

T.C.
MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇEVİK YAZILIM GELİŞTİRME İLKELERİNİN
PROJE SÜREÇLERİNE ETKİLERİ

Sinan YILDIRIM

YÜKSEK LİSANS TEZİ
ELEKTRONİK-BİLGİSAYAR ANABİLİM DALI
BİLGİSAYAR-KONTROL PROGRAMI

DANIŞMAN
Yrd. Doç. Dr. Şemsettin KILIÇARSLAN

İSTANBUL, 2010

T.C.
MARMARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇEVİK YAZILIM GELİŞTİRME İLKELERİNİN
PROJE SÜREÇLERİNE ETKİLERİ

Sinan YILDIRIM
(141522120089012)

YÜKSEK LİSANS TEZİ
ELEKTRONİK-BİLGİSAYAR ANABİLİM DALI
BİLGİSAYAR-KONTROL PROGRAMI

DANIŞMAN
Yrd. Doç. Dr. Şemsettin KILIÇARSLAN

İSTANBUL, 2010

ÖNSÖZ

Tez konumun belirlenmesi aşamasından sonlandırılmasına kadar değerli katkılarıyla beni yönlendirerek yardımcı olan tez danışmanım Yrd. Doç. Dr. Şemsettin KILIÇARSLAN'a teşekkürlerimi sunarım.

Çevirilerde yardımda bulunarak destek veren kardeşim Ünal YILDIRIM'a, sabır ve anlayışlarından dolayı aileme teşekkür ederim.

Sinan YILDIRIM

İÇİNDEKİLER

	SAYFA
ÖNSÖZ	i
İÇİNDEKİLER	ii
ÖZET	vi
ABSTRACT	vii
KISALTMALAR.....	viii
ŞEKİLLER	ix
TABLOLAR	xii
BÖLÜM I	1
GİRİŞ ve AMAÇ	1
I.1. GİRİŞ ve AMAÇ	1
BÖLÜM II.....	2
YAZILIM MÜHENDİSLİĞİ	2
II.1. YAZILIM MÜHENDİSLİĞİ TANIMI	2
II.2. YAZILIM MÜHENDİSLİĞİ TARİHÇESİ.....	2
II.3. ALT DİSİPLİNLER.....	5
II.3.1. Yazılım Gereksinimleri.....	5
II.3.2. Yazılım Tasarımı.....	6
II.3.3. Yazılım Geliştirme	6
II.3.4. Yazılım Testi	6
II.3.5. Yazılım Bakımı	7
II.3.6. Yazılım Konfigürasyon Yönetimi.....	7
II.3.7. Yazılım Mühendisliği Yönetimi	8
II.3.8. Yazılım Geliştirme Süreci.....	8
II.3.9. Yazılım Mühendisliği Araçları	8
II.3.10. Yazılım Kalitesi	8
BÖLÜM III	11
YAZILIM GELİŞTİRME METODOLOJİLERİ	11

III.1. YAZILIM GELİŞTİRME	11
III.2. YAZILIM GELİŞTİRME SÜREÇLERİ	11
III.3. YAZILIM GELİŞTİRME METODOLOJİLERİ	14
III.3.1. Şelale Süreci (Waterfall Process)	16
III.3.2. Artımlı Geliştirme (Incremental Development)	19
III.3.3. Evrimsel Geliştirme (Evolutionary Development)	20
III.3.4. İteratif Geliştirme (Iterative Development)	20
III.3.5. Sarmal Model (Spiral Model)	22
III.3.6. Prototipleme (Prototyping)	23
III.3.7. Çevik Metotlar	23
BÖLÜM IV	25
ÇEVİK YAZILIM GELİŞTİRME	25
IV.1 ÇEVİK YAZILIM GELİŞTİRME TANIMI.....	25
IV.2. ÇEVİK SÜREÇ	26
IV.2.1. Çevik Sürecin Geçmişi	26
IV.2.2. Çevik Bildiri	26
IV.2.3. Çevik İlkeler	29
IV.2.4. Geleneksel Sürecin Problemleri.....	34
IV.2.5. Çevik Sürecin Yararları	35
IV.3. ÇEVİK SÜREÇ TÜRLERİ	39
IV.3.1. Ekstrem Programlama.....	40
IV.3.2. Scrum	40
IV.3.3. Dinamik Sistem Geliştirme Metodu	43
IV.3.4. Uyarlanabilir Yazılım Geliştirme	44
IV.3.5. Kristal.....	46
IV.3.6. Özellik Güdümlü Geliştirme.....	47
IV.4. ÇEVİK YAZILIM GELİŞTİRMEDE ÖLÇEKLENDİRME	49
IV.4.1. Çevik ve Takım Boyutu	50
IV.4.2. Çevik ve Coğrafi Dağılım.....	50
IV.4.3. Çevik ve Mevzuat Uyumluluk.....	51
BÖLÜM V	52
EKSTREM PROGRAMLAMA.....	52
V.1. EKSTREM PROGRAMLAMA TANIMI.....	52

V.1.1. Ekstrem Programlama Değerleri	52
V.1.2. Ekstrem Programlama İlkeleri.....	55
V.1.3. Ekstrem Programlama Uygulamaları	58
V.1.4. Ekstrem Programlama Takım Rollerini.....	64
V.1.5. Haklar ve Sorumluluklar	66
V.1.6. Ekstrem Programlama Proje Evreleri.....	68
V.2. PROJE YÖNETİMİ	70
V.2.1. Planlama Evresi	70
V.2.2. Sürüm Planlaması.....	70
V.2.3. Sürüm Planlama Oyunu.....	71
V.2.4. İterasyon Planlaması.....	76
V.2.5. Kabul Testleri	78
V.2.6. Dijital Hikaye Kartları.....	78
BÖLÜM VI	79
ÇEVİK MODEL TABANLI GELİŞTİRME	79
VI.1. ÇEVİK MODELLEME TANIMI	79
VI.2. ÇEVİK MODELLEME İLE MODEL OLUŞTURMA.....	80
VI.3. ÇEVİK MODELLEME UYGULAMALARI.....	81
VI.4. ÇEVİK MODEL TABANLI GELİŞTİRME YAPISI	82
BÖLÜM VII.....	85
ÇEVİRİMİÇİ ZAMAN ÇİZELGESİ SİSTEMİ.....	85
VII.1. GİRİŞ	85
VII.2. İŞ GEREKSİNİMLERİ	85
VII.3. YAZILIM GELİŞTİRME METODOLOJİSİ	86
VII.4. PROJEYE EP ve ÇMTG UYGULANMASI.....	88
VII.4.1. Etki Alanı Modeli.....	88
VII.4.2. Kullanıcı Arabirimi Prototipi	89
VII.4.3. Hikaye Panosu.....	93
VII.4.4. Kullanıcı Hikayeleri	94
VII.4.5. Sürüm ve İterasyon Planı	95
VII.4.6. Beyaz Panoda Mimari	95
VII.5. EP, ÇMTG TABANLI MİMARİ ve TASARIM MODELLEME ..	96
VII.5.1. Tasarım Yaklaşımı ve Yapı Seçimi.....	96

VII.5.2. Serbest Mimari Diyagramı	97
VII.5.3. CRC Kartları Kullanarak Sınıfları Keşfetmek	97
VII.5.4. Uygulama Akış Haritası	98
VII.5.5. UML Sınıf Diyagramı	99
VII.5.6. UML Paket Diyagramı	99
VII.5.7. Dizin Yapısı.....	100
VII.5.8. Kabul Testleri	101
VII.5.9. Kavramsal, Fiziksel ve Uygulama Katmanları	102
VII.6. ÇALIŞMA ORTAMININ KURULMASI	103
VII.6.1. Java Platformu İçin Startdart Geliştirme Kitinin Kurulması...	103
VII.6.2. Ant Aracı	105
VII.6.3. JUnit Aracı	108
VII.6.4. Eclipse IDE	111
VII.6.5. Apache Tomcat	112
VII.6.6. Spring MVC	113
BÖLÜM VIII	114
SONUÇLAR VE TARTIŞMA	114
BÖLÜM IX	116
SON DEĞERLENDİRMELER VE ÖNERİLER	116
KAYNAKLAR.....	117
EKLER.....	122
ÖZGEÇMİŞ.....	144

ÖZET

ÇEVİK YAZILIM GELİŞTİRME İLKELERİNİN PROJE SÜREÇLERİNE ETKİLERİ

Günlük hayatımızda yazılım birçok alanda kullanılmaktadır. Dev bir endüstri haline gelen yazılım sektöründe, projelerin başarıya ulaşmasında yazılım geliştirme süreçleri büyük önem taşımaktadır.

