliştirilebilir kısımlar tek başına yapılabilir.

5.5. XP Rolleri

Bir XP çevik projede ekip çalışanlarının sorumluluk alanlarını tanımlamak için roller tayin edilir. Her rol beraberinde bazı sorumluluklar ve tanımlanmış haklar getirir. Bu roller statik değildir. Ekip içinde değişik kişilere değişik roller verilebilir ve daha sonra rol değişikliği yapılabilir. Proje gereksinimleri doğrultusunda yeni rollerin oluşturulması mümkündür.

- **Müşteri (Customer)**

Projenin var olma sebebi müşteridir. Müşteri ihtiyaç duyduğu ve gereksinimlerine cevap verebilecek bir yazılım sistemi için yatırım yapan kişidir. Proje bünyesinde ne programlanması gerektiğini müşteri tayin eder. Müşteri yapılması gerekenleri kullanıcı hikayeleri (user story) oluşturarak ifade eder. Programcılar müşteriye bu süreçte yardımcı olurlar. Ama kullanıcı hikayelerinin oluşturulma sorumluluğu büyük ölçüde müşteriye aittir. Her kullanıcı hikayesi yazılım sisteminin bir özelliğini tanımlar. Programcıların geliştirme yapabilmeleri için kullanıcı hikayesini anlayabilmeleri gerekmektedir. Sadece bu durumda geliştirme süreci için bir tahminde bulunabilirler. Ayrıca oluşturulan kullanıcı hikayelerinin test edilebilir yapıda olması ve ilgili kriterler içermesi gerekmektedir.

Müşteri, etki alanı hakkında bilgiye (domain knowledge) sahip olan kişidir. Programcılar müşteriye karşılaştıkları sorunları çözmek için sorular sorabilirler. Bu soruların cevabını en iyi verebilecek şahıs müşteridir.

Hangi kullanıcı hikayelerinin önce geliştirileceğine müşteri karar verir. Bu konuda müşteriye herhangi bir sınırlama getirilmez. Müşteri seçer ve programcılar tasarlar ve kodlar.

Geliştirilen kullanıcı hikayelerini kontrol etmek amacıyla müşteri kabul testleri tanımlanır. Bu testler programcı yada testçi tarafından gerçekleştirilir. Kabul testleri kullanıcı hikayesinin doğru geliştirilip, geliştirilmediğini kontrol edici bir mekanizmadır.

- **Programcı**

Sistem analizi, tasarım, test ve geliştirme programcılar tarafından yapılır. Müşteri tarafından hazırlanan kullanıcı hikayelerinin geliştirme süresi programcılar tarafından tahmin edilir. Bu, programcıların XP projelerinde proje planlama sürecine dahil edildikleri anlamına gelmektedir. Geleneksel projelerde bu tahminleri teknik bilgiye sahip olmayan yöneticiler yapmak zorundadır. Bu yüzden bu tahminler genelde gerçekleri yansıtmaz.

Her programcı test güdümlü (TDD - Test Driven Development) ve bir takım arkadaşıyla beraber çalışır. "Pair programming" olarak bilinen, iki programcının birlikte yazılım yapması, kısa zamanda kod hakkındaki bilginin tüm programcılar tarafından paylaşılmasını kolaylaştırır. Ayrıca eşli programlama programcılar arasında iletişimi ve takım içinde çalışabilme özelliğini artırır. Test güdümlü çalışmak bakımı ve geliştirilmesi kolay kodun oluşmasını sağlar. XP projelerinde programcılar herhangi bir satır kod yazmadan önce gerekli test sınıflarını oluşturarak geliştirmeye başlarlar (TDD).

Programcılar oluşturdukları testler yardımıyla her gün bir veya birden fazla şekilde modül entegrasyonu gerçekleştirirler. Sürekli entegrasyon programcılar tarafından oluşturulan modülleri entegre eden bir süreçtir. Her programcı bu süreçten geri besleme sağlayarak, kendi yaptıklarının ne derecede sisteme entegre olduğunu ölçebilir.

- **Proje yöneticisi**

Proje yöneticisi müşteri ve programcıları bir araya getirir. Onların beraber çalışabilecekleri ortamların oluşmasını sağlar. XP proje yöneticisi tek başına proje planlamasından sorumlu değildir. Programcılara görev atamaz, onların kendi

başlarına seçim yaparak, sorumluluk almalarını kolaylaştırır. Toplantı ve diğer buluşmaları koordine eder, takımın karşılaştığı sorunları ortadan kaldırmak için gerekli olanları yapar.

- **Koç**

Çevik süreci tanıyan ve nasıl uygulanması gerektiğini bilen uzmandır. Koçun görevi proje başlangıcında çevik takımı oluşturmak yada bir araya getirmek ve onlara belirli bir süre rehberlik yapmaktır. Koç projede sorun çıktığı zaman yada takım XP yöntemlerinin dışına çıktığında müdahale eder. Bazı zamanlarda koç geliştirme işinde aktif olarak rol alır. Örneğin birim testlerin nasıl doğru bir yapıda oluşturulabileceğini diğer programcılara gösterebilir.

- **Testçi**

Müşteri tarafından oluşturulan kabul testlerini geliştiren ve gerçekleştiren programcıdır. Aynı zamanda JUnit ve entegrasyon testlerinin geliştirilmesinde takım arkadaşlarına yardımcı olur.

5.6. XP'de Haklar ve Sorumluluklar

Proje çalışanları çoğu zaman içgüdüsel korku hissedebilirler. Bunun en büyük sebebi belirsizliktir. Ne ve nasıl yapılması gerektiği bilinmediği takdirde ekip içinde huzursuzluk doğar. Bu projenin başarısını negatif etkiler.

Projenin başarılı olabilmesi için çalışanların hissettikleri korkunun aza indirilmesi yada yok edilmesi gerekmektedir. XP bu konuda proje çalışanlarına bir takım hakların tanınması gerektiğini dikte eder. Hangi rollerin hangi haklara sahip olduğunu yakından inceleyelim

.

- **Müşteri Hakları**

Müşterinin sahip olduğu haklar şu şekilde özetlenebilir:

1. Müşteri bütçe ve zaman planlaması yapabilmek için neyin yapılabilir olduğunu ve hangi zaman biriminde yapılabileceğini bilme hakkına sahiptir.
2. Müşteri fikir değiştirerek, gereksinimler üzerinde değişiklik yapma ve yeni gereksinimlerin geliştirilmesini talep etme hakkına sahiptir.
3. Müşteri programcılar tarafından sağlanabilecek en yüksek verimi ve değeri elde etme hakkına sahiptir.
4. Müşteri projede somut ilerlemeyi görme hakkına sahiptir. Bu kısa aralıklarla oluşturulan yeni sürüm ve kabul testlerine olumlu cevap veren yeni implementasyonlarla sağlanır.
5. Müşteri bir sonraki sürümde geliştirilecek kullanıcı hikayelerini seçme hakkına da sahiptir.

- **Programcı Hakları**

Programcının sahip olduğu haklar şu şekilde özetlenebilir:

1. Programcı takım arkadaşlarına soru sorma ve cevap alma hakkına sahiptir.
2. Programcı kullanıcı hikayeleri için geliştirme zaman tahmini yapma hakkına sahiptir. Programcı tahminler üzerinde değişiklik yapma hakkında sahiptir.
3. Programcı, kendisine görev verilmesi yerine sorumluluk alma hakkında sahiptir.
4. Programcının her zaman yüksek kalitede iş çıkarma hakkı vardır.
5. Programcının hangi öncelik sırasına göre neyi yapması gerektiğini bilme hakkı vardır.

5.7. XP Safhaları

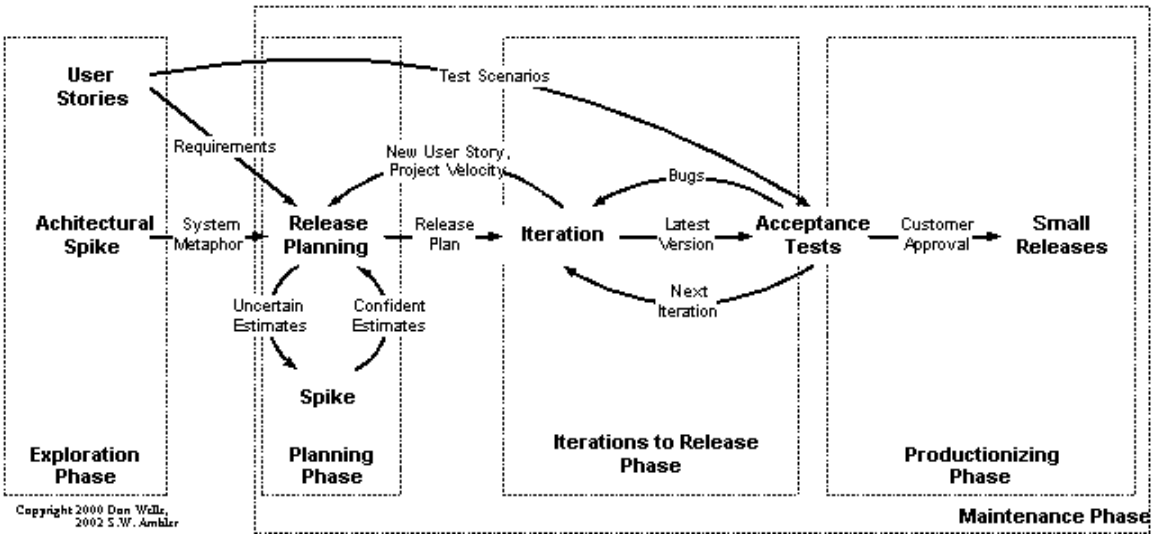
Bir XP projesi değişik safhalardan oluşur. Her safha, bünyesinde kendine has aktiviteler içerir. Bunlar:

- Keşif safhası (Exploration Phase): Projenin başlangıcında keşif safhasını oluşturan aktiviteler yer alır. Bu safhada müşteri kullanıcı hikayelerini (user story) oluşturur. Programcılar teknik altyapı için gerekli deney (spike) ve araştırmayı yaparlar.

- Planlama Safhası (Planning Phase): Keşif safhasını planlama safhası takip eder. Bu safhada müşteri programcılar yardımıyla iterasyon ve sürüm planlarını oluşturur. İterasyon planlaması için oluşturulan kullanıcı hikayelerinin geliştirme süresi programcılar tarafından tahmin edilir. Müşteri kullanıcı hikayelerine öncelik sırası vererek, iterasyonlarda hangi kullanıcı hikayelerinin öncelikli olarak geliştirilmeleri gerektiğini tespit eder. Programcılar tarafından herhangi bir kullanıcı hikayesinin geliştirme süresi tahmin edilemezse, programcılar "spike solution" olarak bilinen basit bir çözüm oluşturarak, kullanıcı hikayesinin gerçek geliştirilmesi için gerekli zamanı tahmin etmeye çalışırlar.

- İterasyon ve Sürüm Safhası (Iterations to Release Phase): Kullanıcı hikayelerinin geliştirilmesi, iterasyon ve sürüm safhasında gerçekleşir. Öncelikle bir iterasyon bünyesinde geliştirilmesi gereken kullanıcı hikayeleri müşteri tarafından belirlenir. Bu geliştirmenin doğruluğunu kontrol etmek için müşteri tarafından kabul testleri oluşturulur. Bu testler programcılar ya da testçiler tarafından kodlanır ve sisteme aktarılır. Bu şekilde müşterinin sistem hakkındaki görüşleri alınır (geri besleme). İterasyon son bulduktan sonra çalışma hızını tahmin etmek için bir önceki iterasyonunda elde edilen tecrübeler kullanılır ve iterasyon planı bu değerler doğrultusunda gözden geçirilir. Böylece her iterasyon sonunda müşteriye, çalışır bir yazılım sistemi sunulur. Bir önceki iterasyonda oluşan hatalar bir sonraki iterasyonda gözden geçirilmek ve giderilmek üzere planlanır.

- Bakım Safhası (Maintenance Phase): Bu programın bakımının ve geliştirilmesinin yapıldığı safhadır. Bu safhada kullanıcılar için eğitim seminerleri hazırlanır ve küçük çapta eklemeler ve sistem hatalarının giderilmesi için işlemler yapılır. Müşterinin istekleri doğrultusunda bir sonraki büyük sürüm için çalışmalara başlanır. Bu durumda tekrar keşif safhasına geri dönülmesi ve oradan işe başlanması gerekmektedir.