Proje üretiminde, geleneksel yazılım geliştirme süreçlerinin kullanımı yazılım projelerinde başarısızlığa sebep olabilmektedir. Geleneksel süreçlerde, müşteri gereksimlerini karşılayamama, zamanında ve bütçe içinde yazılım projesini teslim edilememesi gibi sıkıntılar mevcuttur. Bu durum yazılım geliştirme süreçlerinde farklı fikirlerin ortaya çıkmasına neden olmuştur.

Çevik yazılım geliştirme ilkeleri, yazılım proje süreçlerinde kuralları esnek hale getirmektedir. Bu yaklaşım, yazılım projelerinde firmalara dinamiklik, esneklik ve verimlilik getirmiştir. Ülkemizde çevik yazılım geliştirme ilkeleri halen yaygın olarak kullanılmamaktadır. Çevik İlkeler, Avrupa ve ABD’de çok büyük firmalar tarafından kullanılmaya başlanmıştır.

Bu çalışmada çevik yazılım geliştirme ilkeleri incelenerek proje üretimine etkileri araştırılacaktır. Örnek bir proje ile adım adım çevik sürecin nasıl işlediği incelenecektir.

Şubat, 2010

Sinan YILDIRIM

ABSTRACT

THE EFFECTS TO PROJECT PROCESSES OF AGILE SOFTWARE DEVELOPMENT PRINCIPLES

Software is used in many fields of our daily lives. Software development processes are essential to reach success in projects in the software sector which has become a giant industry.

Traditional software development processes in project production can cause failure in software projects. Some difficulties exist in traditional processes if they aren't receptive to customers requirements, or they can't deliver software projects to deadline and to budget. These conditions have appeared different ideas in project development processes.

Agile software development principles have to be flexible in meeting the conditions and rules of software project processes. This approach gives flexibility, productivity and dynamism to firms in software development projects. Agile software development principles are still not used widely in Turkey. They have started to be used by very large firms in Europe and the USA.

In this study, agile software development principles are going to be examined and the effects on project production will be examined. How this agile process will be shown step by step in a model project.

February, 2010

Sinan YILDIRIM

KISALTMALAR

CASE	: Computer Aided Software Engineering - Bilgisayar Destekli Yaz. Müh.
NATO	: North Atlantic Treaty Organization - Kuzey Atlantik Antlaşması Örgütü
CMM	: Capability Maturity Model - Yetenek Olgunluk Modeli
MDD	: Model Driven Development – Model Tabanlı Geliştirme
WWW	: World Wide Web – Dünya Çapında Ağ
UML	: Unified Modelling Language - Birleşik Modelleme Dili
IEEE	: Institute of Electrical and Electronics Engineers
SGYD	: Sistem Geliştirme Yaşam Döngüsü
EP	: Ekstrem Programlama
DSGM	: Dinamik Sistem Geliştirme Metodu
UYG	: Uyarlanabilir Yazılım Geliştirme
ÖGG	: Özellik GÜdümlü Geliştirme
BT	: Bilişim Teknolojileri
ÇM	: Çevik Modelleme
ÇMTG	: Çevik Modelleme Tabanlı Geliştirme
JSP	: Java Server Pages - Java Sunucu Sayfaları
HTTP	: HyperText Transfer Protocol - Hipermetin Aktarma İletişim Kuralı
MGD	: Model Görünümlü Denetleyici
JDK	: Java Development Kit - Java Geliştirme Kiti
XML	: eXtensible Markup Language - Genişletilebilir İşaretleme Dili
CRC	: Class Responsibility Collaborator - Sınıf Sorumluluk İşbirlikçi

ŞEKİLLER

SAYFA NO

Şekil II.1 Yazılım Kalite Özniteliklerinde İlişkilerin Gösterilmesi – CUPRIMDA ...	9
Şekil III.1 Yazılım Geliştirme Süreçlerine Göre Yaklaşık Maliyet Oranı	13
Şekil III.2 Yazılım Geliştirme Metodolojilerinde Kullanılan Üç Temel Süreç	16
Şekil III.3 Geleneksel Şelale Süreci	16
Şekil III.4 Süreçlerde Ürünlerin Dönüşümü	17
Şekil III.5 Şelale Modeli Süreçleri ve Ürünler	18
Şekil III.6 Programlama ve Tekrarlama Süreçleri	19
Şekil III.7 Artımlı Geliştirme Modeli	19
Şekil III.8 Evrimsel Geliştirme Modeli	20
Şekil III.9 İteratif Geliştirme Modeli	21
Şekil III.10 İteratif Geliştirme	21
Şekil III.11 Sarmal Model	22
Şekil III.12 Prototipleme Modeli	23
Şekil IV.1 Çevik Değerler, İlkeler ve Uygulamalar Arasındaki İlişki	26
Şekil IV.2 Çevik ve Geleneksel Süreç	35
Şekil IV.3 Çevik Süreç Değer İfadeleri	36
Şekil IV.4 Genelleştirilmiş Çevik Süreç Modeli	39
Şekil IV.5 Ekstrem Programlama Görsel Süreç Modeli	40
Şekil IV.6 Scrum Görsel Süreç Modeli	43
Şekil IV.7 DSGM Görsel Süreç Modeli	44
Şekil IV.8 UYG Görsel Süreç Modeli	45
Şekil IV.9 Kristal Metodoloji Boyutları	46
Şekil IV.10 Özellik GÜDÜMLÜ Geliştirme Görsel Süreç Modeli	48
Şekil IV.11 Çevik Takımların Ortalama Boyutu	50
Şekil IV.12 Çevik Yazılım Geliştirme ve Coğrafi Dağılım	51
Şekil IV.13 Çevik Yazılım Geliştirme ve Mevzuat Uyumluluk	51
Şekil V.1 Ekstrem Programlama Uygulamaları	58
Şekil V.2 Ekstrem Programlama Zaman Döngüsü	68