Şekil 5.4 - XP proje safhaları

5.8. XP Süreci

XP projelerinde projenin gidişatını genel olarak şu şekilde özetleyebiliriz:

- Müşteri gereksinimleri ihtiva eden kullanıcı hikayelerini oluşturur. Bunlara öncelik
  sırası atar. Programcılar her kullanıcı hikayesi için geliştirme zamanını tahmin ederler.
- Müşteri programcılarla beraber iterasyon(1-2 hafta) ve sürüm (1-2 ay) planını hazırlar. Her sürüm, birden fazla iterasyon içerir ve müşteri tarafından kullanılacak özellikte çalışır bir yazılımdır.
- Müşteri ilk iterasyon (1 hafta) için gerekli kullanıcı hikayelerini tahminleri de dikkate alarak seçer.
- Programcılar iterasyon için seçilen kullanıcı hikayelerini geliştirirler. Kafalarda oluşan sorular müşteriye danışılarak cevaplandırılır.
- İterasyon sonunda programcılar müşteriye çalışır bir sistem sunarlar. Müşteri sistemi
  değerlendirerek programcılara geri besleme sağlar.
- Edinilen tecrübeler ışığında bir sonraki iterasyon planlanır. Eğer müşteri yeni gereksinimlerin geliştirilmesini isterse, bunlar için tekrar kullanıcı hikayeleri oluşturulur ve tahminler yapılır. Eğer yeni kullanıcı hikayeleri yoksa, mevcut

kullanıcı hikaye listesinden en yüksek öncelik sırasına sahip olanlar seçilir ve bir sonraki iterasyondan geliştirilir.

- İlk sürüm sonunda oluşan yazılım sistemi müşteri tarafından ürün olarak kullanıma alınır. Başka sürümler planlandıysa bir sonraki iterasyonla devam edilir.
- Tüm sürümler sonunda istenen yazılım ürünü elde edilir.

6.BÖLÜM

SCRUM & XP

Scrum ile XP'nin birlikte çalışabilirliği en yaygın kullanılan çevik geliştirme çözümüdür. Scrum organizasyon ve yönetim işlerinde yoğunlaşırken XP ise programlama pratikleri sunar. Yani ikisi de farklı çevik model pratiklerini karşılarlar ve böylece birbirlerini tamamlarlar. Hatta "Tüm takım", "Birlikte oturma", "Hikayeler", "Oyun planlama" gibi bazı XP pratikleri Scrum pratikleri ile örtüşür ve çevik geliştirme süreci pekişir.

Çevik manifesto'nun mimarlarından Jeff Sutherland ve Ron Jeffries; Scrum olmadan XP takımlarını ölçeklendirmenin oldukça zor olduğunu söyler. Onlara göre Scrum, XP için bir yönetim ara yüzü sağlar. Yine çevik manifestonun mimarlarından Robert C. Martin ise bu uyumlu ikiliyi "markasız çevik süreç(unbranded agile)" olarak tanımlar.

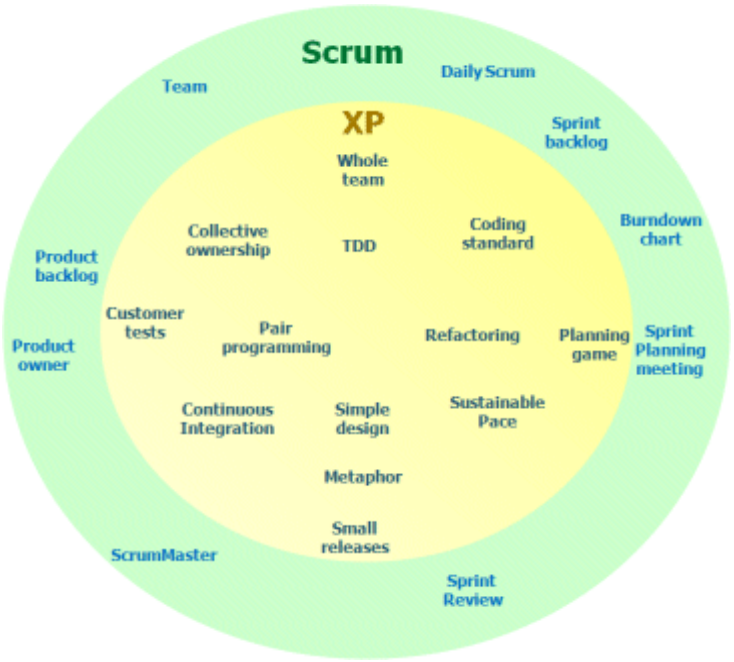
2008 yılında yapılan 2.Yıllık Çevik Geliştirme Araştırması (2nd Annual State of Agile Development Survey) istatistiklerine göre kullanılan Çevik geliştirme yöntem oranları şöyledir.

Tablo 6.1 - Çevik yöntemlerin kullanım oranları

Çevik yöntemler	**Kullanım oranları**
Scrum	%49.1
Scrum/XP Hybrid	%22.3
Extreme Programming (XP)	%8.0
Custom/Hybrid	%5.3
Don’t Know	%3.7
Agile Unified Process (AgileUP)	%2.2

Other	%2.2
Feature-Driven Development (FDD)	%2.1
Lean Development	%1.9
OpenUP	%0.6
Agile Modeling	%0.6
Crystal	%0.5
Dynamic Systems Development Method (DSDM)	%1.4

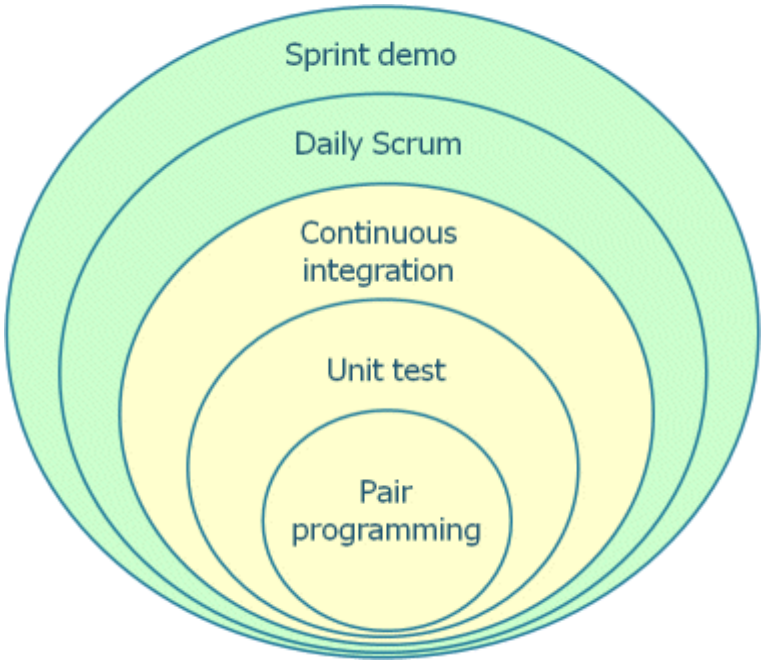
İstatistiklerde görülen mesaj açıktır. Scrum ve XP birlikte kullanım oranının XP kullanımının çok üstünde olduğu görülmektedir. Bu oranın gelecek yıllarda daha da artacağı düşünülüyor. Aşağıdaki şekil ise bu ikilinin uyumluluğunu gösteriyor.(Sarı= XP, Yeşil = Scrum) Scrum'ın takım ile paydaşlar arasında bir arayüz olduğu açık şekilde görülmektedir. Scrum, takıma günlük işlerini XP pratiklerine uygun yapmalarına imkan sağlayacak bir alan sunmaktadır. Scrum'ın tesis edildiği bir projede takım özyönetimli (self-managing) olur ve işi nasıl yapmak istediğine takım karar verebilir. Scrum içine yerleştirilen gereksinimlerle (her iterasyonda test edilmiş çalışan yazılım teslimatı gibi) XP doğal bir çözüm halini alır.



Şekil 6.1 - Scrum ve XP uyumluluğu

Çevik yazılım geliştirme geri besleme döngülerinden oluşur. Scrum ve XP bu döngüde birbirini çok iyi şekilde tamamlayabilirler. Aşağıdaki şekil bunu göstermektedir. (Sarı= XP, Yeşil = Scrum)

Şekilde küçük daire kısa geri besleme döngüsü anlamına gelmektedir. Scrum ve XP'nin günlük toplantılarında ufak bir üst üste binme oluşsa da önemsenmemektedir.



Şekil 6.2 - Scrum ve XP geri besleme döngüleri

Bir seviyede, eşli programlama bir kaç saniye içinde bir geri besleme sağlamakta iken diğer bir seviyede Sprint sunumlar birkaç haftada geri besleme sağlayabilmektedir. Ama her ikisi de "biz işi doğru yapıyor muyuz?" sorusunu sorduran geri besleme döngüleridir.

Bunun yanı sıra XP pratikleri, Scrum planlama ve yönetim pratikleri sayesinde daha disiplinli hale gelir ve çevik olmayan standart modeller içinde bile rahatça uygulanabilir.

7. BÖLÜM

CMMI VE ÇEVİK GELİŞTİRME KARŞILAŞTIRMASI

7.1. CMMI ve Çevik Geliştirme Neden Uyumsuz?

Çevik geliştirme yöntemleri ve CMMI pratiklerinin uyumsuz olduğu şeklinde genel bir algılayış vardır. Çevik yöntemlerin küçük ve doküman gerektirmeyen projelerde, CMMI'ın ise büyük ve dokümantasyonun önemli olduğu projelerde kullanılması gerektiği bunların en başında yer alır.

Peki neden Çevik Geliştirme ve CCMI bu kadar ayrıktır?

SEI tarafından 2008 yılında yayınlanan "CMMI veya Çevik yöntemler: Neden birbirini kucaklamaz?" raporunda[29] bu ayrıma sebep olan iki ana sebep belirtilmiştir:

1. CMMI ve Çevik modeli ilk benimseyenler yazılım geliştirme modellerinin uç örneklerini temsil ettiler. CMMI benimseyenler büyük ölçekli, risk kaldırmayan, kritik amaçlı sistemlerin geliştiricileri idiler. Hatta bu sistemler sık sık yüksek seviyede yönetici gözetimi ve hiyerarşik yönetim altında geliştirilirdi. Fakat çevik yöntemleri ilk benimseyenler ise genelde daha küçük, tek takımla geliştirilebilen, zamanla değişebilen ihtiyaçları olan projelere yoğunlaştılar. Bu uç örnekler ayrımın oluşmasını sağladılar.

2. CMMI ve Çevik model hakkındaki yanlış bilgiler ve ikisinin yanlış kullanımı her iki tarafta da yanlış algılara yol açtı. CMMI ve Çevik modeli birbirinden uzak tutan bu yanlış algıların en önemli sebepleri şunlardır:

    - Yanlış kullanım

    CMM ile başlayıp CMMI ile devam eden 20 yıllık deneyim boyunca bazı pratikler ya yanlış kullanıldı ya da yazılım geliştirme takımının bazı pratikler olmadan

yapamayacağı ve yazılımın bu pratiklerle daha üretken olacakları düşünülerek geliştirme aktivitelerinde üstü üste bindirilerek uygulandı.

- Doğru bilgi eksikliği

Çevik toplulukta CMMI hakkında doğru bilginin yokluğu ve aynı şekilde CMMI topluluğunda Çevik yöntemler hakkında doğru bilginin yokluğu bu ayrıklığı körükledi.

- Terminoloji Zorlukları

CMMI'da (örneğin disiplin, öngörülebilme nitelik güvencesi,) ve Çevik yöntemlerde (sürekli entegrasyon, kolektif kod sahiplenmesi, test güdümlü geliştirme) kullanılan terminolojiler konuya özel yan anlamlar taşırlar ve bu kolayca yanlış anlaşılabilir veya kasten yanlış anlamda kullanılabilir.

- Yukarıdan aşağıya geliştirme yaklaşımına karşı aşağıdan yukarıya geliştirme yaklaşımı

Bazen bir yaklaşımın daha başında, diğerinin işin etkili gerçekleşmesinde önemli olabileceğini ihmal ederek biri(örneğin pratik yapan kişiye karşı yönetim) daha önemsenir.

Bugüne kadar CMMI'ın geliştirme ve sunuş şeklinden dolayı bazı kullanıcıların CMMI'ın asıl doğası ve vermek istediği mesaj bazen farklı anlaşılmıştır ve bu farklı anlaşılmalar uyumsuz ve faydasız CMMI kullanımına yol açmıştır.

Bunun yanında, Çevik geliştirme için de benzer sorunlar mevcuttur. CMMI ve Çevik yöntemlerin yanlış kullanılışı dışında tecrübelerle kalıcı hale gelmiş bir inanış ve bilgi eksiklikleri iki paradigmanın savunucularını birbirinden uzaklaştırmaktadır.

CMMI, 1993'ten beri hiç güncellenmeyen CMM'e göre birçok değişiklik içermektedir. Bazı Çevik model kullanıcıları CMMI'yı yargılamak için CMM'i kullanıyorlar. Örneğin, 2. olgunluk seviyesinin asıl amacının tekrarlanabilir süreçler oluşturmak olduğu hakkındaki yanlış bilgiler yaygındır.

Problemi daha da karışık hale getiren bazı CMMI'ya hiç terfi etmeyen CMM kullanıcıları, yazılım geliştirme konusunda daha az esnek bir model için ısrarcı olmalarıdır. Terfi edenler arasında bile demode bazı metodolojilerde ısrarcı kişiler var.

Bunun yanı sıra yaklaşımları doğru dürüst bilip uygulamadan aktivitelerini CMMI ya da Çevik geliştirme diye tanımlayan bazı kişiler vardır. Bu da dışarıdan bakıldığında CMMI ya da Çevik geliştirme hakkında yanlış algılara neden olan önemli sebeplerden biridir.

7.2. CMMI ve Çevik Geliştirme Karşılaştırması

Günümüzde yaygın kullanım şekillerine göre aşağıdaki tablodaki gibi bir karşılaştırma yapabiliriz.

Tablo 7.1 - CMMI ve Çevik süreç karşılaştırması

CMMI	Çevik Geliştirme
Süreç ve standartlar temellidir.	İnsan temellidir.
Kurum ve yatırım konularına yoğunlaşır.	Proje ve takım konularına yoğunlaşır.
İnsanlar disiplinli, kurallara uyan ve risk almayan kişilerdir.	İnsanlar rahat, yaratıcı, risk alabilen kişilerdir.
Yönetim projenin başarısında önemli bir rol oynar.	Yönetim, sadece oluşan engellerin giderilmesine yardım eden bir fonksiyondur.
Değişiklikler kontrol altındadır ve yapılması zordur.	Değişikliklere açıktır ve yapılması kolaydır.
Genelde büyük yazılım ekiplerince tercih edilir.	Küçük yazılım ekipleri için uygundur.
Proje süresi 12 aydan fazladır.	6-12 ay arası işlerde 10-15 kişilik ekipler içindir.
Başarısızlık durumunda yüksek maliyet oluşturan alanlar içindir.	Başarısızlık durumunda düşük maliyet oluşturan alanlar içindir.
Yüksek kalitede yazılımcıya çok fazla ihtiyaç duymaz.	Uzman bir yazılım ekibi gerektirir.
Süreçler için verilecek eğitim önemlidir.	Temel eğitim önemlidir.
Belgelemeye fazla önem verilir.	Belgeleme azdır.

İletişim doküman, mail, toplantı ile gerçekleşir.	Yüzyüze konuşma, iletişim için en doğru yoldur.
Güçlü bir gözden geçirme yapısına sahiptir.	Eşzamanlı geliştirmeyi benimser.
Tahmin edilebilirlik ve kararlılık önemlidir.	Performans ve hız önemlidir.
Uzun vadeli görüntüyü dikkate alır.	Kısa ile orta vade görüntüyü önemser.
Projelere bakış açısı yukarıdan aşağıdır.	Projelere bakış açısı aşağıdan yukarıdır.

8. BÖLÜM

CMMI VE ÇEVİK GELİŞTİRME EŞLEŞTİRME

8.1. CMMI ve Çevik Geliştirme Eşleme Yöntemi

CMMI ile Çevik modellerin çok farklı çevrelerin yöntemleri olduğu düşünülse de CMMI'ın çevik geliştirme yöntemleri ile kullanılmaması için hiçbir sebep yoktur[24]. Bazı CMMI süreçleri ve çevik pratikler birbirine zıt görünmekte[25] ya da bazı CMMI süreçlerinin çevik geliştirmede karşılığı bulunmamaktadır[26]. Bu istisnai durumlar CMMI süreçlerin plan güdümlü ve çevik pratikler kombinasyonu bir yöntemle geliştirilmesine engel değildir. Burada dikkat edilmesi gereken şey her iki paradigmanın da geliştirmeyi etkileyen faktörlerinin doğasının ne olduğudur. Bunun için de organizasyonun her süreçteki gereksinimlerinin ne olduğunun anlaşılması gerekir. Bu gereksinimler anlaşıldığı takdirde CMMI hedefleri ve çevik pratikler arasında bir eşleme yapmak daha doğru olacaktır. Eşleme öncesi CMMI süreç alanlarından hangilerinin çevik geliştirme metotları tarafından desteklendiğinin, ufak uyarlamalarla bağdaştırılabileceğinin ya da tamamen çeliştiğinin bilinmesi önemlidir. Bunun için her süreç alanını, tüm genel ve özel hedeflerini detaylı incelemeliyiz.

Özel hedefler, özel pratikler ve genel pratiklerin analizinde kullanılacak ölçme sistemi şöyle olacaktır.

- Çelişen (-)
- Değinilmeyen (0)
- Kısmen desteklenen (+)
- Desteklenen (++)
- Tamamen desteklenen (+++)

“Tamamen desteklenen” çevik yöntemlerin pratiklerinin doğru uygulandığı takdirde ilgili bileşeni büyük oranda karşılayacağı manasına gelmektedir. “Desteklenen” ve “Kısmen desteklenen” ise kısıtlı olarak kapsandığını ve tamamına değinilmediği manasına gelmektedir. “Çelişen”, çevik yöntem pratiklerinin kullanılması durumunda CMMI hedefine erişilemeyeceği anlamına geliyor. “Değinilmemiş” ile “Çelişen” arasındaki ayrımı yapabilmek için ise CMMI hedefine ulaşmakta eklenebilecek çevik metot pratiklerinin çevik manifestoda bulunan ana prensiplerle çelişip çelişmediğine bakacağız.

Bu ölçme sistemini kullanırken CMMI gereği bir dokümantasyon yapılmadan sadece çevik yöntemlerin CMMI hedeflerini ne düzeyde karşılayıp, karşılamadığını anlayacağız.

8.2. CMMI ve Çevik Geliştirme Eşlemesi

Çevik yöntemlerin CMMI uygulamasına katkı sağlaması için CMMI’yı iki farklı yaklaşımla ele alacağız. CMMI genel ve özel hedeflerin ve bunların pratiklerinin çevik süreç varlıkları tarafından karşılanıp karşılanmadığını inceleyeceğiz. CMMI ve Çevik yöntemler eşlemesinde ilk olarak özel hedefler ve özel pratiklerine bakacağız. Bu pratikler yapılması tavsiye edilen ama zorunlu olmayan yerine alternatif çözümlerin uygulanabildiği uygulamalardır. Pratikleri sadece bir yol gösterici olarak kullanıp, hedeflerin gerçekleştirilmesi için mümkün olan alternatif yolları arayacağız[8]. Bu esneklik kullanılarak çevik pratiklerin destekleyici ve tamamlayıcı taraflarını araştıracağız.

Daha sonra ise CMMI 3. Seviye’ye kadar var olan genel hedefleri (tanımlanan sürecin kurumsallaşması, yönetilen sürecin kurumsallaşması) ve genel pratiklerini çevik yöntem ve pratikleri ile karşılaştıracağız. Ama özel süreç alanları ile birleşimine değil sadece genel terimlerine değineceğiz. Bunun sebebi çevik metotların özünde kurumsallaşmayı doğrudan destekleyen pratikler içermemesidir. Kurumsallaşma organizasyonel seviyede dikkate alınması gereken bir başlık iken, çevik yöntemler ise proje seviyesinde dikkate alınmalıdır. Bu nedenle yapılacak analizde genel pratiklerin analizleri sınırlı olacaktır.

8.2.1. CMMI Süreç Alanlarının ve Özel Hedeflerinin Çevik Yöntemlerle Eşlenmesi

İlk olarak tarif edilen ölçme sistemini temel alarak Çevik yöntemlerle CMMI süreç alanları ve bu alanların özel hedefleri arasındaki ilişkilerine ve çelişkilerine[5,10,11,12, 18] bakalım ve Çevik pratiklerin CMMI gereklerini ne kadar karşılayabildiğini inceleyelim.

SA1. Gereksinim yönetimi (+++)

ÖH1. Gereksinimleri yönetilmesi (+++)

Gereksinimlerin anlaşılması(ÖP1.1), müşterinin takıma entegrasyonu ve müşteri ile yoğun iletişim çevik pratikleri ile sağlanır. Vizyon, ürün gereksinim dokümanı ve hedefler gibi konular sürüm ve Sprint planlama toplantılarında tartışılır. Böylece ortak bir görüş oluşturulabilir. Gereksinimlerle ilgili anlaşılmayan bir durum oluşması halinde takım üyeleri ve müşteri yan yana çalıştıkları için o anda fikir alışverişinde bulunabilirler. Bu pratiğin karşılanmasından Scrum rollerinden ürün sahibi sorumludur. Proje iştirakçilerinin gereksinimlere onayı(ÖP1.2) ise planlama aşamasında dikkate alınır. Ürün gereksinim dokümanının spesifik, ölçülebilir, gerçekleştirilebilir, anlamlı ve zamanında olması önemlidir. Bu kapsamda hikaye kartları da gizli bir taahhüt sağlar. Müşterinin takım üyeleri ile sürekli iletişim halinde olması da bu onayı destekler.

Gereksinimlerin değişiklikleri(ÖP1.3) hızlıca dikkate alınarak tartışılır. Zaten gereksinim değişikliği çevik sürecin diğer modellere göre en önemli özelliğidir. Sonradan ortaya çıkan ve daha önce bir Sprint için seçilmemiş gereksinim maddeleri ürün gereksinim dokümanına eklenebilir. Gereksinimlerin öncelikleri, büyüklükleri yada harcanacak zaman tahminleri, ürün gereksinim dokümanındaki iş değerleri güncellenebilir. Ama gereksinim değişiklikleri pratiğinin gerçekleştirilmesinde ürün sahibi başta olmak üzere takım koçu ya da liderinin rolü önemlidir.

Gereksinimlerin izlenebilirliği(ÖP1.4) çevik modelin açık hedeflerinden biri değildir. Çevik yöntemler gereksinimlerin müşterinin istediği yazılımı oluşturacak düzeyde toplanmasının yeterli olduğunu düşünür ve dokümantasyonu pek önemsemez. Fakat ürün gereksinim dokümanı tüm herkese açıktır. Sprint dokümanında ürün gereksinim dokümanındaki maddelere eşleştirilmiş görevler bulunur. Bu dokümanların yanı sıra

hikaye kartlarının ve dokümantasyonun eski versiyonlarının kayıt altında tutulması ile izlenebilirlik genişletilebilir. Çift yönlü izlenebilirliğin ana sorumlusu Scrum yöneticisidir.

Gereksinimler ile iş ürünleri arasındaki tutarsızlıklar(ÖP1.5), kısa zaman aralıklarında çıkarılan iterasyonlar ve sürümler, sürekli müşteri geri beslemesi, Scrum hikayeleri, yapılanlar ile ihtiyaçlar arasındaki uyumsuzlukları belirleyen fonksiyonel testler ve birim testleri ile değerlendirilir. Proje boyunca ürün gereksinim dokümanı bölünerek Sprintler ve sürümlere dönüşür. Yapılan işler ve oluşan tüm değişimler her Sprint sonrası ürün gereksinim dokümanına not edilir. Bunların gerçekleştirilmesinden ürün sahibi sorumludur.

SA2. Proje planlama(+++)

ÖH1. Tahminlerin oluşturulması(+++)

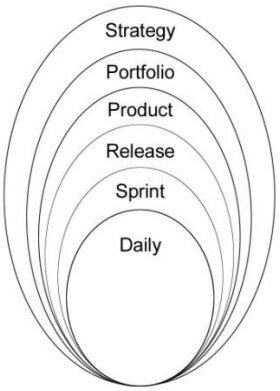
Tüm çevik yöntemlerle projenin başında işlerin ne kadar çaba ile yapılacağına dair gerçekçi tahminler geliştirmek mümkün olmasa da proje kapsamının belirlenmesi(ÖP1.1), iş ürünleri ve görev özelliklerini tahmini(ÖP1.2) ve işgücü ve maliyet tahmini(ÖP1.4) yapılırken XP kullanıcı hikayeleri, Scrum planlama oyunu, Scrum toplantıları ve Scrum kalan zaman grafiklerinden yararlanılır. Daha küçük ve daha sık program sürümleri çevik pratiği uygulanarak planlama tahminlerinin duyarlılığı arttırılır. Proje boyunca gerçekleştirilecek kullanıcı hikayeleri ve görevler için tahminler oluşturulur ve bunlar ürün gereksinim dokümanında saklanır. Gerektikçe daha sonra düzeltilebilir. Tahminlerin doğruluğu kısa iterasyonları sağlayacak kısa planlamalarla arttırılabilir. Tüm bu pratiklerden ürün sahibi ve Scrum yöneticisi sorumludur.

Çevik yöntemler bir proje yaşam döngüsünün tanımlanması(ÖP1.3) ile ilgilenmez. Ürün gereksinim dokümanı bile en başta oldukça yüzeyseldir.

ÖH2. Bir proje planının geliştirilmesi(+++)

Çevik süreçlerde planlar geleneksel yazılım geliştirme modellerinden farklı olarak değişik zaman aralıklarında birbirini kapsayacak şekilde içice geliştirilir ve uygulanır.