Şekil V.3 Ekstrem Programlama Proje Evreleri	68
Şekil V.4 Ekstrem Programlama Proje Evreleri	70
Şekil V.5 Sürüm Planlama Oyunu	71
Şekil V.6 Kullanıcı Hikaye Kartı.....	72
Şekil V.7 Planlama Poker Kartları.....	72
Şekil V.8 Ekstrem Programlama İterasyon ve Sürüm	76
Şekil V.9 İterasyon Plan Toplantısı	76
Şekil V.10 Görev Kartlarının Kullanıcı Hikayesine Göre Gruplanması	77
Şekil V.11 Ekstrem Programlama İterasyon Görsel Modeli	77
Şekil VI.1 Çevik Modellemenin Diğer Yazılım Süreçlerini Geliştirmesi	79
Şekil VI.2 Çevik Modelleme Uygulamaları	81
Şekil VI.3 ÇMTG Yaşam Döngüsü - Proje Süresince Faaliyetlerin Modellenmesi .	83
Şekil VI.4 Çevik Gereksinim Değişimi Yönetim Süreci.....	84
Şekil VII.1 Zaman Çizelgesi Etki Alanı Modeli	89
Şekil VII.2 Zaman Çizelgesi Oturum Giriş Ekranı	90
Şekil VII.3 Zaman Çizelgesi Liste Ekranı.....	90
Şekil VII.4 Zaman Çizelgesi Saat Giriş Ekranı.....	91
Şekil VII.5 Zaman Çizelgesi Saat Çıktı Ekranı.....	91
Şekil VII.6 Zaman Çizelgesi Onaylama Ekranı	91
Şekil VII.7 Zaman Çizelgesi Ödeme İşareti Ekranı	92
Şekil VII.8 Zaman Çizelgesi Kullanıcı Saatleri Ekranı.....	92
Şekil VII.9 Zaman Çizelgesi Genel Özet Ekran.....	93
Şekil VII.10 Zaman Çizelgesi Hikaye Panosu	93
Şekil VII.11 Beyaz Panoda Taslak Mimari	96
Şekil VII.12 Sürüm ve İterasyon Düzeylerinde EP/ÇMTG Tarzı Seçimler	96
Şekil VII.13 Zaman Çizelgesi İçin Üst Düzey Mimari	97
Şekil VII.14 Zaman Çizelgesi İçin Örnek Sınıf Diyagramı	99
Şekil VII.15 Zaman Çizelgesi İçin UML Paket Diyagramı	100
Şekil VII.16 Zaman Çizelgesi İçin Dizin Yapısı.....	100
Şekil VII.17 Zaman Çizelgesi için Kavramsal, Fiziksel ve Uygulama Katmanlar .	102
Şekil VII.18 Java Geliştirme Sisteminin İndirilmesi.....	103
Şekil VII.19 Java Geliştirme Sisteminin Kurulumu.....	104
Şekil VII.20 Java Geliştirme Sisteminde Sistem Yolu Tanımlama	104

Şekil VII.21 JDK Kurulum Test.....	105
Şekil VII.22 Ant Aracı Sistem Yolu Tanımlama	105
Şekil VII.23 Ant Aracı Kurulum Testi	106
Şekil VII.24 Basit Bir Ant Dosya Yapısı	106
Şekil VII.25 Ant Aracı İle İşlem Yapma	106
Şekil VII.26 Eclipse ile Junit Testi.....	109
Şekil VII.27 Eclipse ile Junit Testi.....	109
Şekil VII.28 Eclipse ile Junit Testi.....	110
Şekil VII.29 Eclipse ile Junit Testi.....	110
Şekil VII.30 Eclipse IDE Programını İndirme	111
Şekil VII.31 Eclipse IDE Çalışma Alanı.....	112
Şekil VII.32 Eclipse IDE Ant Build.xml Dosyası	112
Şekil VII.33 Tomcat Açılış Ekranı	113
Şekil VIII.1 Standish Grup - Yazılım Projeleri Başarı Tarihçesi.....	114

TABLolar

	<u>SAYFA NO</u>
Tablo II.1 Yazılım Kalite Öznitelikleri.....	10
Tablo III.1 Belirli Yazılım Geliştirme Metodolojileri.....	15
Tablo III.2 Şelale Modeli Ürünler ve Süreçler	17
Tablo IV.1 Çevik Değerler.....	27
Tablo IV.2 Çevik İlkeler	29
Tablo V.1 Kullanıcı Hikayeleri ve Değerlilik Sırası	74
Tablo V.2 Sürüm Planı	75
Tablo VI.1 Çevik Modelleme İlkeleri	80
Tablo VII.1 Örnek Proje Kapsam Tablosu	88
Tablo VII.2 Kullanıcı Hikayeleri, Öncelikler ve Tahminler.....	94
Tablo VII.3 Sürüm Plan 1.0	95
Tablo VII.4 Örnek Bir CRC Kart Düzeni.....	98
Tablo VII.5 Zamancizelgesi Sınıfı İçin Örnek CRC Kart	98
Tablo VII.6 ZamancizelgesiYöneticisi Sınıfı İçin Örnek CRC Kart	98
Tablo VII.7 zlisteDenetleyicisi Sınıfı İçin Örnek CRC Kart	98
Tablo VII.8 Örnek Uygulama Akış Haritası.....	98
Tablo VII.9 Ant Build.xml Dosyası.....	107
Tablo VII.10 Ant Build.xml Dosyası.....	108
Tablo VII.11 Tomcat Uygulama Sunucusu Dizin Yapısı.....	113

BÖLÜM I

GİRİŞ ve AMAÇ

I.1. GİRİŞ ve AMAÇ

Yazılım geliştirmede hedef, sabit bütçe ve belirli bir zaman diliminde müşteri gereksinimlerini gerçekleştiren, piyasadaki rekabet etme yeteneğini kuvvetlendirecek bir yazılım sistemi geliştirmektir. Yazılım geliştirme süreçleri, hedefi gerçekleştirmek için kullanılan metotları içermektedir. Günümüzde yazılım endüstrisi, zamanında ve bütçe dahilinde çalışan yazılım teslim edilmesinde kötü bir sicile sahiptir. Birçok proje planlanan süre ve bütçenin dışına çıkıp iptal edilmekte veya bitirilen proje müşteri gereksinimlerini, piyasa koşullarını karşılamayacak durumda olmaktadır.

Geleneksel yazılım geliştirme süreçlerinde kullanılmakta olan doğrusal sistemler, müşteri istekleri, zaman, özkaynak, kalite ve vizyon gibi değişkenlerin ihtiyaçlarını çözmekte yetersiz kalmaktadır. Geliştirme sürecinde sonradan gelen değişiklikler karmaşıklığın artmasına sebep olmaktadır. Bundan dolayı tüm değişkenlerin hareketlerine uyumlu bir yazılım geliştirme sürecine ihtiyaç duyulmuştur. Bunun için 2001 yılında toplanan sektörün öncü yazılımcıları buldukları çözüme “çevik” adını vermişlerdir.

Çevik yazılım geliştirme, bir düşünme tarzı ve bir felsefedir. Çevik yazılım geliştirme, tekrarlanan yazılım geliştirme model alınarak geliştirilmiş, sık aralıklarla yazılım teslimatını ve değişikliği teşvik eden bir yazılım geliştirme metodolojisidir.

Tez çalışmamda, başarılı yazılım projelerinin üretilmesi için gerekli olan araştırma sonuçları yer almaktadır. Yazılım mühendisliğinden başlayarak geleneksel yazılım geliştirme süreçleri ve çevik yazılım ilkeleri incelenmiştir. Çevik yazılım süreçleri “çevrimiçi zaman çizelgesi sistemi” adlı proje ile uygulanarak anlatılmıştır. Çalışmamda proje başarısızlık nedenlerinin belirlenmesi ve yazılım geliştirme süreçlerinde çevik metotların etkilerinin tespit edilmesi amaçlanmıştır.