Çevik geliştirme en az ürün seviyesine kadar bu planları oluşturacak ve gerçekleştirecek esnek bir yapıda oluşturulmuştur. Bunun yanında çevik geliştirme takımı sürüm, iterasyon ve günlük seviyede planlarda aktif rol alır. Tüm bu planlar farklı düzeylerde ve farklı kişilerle eş zamanlı gerçekleştirilebilir. Böylece projeye hatta projelerin üzerinde kuruma farklı seviyelerde bakış açısı sağlanır[2]. Proje planı, bu sürüm planı ve proje boyunca geliştirilecek iterasyon planları ile oluşturulabilir. Böylece uzun dönem planları belirsiz kalabilir ve sadece kısa dönem planları detaylandırılır.



Şekil 8.1 - Planlama soğanı

Çevik yöntemler daha güvenilir proje planının(ÖP2.7), bütçesinin ve takviminin(ÖP2.1) oluşturulmasına yardım eder. Bunun sağlanması için Scrum ürün gereksinim dokümanının oluşturulması ve XP haftalık 40 saat pratiğinin uygulanması önemlidir. Çevik yöntemlerle daha küçük ve daha sık program sürümleri ile her iterasyon için daha küçük kapsamda daha iyi planlar ve takvimler oluşturulabilir. Bu iterasyon planları proje süresince evrimleşir. Uzun dönem planlar kaba kalırken kısa vadede gerçekleştirilecek planlar detaylandırılır. Bu küçük iterasyonlar risklerin belirlenmesi (ÖP2.2) açısından önemlidir. Her iterasyon öncesi riskli olduğu düşünülen gereksinimlerin öncelikleri arttırılabilir ve diğerlerinden önce gerçekleştirilebilir.

Kaynakların daha iyi planlanması(ÖP2.4) konusunda takımlar verilen bütçe sınırlarını aşmamak şartıyla özgürdürler. Kısa iterasyonlar yaparak kendi çalışma ortamları dahil tüm kaynak planlamalarında değişikliğe gidebilirler.

Çevik yöntemlerle belirli bir safhada gerekli olan bilgi ve yeteneğin saptanması (ÖP2.5) mümkündür. Ayrıca saptanan herhangi bir eğitim ihtiyacı projeni her aşamasında dikkate alınır. Çünkü bilginin kurum içindeki yayılımı önemlidir. Paydaşların planlama

sürecine etkin katılımı(ÖP2.6) ise işbirlikçi planlama ve yeniden gözden geçirme oturumlarına dahil edilerek sağlanır. Ayrıca müdahale etmemeleri şartıyla günlük Scrum toplantılarını izlemelerine müsaade edilebilir.

Çevik yöntemler veri yönetimini(ÖP2.3) planlamaz yani bu pratik çevik model tarafından kapsanmaz. Fakat ürün gereksinim dokümanı ilgili paydaşlar ve katılımcılar dahil olmak üzere herkesin erişebileceği merkezi bir yerdedir ve gereksinim maddelerine ilişik tüm yorum, görev ve testler görünürdür.

ÖH3. Plana olan taahhütlerin sağlanması (+++)

Sürüm ve iterasyon planlarına taahhüt(ÖP3.3), işbirlikçi planlama çevik pratiği uygulanarak tüm takım üyelerinin ve ilgili tüm paydaşların yüksek katılımları ve sorumluluk almaları ile sağlanır. Bunun yanı sıra ilk Sprinte başlamadan bütçe ilgili paydaşlar tarafından onaylanmış olmalıdır. Ayrıca takımın çalışmasını etkileyebilecek tüm çıktıları ve sınırlar ürün gereksinim dokümanında yer almalıdır. Bu proje planlarının her aşamada gözden geçirilmesi(ÖP3.1.) açısından önemlidir. İş ve kaynak seviyeleri arasında uzlaşma(ÖP3.2) ise proje boyunca ürün gereksinim dokümanında gerektikçe yeni önceliklendirme ve yeni tahminlemelere gidilmesi ile sağlanır. Planların gerçekliğinin hesaplanmasında takımın hızı bilgisi kullanılır.

SA3. Proje izleme ve kontrol (+++)

ÖH1. Plana göre projenin izlenmesi (+++)

Sprint, sürüm ve ürün gereksinim dokümanları ve geri kalan zaman grafikleri projenin planlanan ihtiyaçlara ve planlanan tarihlere uygunluğunun izlenmesini(ÖP1.1) sağlar. Bu zaman çizelgeleri ve tahminler değişiklik durumlarını da kontrol edebilmek için tüm proje boyunca izlenmelidir. Kendi kendine organize olabilen takımlar sık ve düzenli durum toplantıları yaparak her Sprint için seçilen hangi gereksinimlerin gerçekleştirildiğini, iş taahhütlerini(ÖP1.2) ya da risk ve engellerini(Ö.P.1.3) yüz yüze ifade ederler. Risk ve engellerle ilgili güncellenebilir ve herkesin görebileceği bir aksaklık dokümanı (Impediment Backlog) oluşturulur. Günlük ilerleme izleme toplantıları ve takım üyelerinin yoğun iletişimi çevik pratikleri ile takım üyeleri plana göre gerçekleştirilen iterasyonun durumu hakkında günü gününe bilgi sahibi olur.

Böylece plana göre projenin genel ilerleyiş bilgilerinin izlenmesi(ÖP1.6) kolaylaşır. Ayrıca her Sprint sonrası yapılacak yeniden gözden geçirme ve resrospektif toplantıları ile belirlenmiş proje aşamalarında projenin başarıları ve sonuçları hakkında bilgiye ulaşılabilir(ÖP1.7). İterasyonların zaman çizelgesine uygunluğu fonksiyonel testlerle kontrol edilir. Daha küçük ve daha sık program sürümü pratiği gerçekleştirilen kısa iterasyonlar ve plana düzenli taahhütlerle oluşturulan katı sistem, projenin dayanak(baseline) uygunluğunun izlenmesini kolaylaştırır. İlgili paydaşlar proje planına uygun olarak bu süreçlere yorumcu ya da izleyici olarak dahil edilir(ÖP1.5). Sadece veri yönetiminin izlenmesi(ÖP1.4) pratiği çevik yöntemler tarafından karşılanmaz. Yeniden tahminleme yapılan gereksinimlerin başlangıç değerlerinin tutulması faydalı olabilir.

ÖH2. Düzeltici faaliyetlerin kapanıncaya kadar yönetilmesi (+++)

İhtiyaç duyulan düzeltici faaliyet konuları hakkında günlük ilerleme takip toplantıları ve takım üyeleri arasında yoğun iletişimi çevik pratikleri ile gayrı resmi olarak bilgi edinilir ve analiz yapılır(ÖP2.1). Bu analiz sonucu farkedilen ve belirlenen takımla ilgili veya dışsal sorunlar aksaklık dokümanına dahil edilmelidir. Takım üyeleri kurumsal sorunları belirlerken Scrum Yöneticisi ile iletişim halinde olmalıdır. Aksaklık dokümanındaki belirlenmiş tüm aksaklıklar önceliklendirilerek takım ve yönetim tarafından üzerinde çalışılmalıdır. Böylece sorun ve olası risklerle ilgili en doğru düzeltici önlemler alınabilir(ÖP2.2). Yapılacak bu düzeltici faaliyetler, bir metodun ya da fark edilecek bir fonksiyonalitenin düzeltilmesi olabileceği gibi takımın çalışma ortamı ile ilgili bir aksaklıkta olabilir. Çözülen sorunlar ve giderilen risk olasılıkları aksaklık dökümanından silinerek düzeltici faaliyetlerin doğru yönetilmesi sağlanabilir(ÖP2.3). Bunun yanısıra daha küçük ve daha sık program sürümleri ile gerçekleştirilen her yeni iterasyon, bir öncekinden kalan sorunlar için düzeltme ve yeni ayarlamalar yapmak için iyi bir fırsattır.

SA4. Tedarikçi Sözleşme Yönetimi (0)

ÖH1. Tedarikçi Sözleşmesinin oluşturulması

Bu süreç alanı, çevik yöntemler tarafından ele alınmaz. Bu süreç alanının hedeflerini yerine getirecek şekilde genişletilebilir. Fakat bu uyarlama, tedarikçilerin katılımı

iteratif geliştirmeyi engellediği durumlarda problem oluşturabilir. Bazı iterasyonların sonunda çalışır yazılımın sağlanması için tedarik edilmesi gereken bazı şeyler olabilir. O anda olmayışları kritik bir probleme yol açabilir. Bu nedenle ürün gereksinim dokümanında yer alan ürün ya da ürün bileşenlerinin satın alma yönteminin belirlenmesi(ÖP1.1) dış tedarikçiler tarafından yerine getirilmelidir. Potansiyel tedarikçilerle ile bir başlangıç Sprint planlama toplantısı yapılarak sağlanacak ürünle ilgili ortak bir görüş oluşturulabilir. Gereksinimlerle ilgili onların zaman ve bütçe tahminleri öğrenilerek değerlendirme yapılabilir. İlk Sprint toplantısında birden fazla tedarikçi ile yola çıkılarak sonradan tahmini en iyimser olan tedarikçi seçilebilir (ÖP1.2). Bundan sonra tedarikçi ile bir sözleşme oluşturulması (ÖP1.3) mümkündür. Bu sözleşme seçilen ürün gereksinimlerinin tedarikçi tarafından yerine getirilmesi, her Sprint sonunda yapılanların gözden geçirilmesi ve kabul kriterlerini kontrol eden testlerin de sürece dahil edilmesi gibi maddeler içerebilir.

ÖH2. Tedarikçi Sözleşmesinin gereklerinin yerine getirilmesi

Projenin devamı için tedarikçinin sözleşmede yazılan ilgili maddeleri yerine getirmesi (ÖP2.1) gereklidir. Her iterasyon sonunda Sprint gözden geçirme toplantıları ile gerçekleştirilen faaliyetler kontrol edilebilir(ÖP2.2) ve değerlendirilebilir(ÖP2.3). Yapılan değerlendirme sonucu önceden belirlenen kabul kriterlerini sağlayan ürünler kabul edilir (ÖP2.4). Kabul edilen ürünün kullanıma geçilmesi(ÖP2.5) ise sürekli entegrasyon pratiği sayesinde kolaylıkla gerçekleşir.

SA5. Konfigürasyon yönetimi (+++)

ÖH1. Dayanakların oluşturulması(+++)

Konfigürasyon nesneleri, kod, tasarım, testler ve gereksinimlerdir. Yazılım konfigürasyon yönetimi çevik pratiğinin gereği konfigürasyon sisteminlerinin ve araçlarının kullanılması, güvenilirliği arttıran sürekli entegrasyondan dolayı tavsiye edilir. Gerekli konfigürasyon birimlerinin belirlenmesi(ÖP1.1) için öncelikle ortak politikalar ve kodlama standartları oluşturulmalıdır. Belirlenen konfigürasyon birimlerinin yönetimi(ÖP1.2) için bir sistem mevcut değilse başlangıç ürün gereksinim dokümanına uygulama eklenir. Konfigürasyon yönetim dayanakları(ÖP1.3) Sprint ve sürüm takvimlerine göre belirlenir. Dayanaklar genellikle her iterasyonun sonunda

oluşturulur. Düzenli fonksiyonel testlerle de desteklenir. Tüm bu işlerin doğru şekilde oluşturulması ve gerçekleştirilmesinde koçun rolü önemlidir.

ÖH2. Değişikliklerin izlenmesi ve kontrol edilmesi(+++)

Değişiklikler; eşli programlama, birim testler, müşteri katılımı, sürekli müşteri geri beslemesi gibi pratiklerle takip ve kontrol edilir (ÖP2.1). Ama öncesinde konfigürasyon araçlarının ortak sahiplenilmesi sağlanmalıdır. Tüm faaliyetlerle elde edilen bilgiler çeşitli araçlarla dokümantasyona dönüştürülebilir.

ÖH3. Bütünlüğün sağlanması(+++)

Çevik yöntemler, projeyi sürekli entegrasyona yani bütünleşmeye zorlar. Ayrıca çevik yöntemle geliştirilen projede üretilen kod, kodlama standartlarından dolayı kolay okunabilirdir. Gerekli denetlemeler(ÖP3.2); eşli programlama, müşteri katılımı, test etme ile bu hedef gayrı resmi olarak gerçekleştirilir. Çalışan ürün sürekli olarak dayanaklara uygun tutulurken konfigürasyonlar yeniden gözden geçirilir. Çevik yöntemler konfigürasyon yönetim kayıtlarının oluşturulması(ÖP3.1) ile ilgilenmez ama sürekli entegrasyon sağlamak için kullanılan bazı gelişmiş konfigürasyon araçları bu ihtiyacı da giderebilmektedir. Takımın ortak sahiplenme kurallarına uygun hareket etmesi de bu açıdan önemlidir.