BÖLÜM II

YAZILIM MÜHENDİSLİĞİ

II.1. YAZILIM MÜHENDİSLİĞİ TANIMI

Modern yazılımın temelleri, Alan Turing tarafından 1936 yılında yazılan “Saptama Problemi Hakkında Bir Uygulamayla Birlikte Hesaplanabilir Sayılar” adlı makalesi ile atılmıştır. Yazılım terimini ise ilk olarak 1958 yılında ABD’li istatistikçi John W. Tukey kullanmıştır. [1,2]

Yazılım, değişik ve çeşitli görevler yapma amacıyla tasarlanmış elektronik araçların, birbirleriyle haberleşebilmesini ve uyumunu sağlayarak, görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan makina komutlarıdır. Bilgisayar bilimleri ve yazılım mühendisliği olarak, bilgisayar yazılımı tüm bilgisayar programlarıdır. [3]

Yazılım mühendisliği, yazılımın geliştirilmesi, uygulanması ve bakımında kullanılan sistemli, disiplinli, ölçülebilir bir uygulamadır. [4]

IEEE bilgisayar topluluğu, yazılım mühendisliğini, sistemli ve ölçülebilir yaklaşımların, yazılımın geliştirilmesine, işletilmesine ve bakımına, kısacası mühendislik tekniklerinin yazılıma uygulanması olarak tanımlamaktadır. [8]

Yazılım mühendisliğinin amaçlarından birisi, yazılım geliştirmedeki karmaşıklığı azaltmaktır. Bunun için önerilen yöntemlerden birisi soyutlama seviyesinin artırılmasıdır. Soyutlama seviyesinin artırılması, yazılım geliştirme sürecinde üretilen yapıların mümkün olduğu kadar problem alanına yakınlaştırılmasıyla mümkündür. Bunun için yazılım geliştirme süreçlerinde modellerin ana yapılar olarak kullanılması gelecek nesil yazılım geliştirme metodolojilerinin temelini oluşturmaktadır. [17]

II.2. YAZILIM MÜHENDİSLİĞİ TARİHÇESİ

Yazılım mühendisliği terimi ilk kez 1950’lerin sonunda 1960’ların başında görülmeye başlanmıştır. Programcılar, inşaat, elektrik ve bilgisayar mühendisliklerini biliyorken yazılım için mühendisliğin ne olduğunu tartışmaya başlamışlardır. NATO

Bilim Komitesi, 1968'deki ve 1969'daki yazılım mühendisliği üzerine bu alana destek olan iki konferansa (Garmisch, Almanya) destek olmuştur.

1960'lar, 70'ler ve 80'lerdeki yazılım geliştirmedeki sorunları tespit eden yazılım krizleri, yazılım mühendisliğini teşvik edici unsur olarak kendini göstermiştir. Yazılım krizlerinde, birçok yazılım tasarısı bütçeyi aşarak ve düşünülen zamanı geçmiştir. Bazı tasarılar fiziksel zarara neden olmuştur. Bir kısım tasarılar ise ölümlere bile neden olmuştur. Yazılım krizi, önceden verimlilik açısından tanımlanmış olsa da sonradan kaliteyi vurgulama ön plana geçmiştir. Bazıları ise yazılım krizi kavramını yeterli kalitede programcıya sahip olamama karşılığı olarak kullanmıştır. [4]

Yıllardır yazılım krizi, araştırmacılar ve yazılım araçları üreten firmalarla çözülmüştür. 1970'li yıllardan 1990'lara kadar her yeni çıkan yazılım teknolojisi ve pratiği, yazılım krizini çözmek için “Gümüş Kurşun” (Silver Bullet) olarak görülmekteydi. Araçlar, disiplin, biçimsel yöntemler, süreçler ve profesyonellik gümüş kurşun olarak gösterilmiştir.

Araçlar: Yapısal programlama, nesneye tabanlı programlama, CASE araçları, Java, dokümantasyon, standartlar ve UML.

Disiplin: Bazı uzmanlara göre programcının disiplinli olmaması sebebiyle yazılım krizleri çıkmaktaydı.

Biçimsel Yöntemler: Eğer yazılım geliştirmede biçimsel mühendislik yöntemleri uygulanırsa diğer mühendislik dalları gibi yazılım üretiminde bilinen bir endüstri haline geleceğine inanılıyordu.

Süreç: Birçok kişi tanımlanmış süreçler ve CMM gibi metodolojilerin kullanımını savunmuştur.

Profesyonellik: Yazılım çalışmalarında sorumluluk, görevini yerine getirme ve etik kurallar yol gösterici olmuştur.

1986 yılında Fred Brooks'un yayınladığı “Gümüş Kurşun Yoktur” makalesi ile hiçbir bireysel teknoloji ya da pratiğin 10 yıl içinde 10 kat verimliliği sağlayamayacağını savunmuştur. Bunu kabul etmeyenler ve itiraz edenler olmuştur. En sonunda hemen hemen herkes yazılım krizlerinde gümüş kurşun olmayacağını kabul etmiştir. Bilinen tüm teknolojiler ve uygulamalar verimlilik ve kalitede tek aşamalı yapılan iyileştirmelere sahip bulmaktaydı. “Gümüş Kurşun Yoktur” yazılım mühendisliğinde olgunlaşma kanıtı ve zor çalışmalarda projelerin başarı anahtarı gibi yorumlanmıştır.

1990 yılından sonra internetin yükselişi ile WWW üzerinden uluslararası bilgi görüntüleme ve e-posta sistemlerine olan talepte çok hızlı bir büyüme olmuştur. Bu nedenle programcılar az bilinen yöntemlerle görüntü ekranını optimize etmek, depolamak için daha önce hiç görülmemiş biçimde çizimler, haritalar, fotoğraflar ve diğer resimleri, basit animasyonları ele almaları gerekmiştir.

2000 yılı ve bugünlerde hafif metodolojiler kullanılmaktadır. Birçok küçük organizasyon, yazılım yükseltme talebi ve ucuz yazılım çözümleri gereksinimiyle daha basit, daha hızlı metodolojilerin büyümesine yol açmıştır. Tüm hafif metodolojilerde hızlı modellemenin kullanımı gelişmiştir. Ekstrem programlama buna örnek olarak verilebilir. Çok büyük yazılım sistemleri, çok büyük hacimli dokümanları düzenlemede hala yoğun olarak kullanılan doküman metodolojilerini kullanmaktadır. Ancak, daha küçük yazılım sistemleri daha basit ve hızlı gelişim yönetimi için alternatif bir yaklaşıma sahip olmuştur.

Yazılım mühendisliği, gelişmekte olan genç bir disiplindir. Yazılım mühendisliğinde güncel eğilimler şunlardır:

Bakış Açısı (Aspect): Tüm nesnelerin veya fonksiyonların belli koşullarda nasıl davranması gerektiğini tarif etmektedir. Oldukça yeni ve ilgi çekici bir yazılım mühendisliği dalı olan “aspect” tabanlı yazılım mühendisliğinin daha verimli ve etkili uygulamaya geçirilmesi için çalışmalara devam edilmektedir.

Çevik (Agile): Çevik yazılım geliştirme yöntemleri, yazılım geliştirme projelerinde değişen beklentiler ve rekabetçi piyasalar ile hızla gelişmektedir. İlgili kavramlardan bazıları şunlardır: Ekstrem Programlama, Scrum.

DeneySEL (Experimental): DeneySEL yazılım mühendisliği, yazılım denemeleri oluşturulması sırasında, deney verileri toplamak, bu verilerinden kurallar ve teori oluşturulması ile ilgilenen yazılım mühendisliğinin bir dalıdır.

Model Tabanlı (Model Driven): Model Tabanlı Tasarım birincil tasarım eserler olarak metin ve grafik modelleri geliştirmektedir. Yazılım alanında model tabanlı tasarımın kullanılması, tasarımın öncesinde ve tasarım anında tasarımcıya olumlu katkılar sağlayabilmektedir.