SA6. Süreç ve ürün kalite güvencesi (+)

ÖH1. Süreçlerin ve iş ürünlerinin tarafsız değerlendirilmesi(+)

Çevik yöntemler, süreçlerin, iş ürünlerinin ve uygulanabilir süreçlere hizmet eden servislerin değerlendirilmesini açık şekilde talep etmez. Ürün gereksinim dokümanındaki fonksiyonel olan ya da olmayan tüm gereksinimler için testler tanımlanmasını ister ve bu zorundalık süreçlerin tarafsız değerlendirilmesi(ÖP1.1) için gerekli yapıyı oluşturur. Gerekiyorsa bu testlerin geliştirilme yöntemleri de gereksinim dokümanına eklenebilir. İş ürünleri ve hizmetlerinin tarafsız olarak değerlendirilmesi(ÖP1.2) için ise her teslim edilebilir ürüne uygun test güdümlü yöntemler benimsenir. Çevik yöntemlerin tarafsız değerlendirmede doğru şekilde uygulandığı kontrolünün tek senedi XP'nin kullanımında takıma rehberlik eden koç ve

Scrum rollerinden Scrum yöneticisidir. Ayrıca eşli programlama pratiği kodlama seviyesinde denetleme imkanı sunar.

ÖH2. Tarafsız görüş sağlanması(+)

Kalite konuları XP takımı içinde kolayca konuşulabilmelidir. Koçun ya da Scrum yöneticisinin işi bu özel hedefi desteklemektir. Ayrıca günlük toplantılar ve ortak kod sahiplenme gibi çevik pratikler görüşünün tarafsızlaşmasına katkı sağlar. Sprint gözden geçirme toplantılarıyla projede şeffaflık oluşturulur ve var olan sorunlardan herkesin haberdar olması sağlanır, retrospektif toplantıları ile de aksaklık dokümanındaki kalite sorunlarından kaynaklanan problemler çözülür(ÖP2.1). Fakat, çevik yöntemlerde itaatsizlik konularına çözüm ve kalite güvence aktivitelerinin kayıtlarının sağlanması(ÖP2.2) için katı prensipler ilkeler yoktur. Kullanıcı hikayeleri, Scrum gereksinim dökümanları, Scrum kalan zaman grafikleri ve Scrum toplantı bilgileri kalite güvence aktivite kayıtlarının oluşturulmasında kullanılabilir. Doğru bir kayıt mekanizması için öncelikle tamamlanan gereksinim maddeleri ürün gereksinim dokümanından çıkartılır. Ancak tamamlanmayan bir gereksinim maddesi Sprint, iterasyon ya da sürüm için bir ihtiyaç olmaktan çıkması durumunda ürün gereksinim dokümanından da çıkarılabilir.

SA7. Ölçme ve analiz (++)

ÖH1. Ölçme ve analiz çalışmalarının uyumlu hale getirilmesi (++)

Bir ürününün ana ölçme kriteri projenin bitiş tarihinde teslim edilmesidir. Müşteri gereksinimlerini ne derece karşıladığının sonucu en önemli ölçüm değeridir. Çevik yöntemler ölçme ve analiz konusunda için özel bir prensip içermez. Ama yöneticilerin ve müşterilerin proje süreçlerine dahil edilmesi, ara sürümler ve iterasyonlarla ürün gereksinimlerinin ne kadar karşılandığı bilgisi, tamamlanan kullanıcı hikaye kartları ve günlük toplantılar ile süreçle uyumlu ve güçlü bir ölçme ve analiz ortamı sağlar. Ayrıca bu ortamdaki bilgileri saklayıp, kullanan uygulama yazılımlarının kullanarak bu uyumluluk otomatikleştirilebilir. Ölçme ve analiz çalışmalarını takım koçu ya da bu işle görevlendirilecek başka biri takipçi sıfatıyla izleyebilir.

ÖH2. Ölçme sonuçlarının sağlanması (++)

Ölçüm verileri, takım içinde yoğun iletişim ile sağlanır. Takipçi bu veriyi analiz eder ve bir duvar resmi kullanarak sonuçları takımla paylaşır. Bu veri genellikle kalıcı depolanmaz. Fakat, çalışma tahmini ve çevik takımların izlenmesi için kullanılan bazı araçlar mevcuttur. Bu araçlar kullanılarak ölçüm verileri ve sonuçları fazla güç harcanmadan kalıcı olarak depolanabilir.

SA8. Gereksinim Geliştirme (++)

ÖH1. Müşteri gereksinimlerinin geliştirilmesi(++)

Müşteri gereksinimleri ortaya çıkartır ve onları hikaye kartlarında, fonksiyonel testlerde açıkça tanımlar. Yazılım geliştiriciler bu görevlerde müşteriyi destekler. Yoksa gereksinim tanımlama belirsiz şekliyle kalır. Detaylar geliştirme süresince müşteri ile tartışılmalıdır. Bu süreçte müşteri ile yakın çalışma pratiği etkindir. Bu hedefe ulaşırken gereksiz gereksinim dokümanlarının oluşturulmasından uzak durulur. Hem bu dokümanların oluşturulmasının ve idamesinin maliyet gerektirmesinden dolayı hem de bu dokümanlar müşteri için bir değer oluşturmadığından CMMI'ya nazaran gereksinim dokümantasyonu geliştirme minimum tutulur.

ÖH2. Ürün gereksinimlerinin geliştirilmesi (++)

Müşteri gereksinimleri ürün gereksinimlerini geliştirir ve düzeltir. Bunlar görev kartları kullanan özelleşmiş şeylerdir. Nispeten daha belirsizdirler.

ÖH3. *Gereksinimlerin analiz edilmesi ve onaylanması (++)*

Gereksinimlerin analizi sürekli müşteri geri beslemesi pratiği ile ortaya çıkar. Programcılar, gereksinimlerin ortaya çıkarılması süresince müşteriyle görüş alışverişinde bulunurlar. Birim testler ve test güdümlü geliştirme gereksinimlerin daha iyi anlaşılması ve onaylanmasında etkilidir. Ayrıca, gereksinim değişikliklerinin kabulü ve iterasyonların kullanılması sabit bir gereksinim analiz ve onaylama sağlar. İşlevsel kavramlar ve senaryolar, fonksiyonel testler kullanılarak oluşturulur. Fakat baştan sonra gereksinim analizi mümkün değildir.

SA9. Teknik çözüm (+++)

ÖH1. Ürün bileşen çözümlerinin seçilmesi (+++)

Alternatif çözümler; projenin başında prototipler sayesinde, daha sonra ise kodu yeniden yapılandırma (refactoring) ve yinelemeli geliştirme sayesinde ortaya çıkarılır.

ÖH2. *Tasarımın geliştirilmesi (+++)*

Tasarım ilk olarak olabildiğince basit oluşturulmalıdır. Oldukça esnek ve kolay değiştirilebilir olmalıdır. Değişiklik maliyeti böylece azalacaktır. Her tasarımda tasarım yapılmalı ve değişiklik maliyetlerini azaltacak şekilde yinelemeli olarak güncellenmelidir. En doğru tasarım tekrarlanarak ortaya çıkar. Ayrıca kod, tasarım dokümanı olarak kullanılabilir.

ÖH3. *Ürün tasarımının gerçekleştirilmesi (+++)*

Bu özel hedef, çevik pratiklerden en fazla faydalanan hedeflerden biridir. Çünkü bu hedef tamamen ürün geliştirmeye yöneliktir ve bilindiği gibi çevik yöntemlerde aslında geliştirme pratiklerine yoğunlaşır. Kodu yeniden yapılandırma, kodlama standartları, eşli programlama, test güdümlü geliştirme ve sürekli geliştirme gibi çevik pratikler tasarlanan ürünün ortaya çıkarılmasında kullanılır. Müşteri istediği takdirde bir ürün destek dokümanı da geliştirilir.

SA10. Ürün bütünleştirme (+++)

ÖH1. Ürün bütünleştirme için hazırlıkların yapılması (+++)

Bu hedef, sürekli entegrasyon pratiği ile başarılı şekilde sağlanır. Bu entegrasyon adımları sık sık gerçekleştirildiğinden titiz bir hazırlık önemlidir.

ÖH2. *Arayüz uyumluluğundan emin olma (+++)*

Arayüz uyumluluğu her entegrasyon adımında tüm testlerin çalıştırılması ile garanti edilir.

ÖH3. *Ürün bileşenlerinin bir araya getirilmesi ve ürünü teslim edilmesi (+++)*

Bileşen toplama ve teslim projenin nihai hedeflerindendir. Sürekli entegrasyon ve direkt müşteri katılımı sayesinde bu hedefe ulaşmak kolaylaşır.

SA11. Doğrulama (+++)

ÖH1. Doğrulama için hazırlıkların yapılması (+++)

Doğrulama yoğun birim testler ile gerçekleştirilir. Doğrulamaya hazırlanırken de bu konuya yoğunlaşılır. Bu süreçte uygun hazırlık aktivitelerinin gerçekleştirilmesi için birim testleri otomatik gerçekleştirecek bir test çatısı (framework) kullanılmalıdır. Bunun yanı sıra "önce test" yaklaşımını benimseyen test güdümlü geliştirme ile gereksinimlerin doğru şekilde karşılandığı doğrulanabilir. Yani tüm testler kodlamadan önce yazılmalıdır. Doğrulanacak iş ürünlerinin seçilerek herbiri için uygun entegrasyon testleri tanımlanmalı, bu testler Sprint gözden geçirmeleri öncesinde ve sırasında istenen fonksiyonaliteyi sağladığı doğrulanmalıdır.

ÖH2. Eşdeğer gözden geçirmelerin gerçekleştirilmesi (+++)

Eşdeğer gözden geçirme, çevik yöntemlerden XP'nin üstü kapalı olarak bir parçasıdır. Eşli programlama, kodu yeniden yapılandırma ve kolektif kod sahiplenme prensibleri sürekli bir gözden geçirme sağlar.

ÖH3. Seçilen iş ürünlerinin doğrulanması (+++)

Ana doğrulama yöntemleri, sürekli gerçekleştirilen eşdeğer gözden geçirme ve testlerdir. Yukarıda sayılan pratikler de doğrulama sürecinde etkindir.

SA12. Geçerleme (+++)

ÖH1. Geçerleme için hazırlık yapılması (+++)

Geçerlemenin amacı müşteri tarafından istenen ürünün gereklerinin tam olarak karşılandığını göstermektir. Ana ve tek kriteri ise ürünün müşteri tarafından kabul edilmesidir. Geçerleme çevik projelerde müşteri katılımı, sürekli müşteri geri beslemesi, ve sık yapılan sürümlerle gerçekleştirilir. Ürün gereksinim dokümanına uygun olarak geçerlenecek ürünler seçilir (ÖP1.1). Daha sonra takım tarafından ürünler için uygun

kabul ortamı oluşturulur (ÖP1.2). Sprint gözden geçirme toplantılarında ise istenen fonksiyonaliteyi geçerleyecek prosedürler ve kabul kriterlerini(ÖP1.3) içeren yöntemler takım tarafından belirlenir.

ÖH2. Ürün veya ürün bileşenlerinin geçerlenmesi (+++)

Takım tarafından yapılan işler müşteri tarafından sürekli olarak geçerlenmelidir. Çevik yöntemlerde müşteri takıma entegre edildiğinden hemen ulaşılabilir konumdadır ve bu amacı gerçekleştirmek kolaydır. Ayrıca müşteri her iterasyon sonundaki dağıtımı da geçerler. Bu uygulama ek gereksinimlerin ya da gereksinim değişikliklerinin ortaya çıkmasını sağlar. Müşterinin katılımının ürünün tasarlanan operasyonel çevrede kullanıma uygunluğunun artmasında etkisi büyüktür. Geçerlemenin gerçekleştirilmesi(ÖP2.1) için kabul testlerinin uygulanması yeterlidir. Geçerleme sonuçlarının analiz edilmesi(ÖP2.2) ise kabul test sonuçları ile beklenen sonuçların karşılaştırılması ile gerçekleştirilir.

SA13. Risk yönetimi (+++)

ÖH1. Risk yönetimi için hazırlıkların yapılması(+)

Çevik yöntemler risk yönetiminin nasıl olacağını açıkça belirtmez. Fakat projeler bu hazırlıkların bir kısmını zaten yerine getirir.

ÖH2. *Risklerin tespit edilmesi ve analizi (+++)*

Çevik yöntemlerden XP, planlama aşamasında risklerin tanımlanmasına ve analiz edilmesine zorlar. Kendi kendine organize olan takımlar ve yoğun müşteri iletişimi, risklerin tanımlanmasına büyük katkı sağlar.

ÖH3. Risk olasılıklarının azaltılması(+++)

Kendi kendine organize olan takımlar ve yoğun müşteri iletişimi, risklerin yatıştırılmasına da katkı sağlar. Kısa iterasyonlarla oluşan esneklik risklerin azaltılmasında etkin bir araçtır.

SA14. Bütünleşik proje yönetimi (++)

ÖH1. Projenin tanımlı sürecinin kullanılması (0)

Çevik yöntemler bir projenin tamamı için tanımlı bir süreç oluşturmaz. Çevik pratikler sadece proje geliştirme içindir.

ÖH2. İlgili paydaşlar ile işbirliği ve koordinasyon(+++)

İşbirlikçi takım çevik pratiği ile yazılım geliştiricileri, müşteriler, tedarikçiler, testçiler ve yönetim bir araya getirilip koordinasyon sağlanır.