Yazılım Ürün Hatları (Software Product Lines): Yazılım Ürün Hatları, tamamen bireysel bir ürün yaratmak yerine yazılım sistemleri aileleri üretmek için sistematik bir yoldur. [4-7]

II.3. ALT DİSİPLİNLER

Yazılım Mühendisliği 10 ayrı alt disipline ayrılabilir. Bunlar: [8]

- Yazılım Gereksinimleri
- Yazılım Tasarımı
- Yazılım Geliştirme
- Yazılım Testi
- Yazılım Bakımı
- Yazılım Konfigürasyon Yönetimi
- Yazılım Mühendisliği Yönetimi
- Yazılım Geliştirme Süreci
- Yazılım Mühendisliği Araçları
- Yazılım Kalitesi

Yazılım mühendisliği ile ilgili diğer disiplinler ise Bilgisayar Bilimi, Proje Yönetimi ve Sistem Mühendisliği'dir.

Bilgisayar Bilimi (Computer Science): Yazılım mühendisliği birçok akademisyen tarafından bilgisayar biliminin alt alanı olarak kabul edilmektedir. Yazılım mühendisliğinin temellerinin birçoğu bilgisayar biliminden gelmektedir.

Proje Yönetimi (Project Management): Bir yazılım sisteminin üretilmesinde proje yönetimi alanında kullanılan birçok ilkenin aktarılması kabul edilmiştir.

Sistem Mühendisliği (Systems Engineering): Sistem mühendisleri, uzun yıllar büyük sistemlerinin karmaşıklığı ile ilgilenerek bilgilerini birçok yazılım mühendisliği problemlerine uygulamışlardır. [9]

II.3.1. Yazılım Gereksinimleri

Yazılım için gereksinimlerin ortaya çıkarılması, analiz edilmesi, tanımlanması ve doğrulanmasıdır. Yazılım gereksinimlerinde, ihtiyaçlar ve kısıtlamaların ifade edilmesiyle bir yazılım ürünüde bazı yaşanan problemlerin çözümüne katkı sağlanmaktadır. Gereksinim mühendisliği ise, gereksinimlerin sistematik olarak kullanımı alanında yaygın olarak kullanılan bir terimdir. [8]

Ortaya çıkarma: Yazılım belirleme süreci ve sistem gereksinimlerinde çeşitli kaynaklardan görüşmeler yoluyla gelen atölye, iş akışı, görev analizi, belge analizi ve diğer mekanizmalardır.

Analiz: Bilgi gereksinimlerinin sınıflandırılmasında çeşitli kategoriler halinde istenen niteliklerin şartları değerlendirilerek ortaya çıkan üst düzey gereksinimler ve benzeri durumlardan doğan farklı gereksinimlerdir.

Tanımlama: Bir sistemde gereksinimleri belgeleyen, paylaşılabılır, yapılandırılmış ve yönetilebilir bir formdur.

Doğrulama: Bir çalışmada ürün değerlendirme sürecinde müşteri gereksinimlerinin karşılanıp karşılanmadığının kontrolüdür. [19]

II.3.2. Yazılım Tasarımı

Tasarım, bir sistem ya da bileşenin, parça, arabirim ve mimari olarak tanımlanma süreci olup aynı zamanda sürecin sonucu olarak tanımlanabilmektedir. Bir süreç olarak görülen yazılım tasarımı, yazılım mühendisliği yaşam döngüsü yazılım gereksinimlerinde, yazılım iç yapısının tanımı temel olacak şekilde analiz edilmesidir.

Daha kısa bir tanımlama ile bir yazılım tasarımı yazılım mimarisi olarak açıklanabilir. Tasarım, yazılımın ayrıştırılması ve bileşenler halinde arayüzlerinin düzenlenmesidir. Bileşenlerin yapısını detaylı bir biçimde açıklamak gereklidir. Yazılım geliştirmede, yazılım tasarımı önemli bir rol oynamaktadır. Yazılım mühendisleri, çeşitli modelleri üretmek için bu tür yapıların bir çözüm planı ile uygulanmasını sağlamaktadır.

II.3.3. Yazılım Geliştirme

Yazılım geliştirme terimi kodlama, doğrulama, birim testi, entegrasyon testi ve hata ayıklama kombinasyonu aracılığıyla anlamlı bir yazılım için ayrıntılı çalışma oluşturmaktan gelmektedir. Geliştirme sürecinde, yazılım mühendisleri birim test ve entegrasyon çalışma testlerini gerçekleştirmektedir.

II.3.4. Yazılım Testi

Test, ürün kalitesini değerlendirmek, sorunların ve hataların belirlenerek iyileştirilmesi için gerçekleştirilen bir etkinliktir. Yazılım testi, bir programın davranışını dinamik doğrulama ve örneklerin sonlu bir küme içinde seçilmesidir. [8]

Yazılım testi, paydaşlarla ürün kalitesi hakkında bilgi sağlamak için yürütülen deneysel bir incelemedir. [20]

II.3.5. Yazılım Bakımı

Yazılım geliştirme çalışmalarında bir yazılım ürünü için müşteri gereksinimlerinin karşılanması teslimat ile sonuçlanmaktadır. Teslimat sonrası yazılım ürünü değiştirilebilmektedir. Bunun temel nedeni yeni kullanıcı ihtiyaçlarının ortaya çıkması ve ortamları değiştirmede kusurlar meydana gelmesidir. Yazılım yaşam döngüsünün bakım aşamasından sonra garanti süresi veya uygulama sonrası teslimat desteği başlamaktadır. Fakat bakım faaliyetleri çok daha erken ortaya çıkabilmektedir.

Yazılım bakımı, yazılım yaşam döngüsünün ayrılmaz parçasıdır. Yazılım bakımı, yazılım mühendisliğinde bir yazılım ürününün teslimatı sonrasında hataları düzeltmek, performansı arttırmak veya uyarlaması farklı olan bir yapıya göre değiştirilmektedir.

II.3.6. Yazılım Konfigürasyon Yönetimi

Bir sistemde, belli bir fonksiyonu gerçekleştirmek için düzenlenen bileşenler topluluğu veya fonksiyonlar kümesi olarak tanımlanabilir. Bir sistemin yapılandırılması, donanım, yazılım veya bunların kombinasyonunda teknik dokümanlarla ve süreç içinde elde edilen fonksiyonel ve/veya fiziksel özelliklerdir.

Konfigürasyon yönetimi, sistem yaşam döngüsü boyunca yapılandırma, bütünlüğünü korumak ve izlenebilirlik için zaman içinde farklı noktalarda bir sistemin yapılandırması ile tanımlanan disiplindir. [8]

Basit olarak konfigürasyon yönetimi tanımı, bir yazılım projesi gelişiminin nasıl kontrol edilmesidir. Bir şirketin, yazılımının niteliklerinin değişimini yönettiği, kontrol ettiği, araç ve teknikleri içeren bir yazılım mühendisliği disiplindir. [21]

Konfigürasyon yönetiminin, gelişim sürecinde ve sonrasında en kritik etkinliklerden biri olmasının nedenlerinden bazıları şunlardır.

- Yazılımın soyut olması ve iyi tanımlanmış bileşenlerin azlığı yazılımın anlaşılmasını zorlaştırmaktadır.
- Daha fazla arayüzün olması ve bunların etkileri takip etme zorluğu getirmektedir.
- Herhangi bir değişiklik, sistemin derinliklerine kadar yayılmaktadır.
- Herhangi bir değişiklik, tüm sistemin testinin yapılmasına neden olabilmektedir.
- Hataların oluşması ani olabilmektedir.

- Değişiklik için kodları eklemek basit ve hızlıdır. Fakat büyük riskler taşıyabilmektedir. [41]

II.3.7. Yazılım Mühendisliği Yönetimi

Yazılım mühendisliği yönetimi, disiplinli, sistematik ve niceliksel olan yazılımın geliştirme ve bakımında planlama, koordineli ölçme, izleme, kontrol ve raporlama yönetim faaliyetlerinin uygulaması olarak tanımlanabilir.