SA15. Kurumsal eğitim (+++)

ÖH1. Kurumsal eğitim yeteneğinin oluşturulması (+++)

Kurumsal eğitimin amacı yetenekli ve bilgili insanlar oluşturarak kendi rollerinde daha başarılı ve etkin olmalarını sağlamaktır. Kurumsal eğitim XP çevik yönteminin araştırma aşamasında yerine getirilir. Eşli programlama ve koçluk bir eğitim olarak görülebilir. Çünkü bu pratikler kurumun eğitim yeteneklerini arttırır.

ÖH2. Gerekli eğitimlerinin sağlanması (+++)

Yukarıda belirtildiği gibi, eğitim araştırma safhasında açıkça ortaya konur. Tüm proje boyunca da koçluk ve eşli programlama ile dolaylı olarak ele alınır. Fakat eğitim etkinliğinin kayıtlarının ve değerlendirmelerinin oluşturulmasına dair eksiklikler içerir.

SA16. Kurumsal süreç tanımlama (0)

Bu süreç alanı proje seviyesinde değil kurum seviyesinde uygulama gerektirdiği için çevik yöntemler tarafından ele alınmaz ve desteklenmez.

SA17. Kurumsal süreç odaklılık (-)

Bu süreç alanı da kurum seviyesinde uygulama gerektirdiği için çevik yöntemler tarafından desteklenmez. Çevik yöntemlerde uyarlamalar genellikle tüm proje süresince yapıldığı için çelişki içindedir. Bu gelişmeler belgelenemeyeceğinden dolayı o proje ile sınırlıdır. Gelişmelerdeki bilgi kişilerde saklıdır. Diğer projeler ancak aynı insanlar bu

projelerde de bulunduğu takdirde bu gelişmelerden fayda sağlar. Fakat büyük kurumlarda birçok proje vardır. Bu demektir ki, belirli bir projenin deneyimlerinden ortaya çıkan şeylerden diğer tüm projelerde de faydalanmak imkansızdır. Kişilerin emekli olduğu ya da işten ayrıldığı düşünüldüğünde bilginin kalıcı olmadığı da görülecektir. Bu çelişki önceki projelerde ortaya çıkan iyi pratikleri depolayan kurumsal havuzlar oluşturularak ya da projelerde edinilen tecrübelerin projeler arasında değiş tokuş edilmesi yönünde kurumsallaşma sağlanarak azaltılabilir.

SA18. Karar analiz ve çözümleme (-)

Turner, çevik yöntemlerle tercih edilen yeni durumlara hızlı geçişin ancak resmi bir değerlendirme sürecine girmekle mümkün olduğunu söyler. Oysa XP, CMMI'ın aksine alternatifleri gayrı resmi olarak tanımlar ve değerlendirir.

SA19. Nicel proje yönetimi (-)

İstatistik metotları tanımlı süreçler üzerinde yoğunlaşır. Nicel analizler statik bir dayanağa ihtiyaç duyduğundan bireyler üzerinde yoğunlaşmaz. Bu nedenle istatistik yöntemleri çevik prensiplerle çelişki içindedir. Ayrıca istatistik yöntemleri büyük sayıları temel alır ve büyük takımlardaki sonuçların ortalamalarını hesaplar. Çoğu çevik yazılım projeleri küçük olduğundan istatistik yöntemlerinin kullanımı tartışılabilir.

SA20. Kurumsal süreç performansı (–)

Bu süreç alanı kurumsal seviyede uygulama gerektirdiği için çevik yöntemler tarafından desteklenmez. Zaten çevik yöntemler süreç ya da süreç alanı bazlı konulardan çok bireylere yoğunlaşır. Turner bir süreci ölçmenin, dayanakların ve modellerin bakım idamesini yapmanın çevik manifesto ile çeliştiğini söyler.

SA21. Kurumsal yenilik ve yayma (-)

Çevik yöntemlerde süreç gelişme ve adaptasyonları proje içinde yapılır ve belgelenmez. Bundan dolayı tüm kuruma yaygınlaştırılamaz. Bu konu çevik yöntemlerle çelişkili olan kurumsal süreç odağına bağımlıdır. Kurumsal seviyede bir konudur.

SA22. Nedensel analiz ve çözümleme (0)

Çevik yöntemler ne hataları ölçer ne de bu hataların sebeplerini analiz eder. Bu nedenle bu süreç alanı çevik yöntemler tarafından kapsanmaz ve desteklenmez.

2. CMMI Genel Hedeflerinin Çevik Yöntemlerle Eşlenmesi

En önemli odak noktalarımızdan biri de CMMI kullanılırken çevik yöntemlerin kurumsallaşmasıdır. Kurumsallaşma, CMMI'da "kurum kültürünün bir parçası olarak rutinleşmiş kurumun iş yapma biçimi"[5] olarak tanımlanır. Ama bazı kimseler[11] kurumsallaşmayı basit bir şekilde "iş yapma yolu" olarak tanımlamaktadır. CMMI birçok süreç alanına uygulanan genel pratikler ile kurumsallaşmayı destekler. Çevik süreçler ise 3. yetenek ve olgunluk seviyelerine kadar olan genel hedefleri (özel hedefleri gerçekleştir, yönetilen bir süreci kurumsallaştır, tanımlanmış bir süreci kurumsallaştır) doğru çevik pratiklerle destekleyerek kurumsallaşmayı güçlendirebilir. Yani süreç alanlarına ait özel hedef ve özel pratikleri yerine getiren çevik yöntemler kurumsallaştırılabilir.

Şimdi genel pratiklerin çevik yöntemlerle ilişkilerini inceleyelim.

GP1.1. Temel pratikleri yerine getir (+++)

Özel hedefler ve bağlı özel pratiklerinin çevik yöntemlerle ilişkisi üst bölümlerde anlatılmıştı. Çoğunlukla destekleyici olduğunu, hedef ve pratiklerin gereklerini karşıladığını biliyoruz.

GP2.1.Kurumsal bir politika oluştur (0)

İlk olarak CMMI süreçlerinde çevik yöntemlerin nerede ve ne zaman kullanılacağının bilinmesi önemlidir. Bir kurum projenin boyutuna, ürünün tipine, teknolojiye ve diğer faktörlere bağlı olarak tüm projelerinde veya projelerinin alt bölümlerinde çevik yöntemleri kullanabilir[22]. Bu esnek bilinç kurumsal kültürün bir parçası haline getirilebilir. Çevik yöntemler projeler bazında değil proje bazında çözümler sunar ve kurumsallaşmayı desteklemez. Dolayısıyla kurumsal bir politikanın oluşturulmasında bir etkinlik göstermez. Ama yüz yüze iletişim veya üst düzey yönetimin ziyaretleri ve

çalışmalara müdahil olması gibi bazı etkili çevik pratikler kurum kültürünün bir parçası haline getirilebilir.

GP2.2. Süreci Planla(0)

Çevik yöntemlerin disiplinsiz bir çalışma şekli olduğu önyargısı yaygındır. Çevik yöntemler bir süreç tanımını desteklemeseler de, tanımlı bir süreci takip edecek şekilde planlanıp uygulanabilirler. Bu tanımlı süreç, projenin ne yapacağını açıklarken minimum zorunlu bilgi sunan bir adımlar serisi içermelidir. Ayrıca bu süreç planı, diğer genel pratiklerin projede nasıl gerçekleştirileceği bakış açısını da sunmalıdır. Scrum, ürün gereksinim dokümanı veya sprint dokümanı gibi kavramlarla bu ihtiyacı destekleyebilir.

GP2.3. Kaynakları Sağla (+)

Her proje gelişmiş yetenekleri, profesyonelleri, yeterli bütçeyi, uygun olanak ve araçları ister, ihtiyaç duyar ve umar. Bu ihtiyaçların yönetilmesi eylemin gerçekleştirilmesi açısında önemli ve faydalıdır. Sprint planlama toplantısı gibi çevik pratikler kaynakların sağlanması ve değişikliklerinin yönetilmesinde faydalı olabilir.

GP2.4. Sorumlulukları Ata (+++)

Bir projenin başarılı olması için sorumluluk, yetki ve rollerin iyi tanımlanmış olması gereklidir. Rol modeli sorumlulukları belirli takım üyelerine tahsis eder. Ayrıca geliştiriciler proje süresince belirli görevlerle ilgili sorumluluk alırlar. Scrum'da Ürün sahibi, Scrum yöneticisi ve takım üyesi gibi roller tanımlamıştır. Takım üyesi alan uzmanı, sistem mühendisi, yazılım mühendisi, yazılım mimarları, programcılar, analistler, kalite güvence sorumluları, testçiler, arayüz tasarımcılar gibi tüm uzmanlar çalışan ürünün teslim edilmesinden sorumludurlar. Yani sorumluluk tüm takıma yüklenmiştir. Ürün sahibi ise işin içeriğini belirtmekten ve önceliklendirmekten sorumludur. Scrum yöneticisi plana uygun sürecin sürdürülüp sürdürülmediğini kontrol etmekten sorumludur.

GP2.5. İnsanları Eğit (+++)

Doğru eğitim yetenekli profesyonellerin performansını arttırır ve kurumun yeni yöntemleri benimsemesini hızlandırır. Bu genel pratik, gereken eğitimin ve eğitim ihtiyacı olan kişilerin belirlenmesi, eğitim gerçekleştirilmesini kapsamaktadır. Bu eğitimin gerçekleştirilmesinde programlanmış yönergeler, iş başında eğitim, bir uzmanla çalıştırma, kurs gibi yöntemler kullanılabilir. Eşli programlama ve koçluk bu süreçlerde etkilidirler.

GP2.6. Konfigürasyonları Yönet (++)

Bir proje genellikle kod, kılavuzlar, yazılım sistemleri gibi bir yığın iş ürünlerinden oluşur. Bu iş ürünlerinin her biri bir değerdir ve değerini arttıracak bir seri uygulamalara tabi tutulurlar. Bu değerlerin korunması ve kontrol edilmesi için mutlaka bir konfigürasyon yönetim sistemine ihtiyaç vardır. Bu tür sistemlerle kod, test, tasarım ve gereksinimlerin konfigürasyonları otomatik olarak yönetilebilir. Versiyon kontrol, dayanak kontrol gibi önemli denetimler gerçekleştirilebilir. Fakat test durumları, ölçüm verileri, sürüm ve iterasyon plan protokolleri için konfigürasyon yönetimi planlanmaz.

GP2.7. İlgili paydaşları belirle ve dahil et (+++)

Çevik yöntemlerde tüm ilgili paydaşlar takımın bir parçasıdır ve bu ilişki çevik yöntemlerin en güçlü taraflarından biridir. Bu genel pratik ilgili paydaşların yeterli katılımını tanımlar. Örneğin proje her iterasyonda ya da sprintte müşterinin geri beslemesine ihtiyaç duyuyorsa müşteriye yakınlık önemlidir. Tüm süreçlerde müşteri ve tedarikçi gibi paydaşların geliştirme çalışmalarına eşlik etmeleri sağlanmalıdır. Bazı gelişmiş Scrum uygulamalarında paydaşlar ürün sahibine hizmet veren yönetici kurulu gibi yapılandırılarak[13] etkinlikleri arttırılır.

GP2.8. Süreci izle ve kontrol et (++)

Bu genel pratik güncel performansın ölçülmesi ve gerekli düzeltici faaliyetlerin gerçekleştirilmesini kapsar. Kendi kendine organize olabilen takımlar, günlük ilerleme izleme toplantıları ve takım üyelerinin yoğun iletişimi çevik pratikleri ile bu genel pratik sağlanır. Bunun sadece ilgili süreç alanlarında değil tüm projede kullanılması için süreçlerin güncel performans izleme ölçümleri ortaya konulmalıdır. Günlük Scrum

toplantıları, Sürüm kalan zaman grafiği, Sprint kalan zaman grafiği gibi Scrum kavramları projenin günü gününe izlenmesi ve kontrol edilmesi açısından etkilidir. Testlerle kontrol otomatikleştirilebilir. Tüm bu pratikler doğru uygulandığı takdirde plana uyumluluk ve geri dönen fayda (ROI) artacaktır.

GP2.9. Nesnel olarak süreci değerlendir (+)

Bu pratiği XP'de destekleyecek şey koçtur. Scrum'da da Scrum yöneticisi süreçleri değerlendirmekten sorumludur. Ancak XP ile çelişmeyen "süreç ve ürün kalite güvencesi" süreç alanının tamamlanması ile bu pratiğin tüm süreç alanlarının ilgili gereklerini karşılaması mümkün olacaktır. Bunun yanında Ürün sahibi yazılımın gereksinimleri karşıladığından ve yüksek kalitede olduğundan emin olmakla sorumlu ilk kişidir.