Yazılım mühendisliğinde yönetimi üç düzeyden meydana gelmektedir.

- Organizasyon ve altyapı yönetimi
- Proje yönetimi
- Ölçüm programı planlama ve kontrol

II.3.8. Yazılım Geliştirme Süreci

Yazılım mühendisliği süreci, sadece büyük organizasyonlar için değildir. Süreç ile ilgili etkinlikler, küçük organizasyonlarda takımlar ve bireyler tarafından başarıyla gerçekleştirilerek yapılabilmektedir.

Yazılım yaşam döngüsünde süreçleri yönetmenin amacı, bireysel proje veya organizasyon uygulama süreçlerinde yeni ve daha iyi süreçlerin uygulanmasını sağlamaktır.

II.3.9. Yazılım Mühendisliği Araçları

Yazılım geliştirme araçlarında bilgisayar tabanlı araçlar bulunmaktadır. Yazılım yaşam döngüsüne yardımcı olmaları için tasarlanmıştır. İyi tanımlanmış eylemlerle yazılım mühendisinin bilişsel yükü azaltılarak otomatikleştirilir. [8]

Yazılım destek araçları, yazılım programlarının bakımına, gelişimine yardım eden yazılım programlarıdır. Bu araçların uygunluğu, kalitesi, başarı ve başarısızlık arasındaki farkı belirleyebilmektedir. Destek araçları, tüm süreçte kullanılmaya ve ticari marketlerde daha fazla yer almaya başlamıştır. Bu nedenle araçlar belli bir yatırıma gereksinim duymakta, proje yöneticileri ve sistem mühendisleri için ilgilenilmesi gereken bir konu olmaktadır. [41]

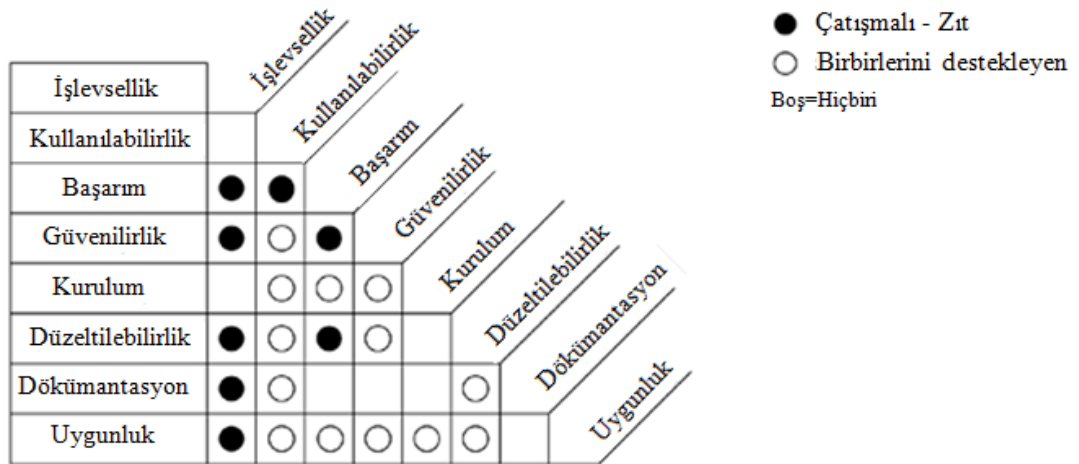
II.3.10. Yazılım Kalitesi

Yazılımda ürün kalitesi yaygın olarak kullanılan anlamıyla üründe hataların olmamasıdır. Temel anlamıyla gereksinimlere uygunluktur. Eğer yazılım, istediğiniz

işlevi sağlayan temel gereksinimlere sahip değilse yazılımda kalite sorunu bulunmaktadır. Bu tanım genellikle iki şekilde ifade edilmektedir. Kusur oranı ve güvenilirlik.

Müşteri memnuniyeti, genellikle memnuniyet anketlerinden gelen yüzde sonuçları ile ölçülmektedir. Genel müşteri memnuniyetinde, yazılım ürününün belirli özellikleri aynı zamanda doğru memnuniyeti ölçmede kullanılabilir. Örneğin, IBM sık sık müşterilerine gönderdiği yazılım ürünlerinde memnuniyeti belirlemeye yarayan CUPRIMDSO (capability[functionality]-işlevsellik, usability-kullanılabilirlik, performance-başarım, reliability-güvenilirlik, installability-kurulabilirlik, maintainability-düzeltilbilirlik, documentation/information-dokümantasyon/bilgi, service-servis, and overall-toplam) düzeylerini kullanmaktadır.

Genel müşteri memnuniyetinin artırılması için yazılım tasarımı ve planlaması aşamasında kalite öznitelikleri dikkate alınmalıdır. Bazı durumlarda kalite öznitelikleri birbirleriyle uyumlu olmayabilmektedir. Örneğin, yazılımın işlevsel karmaşıklığı yüksek, düzeltilbilirliğin zor olması gibi. Yazılım ve müşterilerin türüne bağlı olarak farklı ağırlık faktörleri, farklı kalite öznitelikleri için gereklidir. Büyük müşteriler için karmaşık ağlar ve gerçek zamanlı işleme, performans ve güvenilirlik önemli özellikler olabilmektedir. Diğer taraftan müşteriler için bağımsız sistemler ve basit operasyonlar ile birlikte, kullanım, kurulum kolaylığı ve dokümantasyon daha önemli olabilmektedir.



Şekil II.1 Yazılım Kalite Özniteliklerinde İlişkilerin Gösterilmesi – CUPRIMDA

Özellikle müşteri görüşleriyle, yazılım endüstrisi ile ilgili kalitenin tanımları (müşteri gereksinimlerine göre) güncellenmiştir. Gereksinimlerde oluşan hatalar büyük yazılım sorunlarına yol açabilmektedir. Tüm yazılım kusurlarının %15 veya

daha fazlası yazılım gereksinim hatalarından oluşmaktadır. Genel olarak yazılım kalite öznitelikleri Tablo II.1’de gösterilmektedir. [10,11]

Tablo II.1 Yazılım Kalite Öznitelikleri

Doğruluk	Yazılımın belirtilmiş ihtiyaçlarını karşılamasıdır.
Verimlilik	İşlevin gerçekleştirilmesi için kullanılması gereken bilgisayar kaynakları ve kod miktarıdır.
Genişleme	Yazılım yardımıyla işlevselliğin artırılmasıdır.
Güvenlik (Bütünlük)	Yetkisiz insanların ve istenmeyen yazılımların erişiminin ne derece engellenebildiğidir.
Birlikte çalışabilirlik	Bir yazılım sisteminin diğer sistemlerle bağlantı sağlama kolaylığıdır.
Bakım kolaylığı	Yazılımın kullanıcı gereksinimlerinde ortaya çıkan değişiklikleri karşılamadaki yeterliliğidir
Taşınabilirlik	Programın, farklı donanımlarda ve yazılım sistemi ortamlarında kullanılmasıdır.
Tekrar kullanılabilirlik	Yazılımın tamamının ya da bir bölümünün farklı bir uygulamada kullanılabilmesidir.
Güvenilirlik	Gerekli işlevi ne derecede hassaslıkla yerine getireceği beklentisidir.
Test kolaylığı	Yazılımın doğruluğunun sınanmasındaki doğruluktur.
Kullanılabilirlik	Programın öğrenilmesi, çalıştırılması, girdi hazırlama ve çıktı yorumlama işlemlerinin kolaylık derecesidir.