GP2.10. Durumları üst yönetimle gözden geçir(++)

Bu genel pratiğin amacı üst düzey yöneticilerin proje aktivitelerinden haberdar olmasını sağlamaktır. Sıkça oluşturulan sürümler yönetimin projeyi gözden geçirmesinde çok etkilidir. Fakat her yönetici farklı bilgi ihtiyacı duyar. Bunun içinde yoğun bir etkileşim şarttır. Mesela Sprint planlama toplantıları, Günlük Scrum toplantıları, Sprint gözden geçirme toplantıları, Sprint sonu toplantıları, mevcut durumu gösteren Scrum geri kalan zaman grafiği bu etkileşimi sağlar. Bu etkileşimde yönetimin görevi stratejik bir vizyon, iş stratejisi ve kaynak sağlamak, takımların kendi başlarına kaldıramadıkları engelleri ortadan kaldırmak, büyümeyi sağlamaktır.

GP3.1. Tanımlanmış bir süreç oluştur (0)

Bu pratik GP2. genel pratiğinin arıtılmış halidir. Tek fark burada süreç tanımının bir proje için değil kurumsal genişlikte düşünülmesidir. Projeler arasında kişilerin araçların, bilginin değiştirilmesi ile bu pratik desteklenebilir ve süreç tüm kuruma yaygınlaştırılabilir.

GP3.2. Geliştirme bilgilerini topla (-)

Gelişmeler çevik yöntemler tarafından dokümante edilmez ve bundan dolayı bu genel pratik yerine getirilemez. Bu çelişki bir projedeki süreç değişikliklerinin uygun şekilde

belgelenmesi ve bunların kurumdaki diğer projeler tarafına da açık olmasının sağlanması ile çözülebilir. Buna ek olarak, süreç geliştirme bilgisi iterasyon planlama süresince ve çözümleme son durum incelemesi ile kolayca anlık resmedilebilir. Sprint sonu toplantıları ile bu pratiğin gerçekleştirilmesi için bir mekanizma oluşturulabilir.

8.3. CMMI ve Çevik Geliştirme Eşleme Sonuçları

CMMI süreç alanlarının Scrum ve XP çevik pratikleri ile ne düzeyde birlikte çalışabilirliğini detaylı analiz ettik. Scrum çevik geliştirmenin daha çok proje yönetim tarafıyla, XP geliştirme tarafıyla ilgilendiğinden bu iki modelin CMMI süreçlerine ve pratiklerine farklı katkılar sağladığını gördük. Bazı süreç alanlarının iki çevik yöntemle de çelişkili olduğunu, ama birçok sürecin ve özel pratiklerinin bu yöntemler tarafından yerine getirildiğini belirledik. CMMI 3 seviyesine kadar büyük bir uyum ve paralellik var iken kurumsallaşma kavramının iyileştirme ile birleştiği ve yoğunlaştığı 4. ve 5. olgunluk seviyelerinde ise çelişkilerin olduğunu farkedildi.

Bu çelişkiler de çevik yöntemlerin proje odaklılığının genişletilerek kurum perspektifine kavuşturulmasıyla azaltılabilir.

Tablo 8.1 - CMMI süreçlerinin XP ve Scrum pratikleri ile uyumluluğu

Süreç alanı	**XP**	**Scrum**
2.1 Gereksinim yönetimi	+++	+++
2.2 Proje planlama	+++	+++
2.3 Projenin izlenmesi ve kontrolü	+++	+++
2.4 Tedarikçi sözleşme yönetimi	0	0
2.5 Ölçme ve analiz	+	+++
2.6 Süreç ve ürün kalite güvencesi	+	0
2.7 Konfigürasyon yönetimi	+++	0
3.1 Gereksinim geliştirme	++	++
3.2 Teknik çözüm	+++	0
3.3 Ürün bütünleştirme	+++	0
3.4 Doğrulama	+++	0
3.5 Geçerleme	+++	+++
3.6 Organizasyonel süreç odağı	-	-
3.7 Organizasyonel süreç tanımı	0	0
3.8 Organizasyonel eğitim	++	+
3.9 Bütünleşik proje yönetimi	++	+++
3.10 Risk yönetimi	+++	+++
3.11 Bütünleşik takımlama	+++	+++
3.12 Entegre tedarikçi yönetimi	0	0
3.13 Karar analiz ve çözümleme	-	-
3.14 Entegrasyon için organizasyonel çevre	+	+
4.1 Organizasyonel süreç performansı	-	-
4.2 Nicel proje yönetimi	-	-
5.1 Organizasyonel yenilik ve geliştirme	-	-
5.2 Nedensel analiz ve çözümleme	0	0

Tablo 8.2 - CMMI genel pratiklerinin XP ve Scrum pratikleri ile uyumluluğu

Genel pratikler	**XP**	**Scrum**
2.1 Kurumsal bir politika oluştur	0	0
2.2 Süreci Planla	0	0
2.3 Kaynakları Sağla	+	+++
2.4 Sorumlulukları Ata	+++	+++
2.5 İnsanları Eğit	+++	+++
2.6 Konfigürasyonları Yönet	++	0
2.7 İlgili paydaşları belirle ve dahil et	+++	+++
2.8 Süreci izle ve kontrol et	++	++
2.9 Nesnel olarak süreci değerlendir	+	0
2.10. Durumları üst yönetimle gözden geçir	++	+++
3.1 Tanımlanmış bir süreç oluştur	0	0
3.2 Geliştirme ve iyileştirme bilgilerini topla	-	-

9.BÖLÜM

SONUÇ VE ÖNERİLER

Yazılım kalite güvencesi ve sertifikasyonu yazılım şirketlerinin stratejilerinden biri haline gelmiştir. Bunun en önemli sebepleri global markete açılabilmek için uluslar arası bir imaja sahip olmak ve projeleri daha etkili yönetecek birimler oluşturmaktır. Dünyada ve Türkiye`de kalite standartlarının öneminin artması sonucu telekom, askeri ihaleler gibi büyük projelerde CMMI L3, AQAP 160 v.b. kalite belgeleri istenir olmuştur. Bu nedenle Yurtiçi ve yurtdışı yazılım projelerinde teklif verebilmesi için gerekli olan CMMI L3 ve diğer kalite yeterlilik belgelerini en kısa zamanda alması gerekmektedir. Bu belgeler sadece bazı büyük ihalelere girmek için değil, aynı zamanda şirketin verimliliği ve müşteri memnuniyeti açısından da hayati önem taşımaktadır.

Bunun yanı sıra hızla artan bir şekilde yazılım geliştirme organizasyonlarının birçoğu yazılım süreçlerini genelindeki çevik yöntemlere adapte etmeye başladılar. Çünkü dünya yazılım sektörünün çoğunluğunu küçük ve orta büyüklükte yazılım şirketleri oluşturuyor. Bu şirketler daha az başlangıç yatırımı gerektirdiği için daha çok çevik yöntemleri benimsiyorlar. CMMI ile çalışan yerler bile çeviklik sağlayacak gelişmeler gerçekleştiriyorlar.

Aslında CMMI ve Çevik Geliştirme modelleri yazılım projelerinde bir arada kullanıldığında iyileştirilmiş, hızlı ve müşterinin ihtiyaçlarını karşılayan yazılımlar elde edilmiş, bu birlikteliğin proje geliştirme sürecine katkısı kanıtlanmıştır. CMMI'ın beraberinde getirdiği dokümantasyon uğraşı ve Çevik modellerle ortaya çıkabilen disiplinsizlik minimuma inmiştir. Günümüzde CMMI L4 ve L5 derecelerine çok kısa sürede gelen ve SEI'i bile bu konuda şaşırtan, çevik süreçleri kullanmasıyla tanınan şirketler (Boldtech, British Aerospace Industries gibi) ortaya çıkmıştır. Bu şirketler CMMI ve Çevik geliştirme birlikteliğini bir yatırım olarak görüyorlar. Çünkü çevik

geliştirme onlara tam müşteri memnuniyeti[19] ve hızlı kaliteli yazılım[20,21] vaadediyor. CMMI ise süreçlerini standartlaştırıp iyileştiriyor. SEI'de CMMI çalışmalarını bu yönde yoğunlaştırmış ve bu iki paradigmanın birlikte kullanımı yönünde çalışmalar yapmaktadır. SEI CMMI teknik ekibinin kıdemli üyesi Mike Konrad yazılım mühendisliği açısından sağlıklı olacağı düşünülmeyen bu birlikteliğe "İki toplulukta giderek büyüdüğünden bu konuyu tartışmanın en doğru zamanıdır. Bu iki popüler yöntem hakkında bilinen efsaneler yerine doğru bir görüş oluşturmak için iyi bir fırsat yakalanmıştır.[30]" diyerek özel ilgisini belirtmiştir. Aynı ekipten Jeffrey Dalton'a göre ise CMMI ve çevik geliştirmenin gelişmesini farklı şeyler tetikliyor. Yani CMMI kurumsal bir ihtiyaç güdümlü ve yukarıdan aşağı bir süreç izlerken, çevik yöntemler uyarlama süreçlerinde organik ve dinamik bir yapı gösteriyor. Dalton "Önceleri bu iki yöntemin aynı ortamda birarada olamayacağına dair bir yanlış algılama vardı. Fakat zamanla ortaya çıktı ki; iki yaklaşım birlikte işleyebilmektedir." diyerek bu iki yaklaşım arasında bir karşılıklı etkileşim ve değişim yapılarak faydanın arttırılacağına dikkat çekiyor. Hatta bu iki yaklaşımın yüksek düzeyde uyumluluğuna kendisi de şaşırıyor. Ayrıca çevik yöntemleri benimseyen topluluğunun CMMI ve çevik geliştirme birlikteliğine sıcak bakmadığını ve bu yöntemlerden çıkacak faydalara erişmek için öncelikle çevik süreç topluluğunun CMMI hakkında bilgi edinmesi gerektiğini belirtiyor. Konrad bu yaklaşımı bir adım daha ileri götürerek bu konuda yapılan çalışmaları "SEI çalışanları bizler için ortalıkta dolanan uyumsuzluk efsanelerini ayrıştırmak ve çevik topluluktan özür dilemek için bir fırsat." olarak tanımlıyor ve iki yöntemin kesinlikle uyumlu olduğunu belirtiyor.

Bu gelişmelere rağmen hala çevik süreçler ve CMMI modelinin iki farklı yaklaşımı temsil ettiği düşünülmektedir. Fakat CMMI ve çevik süreçler farklı seviyelerde detaylı incelenmesi gereken yaklaşımlardır[7].

CMMI proje seviyesinde ne yapılacağına yoğunlaşır. Çevik yöntemler ise projelerine nasıl geliştirileceği ile ilgilenir. CMMI süreç iyileştirme için kullanılabilecek bir modeldir ve modelin gereksinimlerini nasıl karşılayacağınız konusunda CMMI sizi belli bir yöntemle kısıtlamaz. CMMI büyük projelerde çevik yaklaşımı uygulamaya yardım edecek yazılım mühendisliği pratikleri sağlarken, çevik yöntemler ise CMMI pratiklerinin kaçırdığı yazılım geliştirmenin nasıl olacağı bilgisini sunmaktadır.

CMMI modeline uygun bir süreç uygulamasını çevik süreçleri etkin kullanarak yapabileceğiniz gibi reçete şeklinde oluşturulmuş, organizasyon yapınıza uygun olmayan ve etkinliği sınırlı bir süreçle de yapmak mümkündür. Aynı CMMI derecelendirmesine sahip iki şirketin iş yapış biçimleri, organizasyon yapıları, verimlilikleri ve proje teslim kabiliyetleri birbirlerinden çok farklı olabilir. Bunun yanı sıra CMMI uyumlu bazı büyük şirketler ayrıca çevik geliştirme takımlarına sahiptirler. Yine bazı şirketler iteratif ve zaman kısıtlı çevik yaklaşım benimseyen projelerinde CMMI gereklerini uygulayabilmektedir.

Çevik süreçlerin prensipleri ile çelişen durum CMMI modeli değil CMMI derecelendirmesi ile ilgili konularda şirketlerin çoğu zaman yaptıkları yanlış uygulamalar ve bu konuda yerleşmiş kültürdür.

CMMI modeline bağlı olarak süreç iyileştirmesi yapan organizasyonlarda modelin aradığı süreç alanlarına dair kanıtların çoğunlukla dokümantasyon olarak yorumlandığını ve uygulandığını görürüz. Sonuç olarak ağır dokümantasyon yükü altında organizasyon hantallaşır ve değer sağlayabilme niteliğini kaybeder. Süreç ekibin doğal çalışma ritmi olmaktan çıkar ve bürokratik bir hal alır ve organizasyona gerçek yarar sağlayamaz.

Çevik süreçler çalışan yazılımı kapsamlı dokümantasyona yeğ tutar ve sadece gerektiği kadar dokümantasyon oluşturmaya gayret eder. Çevik süreçler konusunda yanlış inanışlardan birisi hiç dokümantasyon oluşturmadan sadece yüz yüze iletişime dayandığıdır. Fakat herhangi bir çevik ekibin ofis ortamına girdiğinizde oluşturulan ve sürekli canlı tutulan dokümantasyonlar hemen gözünüze çarpacaktır.

Bir başka konu çevik süreçler dendiğinde sürekli Extreme Programming'in akla gelmesidir. Extreme Programming ekip olarak etkin yazılım geliştirme pratikleri sunar. Scrum ise proje yönetimine odaklanır. XP ve Scrum'in sentezi bir süreç CMMI L3'ün gereksinimlerini karşılayabilmektir ve organizasyona en yararlı olan süreç alanları CMMI L3 seviyesine kadar olanlardır.

CMMI modelli süreç iyileştirme ile ilgili günümüzde şirketlerin en büyük yanlışlardan biri süreç iyileştirme çalışmaları sırasında süreç alanlarına ait detaylarına odaklanmaları ve büyük resmi gözden kaçırmalarıdır. Sürecin ana amacı CMMI'a uygunluk değil yazılımı kullanmayı bekleyen müşteriye değer katabilmesidir. Optimize edilmiş süreç adımlarının entegrasyonu sonucu verimli bir süreç yapısına ulaşmak mümkün olmayabilir. Çevik süreçler bu konuda bütüne odaklanır ve ana amaç olarak müşteriye 2-4 hafta arası aralıklarla sürekli yazılım teslimleri yaparak değer katmayı amaçlar. Tüm pratikler bu ana amaca uygun olacak şekilde organize olur. Örneğin sürekli entegrasyon pratiği sayesinde proje her an teslime hazır halde tutulur.

Problemlere neden olan bir başka konu organizasyonların CMMI derecelendirmesine çabuk kavuşmak adına baştan belli hazır süreç paketleri ve araçları alıp uyarlama yoluna gitmeleridir. Bu yöntem maliyetleri arttırdığı gibi başarı şansı kullanılacak süreçlerin organizasyonla ne kadar örtüştüğü, ne kadar kolay adapte edilebileceği ile sınırlıdır.

Çevik süreçler ise disiplinli yazılım geliştirebilmek için gerekli çekirdek pratiklerden başlayarak organizasyonun kültürüne ve projelere göre süreci sürekli geliştirerek ve iyileştirerek organik olarak oluşturmayı tercih eder. Extreme Programming pratiklerinden daha kompleks DSDM (Dynamic Systems Development Method) gibi süreçlere kadar bir evrim izlenebilir. Sonuç olarak organizasyon yapısıyla birebir uyumlu süreçlere ulaşılır.

CMMI'in hedefi olan tekrarlanabilir ve sonuçları önceden tahmin edilebilir olgunlukta olan süreçler çevik süreçlerin uygulanması ile başarılabilir ve günümüzde bu konuda yaşanan problemlere çözüm olabilir .

CMMI bir organizasyonu süreç alanlarıyla birlikte bir bütün olarak değerlendirir. Çevik geliştirme ise bir süreç için değerlendirme yapar. Bu yüzden içeriklerinin karşılaştırılması çok doğru olmaz. Fakat yoğunlaştıkları alanlar farklı olsa da aralarında güçlü bir ilişki vardır. M.C.Paulk[27] CMMI'yı "yazılım yönetimi metodu" olarak isimlendirirken, çevik geliştirmeye de "yazılım geliştirme metodu" der. Birarada bulunmasalar bile birbirilerini

destekleyen yapılar olarak kullanılabilirler. CMMI süreçleri çevik yöntemlerle geliştirilecek yapıdadır[28] .

Düşünüldüğünün aksine, çevik yöntemler sadece küçük takımlar ve kapsamı dar projeler için değildir. A. Cockburn ve J. Highsmith 250 kadar kişinin çalıştığı çevik projelerin başarısını göstermişlerdir[15]. Ayrıca dış kaynaklardan edinilen (outsourcing) ve destekleyici (off-shoring) projelerde de başarı ile uygulanabilmektedir[16,17].

Ayrıca CMMI uyumluluk sürecinde CMMI pratiklerine çevik yöntemlerin uygulamalarını adapte etmek zaman ve emek harcayan CMMI gereklerinin daha hızlı ve kolay karşılanmasını sağlayabilir.

Yazılım sektörünün hızla geliştiği ülkemizde bu iki kavram çok yeni olsa da her iki alana da ilgi artmıştır. Kamu ihalelerinde doğru firmayı seçmek için CMMI Seviye 3 şartı aranır hale gelmiştir. Milli Savunma Bakanlığı'na bağlı Savunma Sanayi Müsteşarlığı 2007 yılından itibaren yapılacak bütün ihalelerinde CMMI 3 serfitikasını isteyeceğini açıklamıştır. Türkiye'de biri 5. Seviye olmak üzere 9'un üzerinde firma CMMI sertifikasına sahiptir ve projelerini buna uygun olarak geliştirmektedir. Yine onlarca firma da bu sertifikanın bünyelerine kazandırılması için çalışmaya devam etmektedirler. Emek ve zaman isteyen bu süreçte çevik yöntemlerin kullanılması yazılım firmalarımıza hem tasarruf sağlayacaktır, hem de sertifikanın edinimini ve sonrasında projelerde uygulanmasını hızlandıracaktır.

Akılda tutulması gereken şudur; CMMI bir yazılım geliştirme süreci değildir. CMMI etkin ve verimli bir sürecin tipik özelliklerini ana hatlarını tanımlar. Süreç iyileştirmesi için bir temel sağlar. Özel ve geçici olarak kurulmuş, olgunlaşmamış bir süreçten, olgun disiplinli bir sürece evrimsel bir iyileştirme yolu çizer. CMMI süreç iyileştirme için kullanılabilecek modelin gereksinimlerinin nasıl karşılanacağı belli bir yöntemle kısıtlanmamaktadır. Yani süreçlerin iyileştirilmesi için ne yapılması gerektiğini söylerken bunların nasıl yapılacağına dair detaylara girmez. Bu modele uygun bir süreç uygulaması, çevik yöntemlerin etkin kullanımıyla da yapılabilir. CMMI modelinin en önemli özelliğinin ağır dokümantasyon olduğu, süreç ekibinin doğal çalışma ritminin de bu ağır dokümantasyonu sağlayacak hantallıkta olması gerektiği düşünülür. Oysa çevik

yöntemlerle de (minimum L3 seviyesine kadar) gerektiği kadar ve canlı bir dokümantasyon oluşturulabilir. Şuanda sadece dünyada değil Türkiye'de bu eğilimi destekleyen görüşler vardır. Dünya genelinde çevik süreçler konusunda danışmanlık veren ThoughtWorks şirketinde Uzman Danışman olarak çalışmış Cenk Çivici ve şuanda Türkiye'de çevik geliştirme danışmanlığı yapan bir firmanın sahibi Tolga Gündüz bu görüşü benimsemektedir[1].

Umarım bu konuda yazılım sektöründe öncü bazı ülkelerde olduğu gibi CMMI ve Çevik yöntemler ayrımına gidilmez, daha sonra da bu modellerin entegrasyonu için çaba harcanmaz. Bu durum standartlaşmanın her alanda çok önem taşıdığı günümüzde ülkemizin yazılım sektörünün gelişmesine gözle görülür yavaşlatıcı etkiler oluşturabilir.

KAYNAKLAR

1. Aktaş, M.Ş., Çevik Yöntemler ve Süreç Bazlı Yöntemlerin Arasındaki Çelişki, http://www.ipyd.org/bolum/149/cevik-yontemler-ve-surec-bazli-yontemlerin-arasindaki-celiski, 2008.
2. Cohn, M., Agile Estimating and Planning, Prentice Hall, 2006.
3. Kniberg, H., Scrum and XP from the Trenches, InfoQ, 2007.
4. Martin, R.C., Agile Development: Principles, Patterns, and Process, Prentice Hall, 2003.
5. SEI CMMI Product Team, CMMI for Development Version 1.2, Software Engineering Institute, Carnegie Mellon University, 2006.
6. Kalaycı, O., CMMI:Yöneticiler için Doğru Sorular, Shamrock Process Improvement and Innovation, 2007.
7. Çivici, C., Çevik süreçler ve CMMI, http://cenkcivici.com/blog/?p=43, 2007.
8. Fritzsche, M., Keil, P., Agile Methods *and.* CMMI*:* Compatibility or Conflict, e-Informatica. Software Engineering Journal, vol. 1, 2007.
9. Chrissis, M.B., Konrad, M.,Shrum, S., CMMI: Guidelines for Process Integration and Product Improvement, Addison-Wesley, 2003.
10. Elshafey, L.A., Galal-Edeen, G.H., Combining CMMI and Agile Methods, The Sixth Annual Conference on Informatics and Systems(INFOS), Cairo University, Cairo, March 27-29,2008.
11. Sutherland, J., Jacobson, C., Scrum and CMMI Level 5: A Magic Potion for Code Warriors, Agile 2007 Conference, Washington, D.C., IEEE. 2007**.**
12. Stuart, J., Optimizing Agile for Your Organization, http://www.construx.com/whitepapers, 2008.
13. Sutherland, J., Future of Scrum: Parallel Pipelining of Sprints in Complex Projects, Agile 2005 Conference, Denver, CO, 2005.
14. SEI CMMI Product Team, CMMI Version 1.1 (CMMI-SE/SW/IPPD/SS, V1.1), Software Engineering Institute, Carnegie Mellon University, 2002.
15. Cockburn, A., Highsmith, J., Agile Software Development: The People Factor. IEEE Computer, 34(11):131–133, 2001.

16. Sangwan, R.S., Masticola, S.P., Model-Driven Rapid Application Development: A Framework for Agile Development in Outsourced Environments, Siemens Corporate Research, 2004.
17. Fowler, M., Using an Agile Software Process with Offshore Development, http://www.martinfowler.com/articles/agileOffshore.html, 2005.
18. Fritzsche, M., Agile Methoden im industriellen Umfeld, Technische Universit at Munchen, 2005.
19. Boehm B., Turner, R., Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley, 2003.
20. Highsmith, J., Agile Project Management: Creating Innovative Products, Addison-Wesley, 2004.
21. Anderson, D. J., Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results, Prentice Hall, 2003.
22. Kulpa, M.K., Johnson, K.A., Interpreting the CMMI: A Process Improvement Approach, CRC Press, 2008.
23. Acar, Ö., Extreme Programming, http://www.kurumsaljava.com, 2008.
24. Johnson, D.L., Brodman, J.G., Applying CMM Project Planning Practices to Diverse Environments, IEEE Software, 17 (4), 40–47, 2000.
25. Turner, R., Jain, A., Agile Meets CMMI: Culture Clash or Common Cause, the Second XP Universe and First Agile Universe Conference, Chicago, USA, August 4-7, 2002.
26. Kähkönen, T., Abrahamsson, P., Achieving CMMI Level 2 with Enhanced Extreme Programming Approach, the 5th International Conference on Product Focused Software Process Improvement, Kansai Science City, Japan, April 5-8, 2004.
27. Paulk, M.C., Extreme Programming from a CMM Perspective. IEEE Software, 18(6):19–26, 2001.
28. Glazer, H., Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity. CrossTalk: The Journal on Defense Software Engineering, (14):27–30, 2001.
29. Glazer, H., et al., CMMI or Agile: Why Not Embrace Both, SEI, 2008
30. ExecutiveBrief Team, CMMI and Agile: Opposites Attract, http://www.executivebrief.com/article/cmmi-and-agile-opposites-attract/, 2008

31. Beck, K., Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.
32. Schwaber, K., The Scrum development process, OOPSLA ’95, Texas, USA, October 1995.
33. Jacobson, I., et al., The Unified Software Development Process. Addison-Wesley, 1999.
34. Ambler, S., Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, Wiley Computer Publishing, 2002.
35. Gilb, T., Principles of Software Engineering Management, Addison-Wesley, 1988.
36. Cockburn, A., Agile Software Development, Adisson-Wesley, 2002.
37. Palmer, S.R., Felsing, J.M., A Practical Guide to Feature-Driven Development, Prentice Hall, 2002.
38. Stapleton, J., Dynamic Systems Development Method - The method in practice, Addison-Wesley, 1997.
39. Kueng, P., Process performance measurement system: a tool to support process-based organizations, Total Quality Management, Vol. 11, No. 1, 2000.

ÖZGEÇMİŞ

İbrahim Halil SARAÇOĞLU, 1980 yılında Şanlıurfa'da doğdu. 1997 yılında Şanlıurfa Davud Zeki Akpınar Lisesi'ni ve 2002 yılında Ege Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği bölümünü tamamladı. 2002-2005 yılları arasında İstanbul ve Ankara'da çeşitli yazılım şirketlerinde "Yazılım Uzmanı" olarak çalıştı. 2005 yılında "Sözleşmeli Subay" olarak Hava Kuvvetleri'ne katıldı. 3 yıl "Hava Savunma Subayı" olarak görev yaptıktan sonra 2008 yılında Türk Silahlı Kuvvetleri'nden ayrıldı ve Oyak Teknoloji'de "Uzman Yazılım Mühendisi" olarak çalışmaya başladı. Halen bu görevine devam etmektedir.

E-posta: saracogl@gmail.com