BÖLÜM III

YAZILIM GELİŞTİRME METODOLOJİLERİ

III.1. YAZILIM GELİŞTİRME

Yazılım geliştirme, yazılım ürünlerinde sonuca yönelik yapılan faaliyetlerin düzenlenmesidir. Yazılım geliştirme, yazılım ürünlerindeki araştırma, yeni geliştirme, değiştirme, yeniden mühendislik, bakım veya diğer etkinliklerdir. [15]

Yazılım geliştirme süreci, bir yazılım ürününün üretilebilmesi için gerekli olan birçok etkinlikler topluluğudur. İdeal bir süreç olmadığından her farklı organizasyon, yazılım geliştirme için farklı yaklaşımlar geliştirmiştir. Süreçler, organizasyondaki insanların yeteneklerine, geliştirilecek olan projenin özelliklerine göre geliştirilmektedir. Bu nedenle aynı şirkette farklı projeler için farklı süreçler olabilmektedir.

Tüm yazılım süreçlerinde ortak olan etkinlikler bulunmaktadır.

- a) Yazılım özelliklerinin belirlenmesi; yazılım ihtiyaç ve kısıtlarının saptanması,
- b) Yazılımın tasarımı ve gerçekleştirilmesi,
- c) Yazılımın geçerliliğinin sağlanması ve teslim edilmesi; müşteri ihtiyaçlarının karşılanıp karşılanmadığının test edilmesi ve teslimi,
- d) Yazılım bakımı; müşteri ihtiyaçlarının değişikliğine göre ya da oluşan herhangi bir yanlışlığa karşı yazılımın bakımı. [9]

III.2. YAZILIM GELİŞTİRME SÜREÇLERİ

Tüm yazılım geliştirme süreçlerinde ortak olan altı etkinlik sırasıyla aşağıda sıralanmıştır.

- | | |
|----------------------------|----------------|
| a) Gereksinim Mühendisliği | d) Entegrasyon |
| b) Tasarım | e) Teslimat |
| c) Programlama | f) Bakım |

a) Gereksinim Mühendisliği (*Requirements Engineering*)

Bu süreçte, geliştiriciler ve müşteriler yeni yazılım ürünü için fikir alışverişi yaparlar. Geliştiriciler, müşterinin gerçek ihtiyaçlarını belirlemek amacıyla çeşitli teknikler kullanırlar. Gereksinimleri geliştirmenin bir diğer yöntemi ise prototip oluşturma ve bu prototipi müşteriye sunarak daha önceden alınan gereksinimlerin eksiklerini tamamlamak ve geliştirmektir.

Gereksinim mühendisliği, müşterinin bir yazılımdan beklentilerini, işlem yürütme ve geliştirme esnasındaki kısıtları da dikkate alarak ortaya koyma ve gereksinimleri belgeleme işlemlerini sistematik olarak yapabilme disiplini.

b) Tasarım (*Design*)

Bu işlem sırasında, geliştiriciler belge üzerinde tespit edilen gereksinimlerin belirlenmesi ve çeşitli tanımlamaların kabul edilmesi ile yazılım üretimine başlamaktadır. Genellikle yazılım tasarımı, çeşitli aşamalar içinde ilerledikçe daha ayrıntılı olmaktadır. Geliştiriciler, yazılım karmaşıklığının yönetilmesi ve tasarıma mümkün olduğunca yoğunlaşabilmek için detaylarla ilgili kararların ertelenmesini sağlamaktadır. Tasarım tamamlandığında, tasarım özellikleri belgeye kaydedilmektedir.

c) Programlama (*Programming*)

Bu aşamada, takıma ait programcılar, yazılımın gerçek kodunu yazmaya başlarlar. Programlama sürecinde karmaşıklığı ele almak için, yazılım bölümlere ayrılarak modül olarak incelenmektedir. Takımlar kendi modüllerinden sorumlu olmaktadır. Uygun dokümantasyon kodlarını tanımlama ve kodların doğruluğunu test etmede takım üyeleri sorumlu olmaktadır.

d) Entegrasyon (*Integration*)

Bu işlem sırasında, yazılım ürününe ait modüller tek tek birleştirilerek istenen yazılım oluşturulmaktadır. Modüller ayrı ayrı geliştirilmiş olup, test entegrasyon süreci için çok önemlidir. İyi bir tasarımda bile, modüller arasında uyumsuzluklar çıkabilmektedir. Entegrasyonun tamamlanması için bu sorunların tespit edilip düzeltilmesi gerekmektedir.

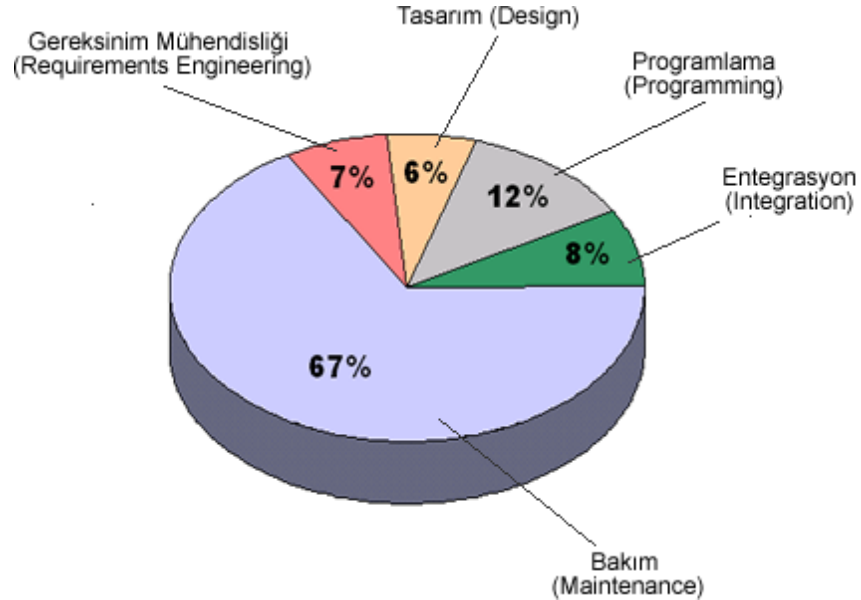
e) Teslimat (*Delivery*)

Bu işlem sırasında, geliştiriciler müşterilerine tamamlanmış yazılım sunabilmektedir. Genellikle müşteriler, yazılımın kabul kuralları testine uygun olup olmadığını veya belgedeki gereksinimleri karşılama durumunu kontrol etmektedir. Bir kez kabul edilen yazılım, müşteri tarafından kurularak kullanılabilir.

f) Bakım (*Maintenance*)

Yeni işlevler eklemek, yeni platformlara uygunluk, yeni teknolojilere uyarlama ve hataları düzeltme sırasında, yazılım ilk aşamaya göre çeşitli değişikliklere uğramaktadır. Görünürde yazılımın bitmesi teslim edilebilir durumda olması anlamına gelmez. Tüm başarılı yazılım ürünleri, müşterilerin zamanla değişen ihtiyaçlarını karşılamalıdır.

Yazılım geliştirme süreç döngüsünde bakım maliyetinin durumu öğrenilebilir. Şekil III.1’de süreçlerdeki göreceli maliyet oranı gösterilmektedir. Birçok geliştiricinin, tasarım yaklaşımları kullanırken yazılımı koruması açısından bakım maliyetleri çok önemlidir.



Şekil III.1 Yazılım Geliştirme Süreçlerine Göre Yaklaşık Maliyet Oranı

Yazılım geliřtirmede bu temel altı srece sahip olan, ok farklı modeller yaratılmıřtır. Bu modellerde kullanılan sreler, farklı isimlerde kullanılarak ortak olan altı srecin tarifi model iinde yer almaktadır. Ayrıca bu modeller, yazılım geliřtirmede belli bir teknięi vurgulayabilmektedir. rneęin tasarım teknięi olarak (hızlı prototipleme-rapid prototyping), ynetim teknięi olarak (risk ynetim-risk management) ya da yazılım geliřtirmede sınırlı bir etkiye sahip olan model (gerek zamanlı yazılım-real-time software) olabilmektedir. [22,23]

III.3. YAZILIM GELİřTİRME METODOLOJİLERİ

Metodoloji, yazılım geliřtirme srecinin ařamalarını eřitli basamaklara blmektedir. Her adımda ne yapılması, hangi sırada yapılması gerektięini, hangi dokmanların oluřturulması gerektięini tarif etmektedir. Sre ařamalarında detaylı bir plan sunmaktadır. Kısaca, sistem geliřtiricilerinin yazılım ve bilgi sistemi oluřturabilmesi iin gerekli olan etkinlikleri, modelleri, rotaları, sre rehberlerini ve otomatikleřtirilmiř araları iermektedir. Yazılım geliřtirme metodolojisi, yazılım mhendislięinde bir bilgi geliřtirme srecinin kontrol, planlaması ve yapısında kullanılan bir erevedir. [9,24,26]

Birok sayıda olan bu ereveler, kendine has gl ve zayıf ynleri ile yıllar iinde geliřmiřlerdir. Bir yazılım geliřtirme metodolojisi, tm projelerde kullanım iin uygun deęildir. Mevcut olan tm metodolojiler, belirli tr projelerde eřitli teknik tabanlı, rgtsel, proje ve takım deęerlendirmelerine dayalı olarak kullanılabilmektedir.

Yazılım geliřtirme aralarının en eskisi, kkleri 1920’li yıllara uzanan akıř řemasıdır. 1960’lı yıllara kadar yazılım geliřtirme metodolojisi meydana gelmemiřtir. Sistem geliřtirme yařam dngs (SGYD), bilgi oluřturmak iin bilinen en eski metodoloji olarak dřnlebilir. SGYD’nin temel amacı, disiplinli ve sıralı olarak fikrin bařlangıcından son sistemin teslimine kadar, yařam dngsnn her ařamasında gerekli yapısal, planlı, metodik yollarla bilgi sistemlerinde olan geliřmeleri takip edebilmektedir. [25]

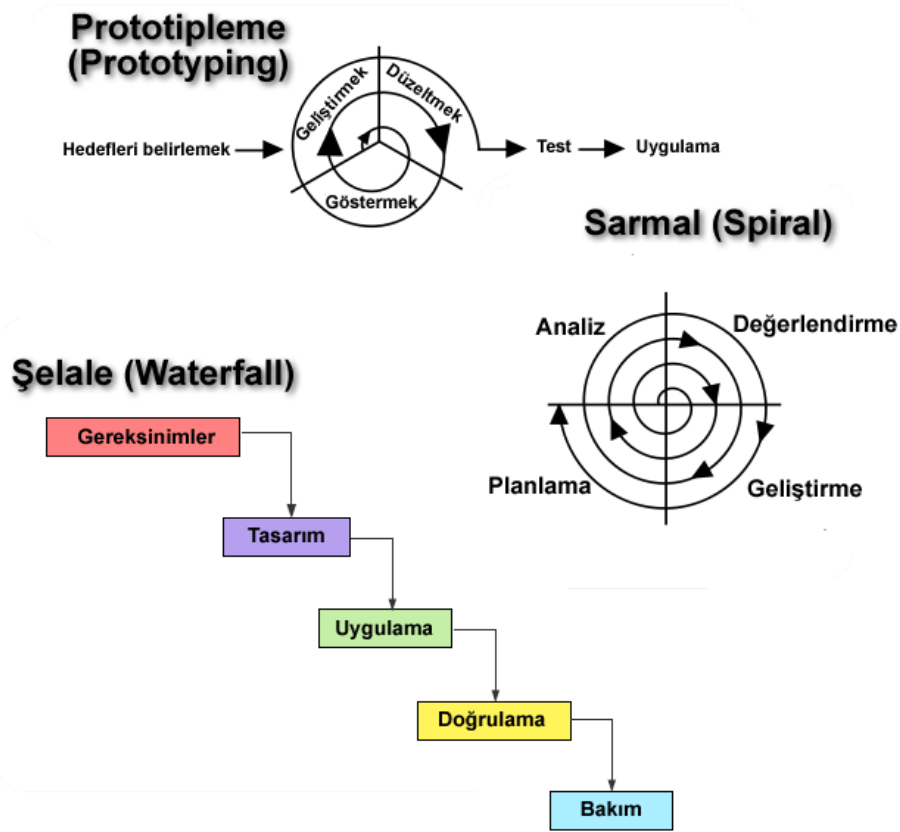
Tablo III.1 Belirli Yazılım Geliştirme Metodolojileri

YILLAR	YAZILIM GELİŞTİRME METODOLOJİLERİ
1970'ler	1969'dan günümüze yapısal programlama
1980'ler	1980'den itibaren yapısal sistem analizi ve tasarımı metodolojisi.
1990'lar	1960'lardan günümüze nesneye dayalı programlama
	1990'ların sonlarından itibaren Scrum (Çarpışma).
	1991 yılından günümüze hızlı uygulama geliştirme.
	Takım yazılım süreçleri Watts Humphrey tarafından geliştirildi.
2000'ler	1998' den itibaren rasyonel birleşik süreç.
	1999'dan itibaren Ekstrem programlama. (Extreme Programming)
	2005 yılından bu yana Scott Ambler tarafından birleşik çevik süreçler.

Her yazılım geliştirme metodolojisi, kendine özel yazılım geliştirme yaklaşımına sahiptir. Bazı geliştirilen yazılım metodolojileri ise daha genel yaklaşımların düzenlenmesidir. [26]

En çok kullanılan yazılım geliştirme yaklaşımları aşağıda sıralanmıştır.[28]

- a) Şelale Süreci (Waterfall Process)
- b) Artımlı Geliştirme (Incremental Development)
- c) Evrimsel Geliştirme (Evolutionary Development)
- d) İteratif Geliştirme (Iterative Development)
- e) Sarmal Model (Spiral Model)
- f) Prototipleme (Prototyping)
- g) Çevik Metotlar



Şekil III.2 Yazılım Geliştirme Metodolojilerinde Kullanılan Üç Temel Süreç [27]

III.3.1. Şelale Süreci (Waterfall Process)

Geleneksel şelale süreci, analiz, tasarım, kodlama, test gereksinimleri ile yazılım sürecinde ilerlemede tek geçişli, ardışık bir yaklaşımdır. Teorik olarak her bir aşama, sonraki aşamaya geçilmeden önce tamamlanmalıdır. Şelale modelinin varyasyonları, normalde tanımlama, tasarım, kodlama, test ve sık sık operasyon/bakım gibi temel faaliyetleri içermektedir. Şekil III.3' de yaygın olarak kabul edilen şelale süreci modeli gösterilmektedir.[28]



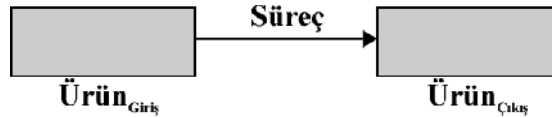
Şekil III.3 Geleneksel Şelale Süreci [28]

Şelale modeli, klasik yazılım yaşam döngüsü modelidir. Bu model 1980'lerin başına kadar yaygın olarak kabul edilen yazılım geliştirme süreciydi. Her sürecin ürün dönüşümünde çıkış olarak yeni bir ürün üretilmektedir. Üretilen yeni bir ürün bir sonraki sürecin girdisi olmaktadır. Şelale modelinin ürünleri ve süreçler Tablo III.2'de listelenmiştir. [29]

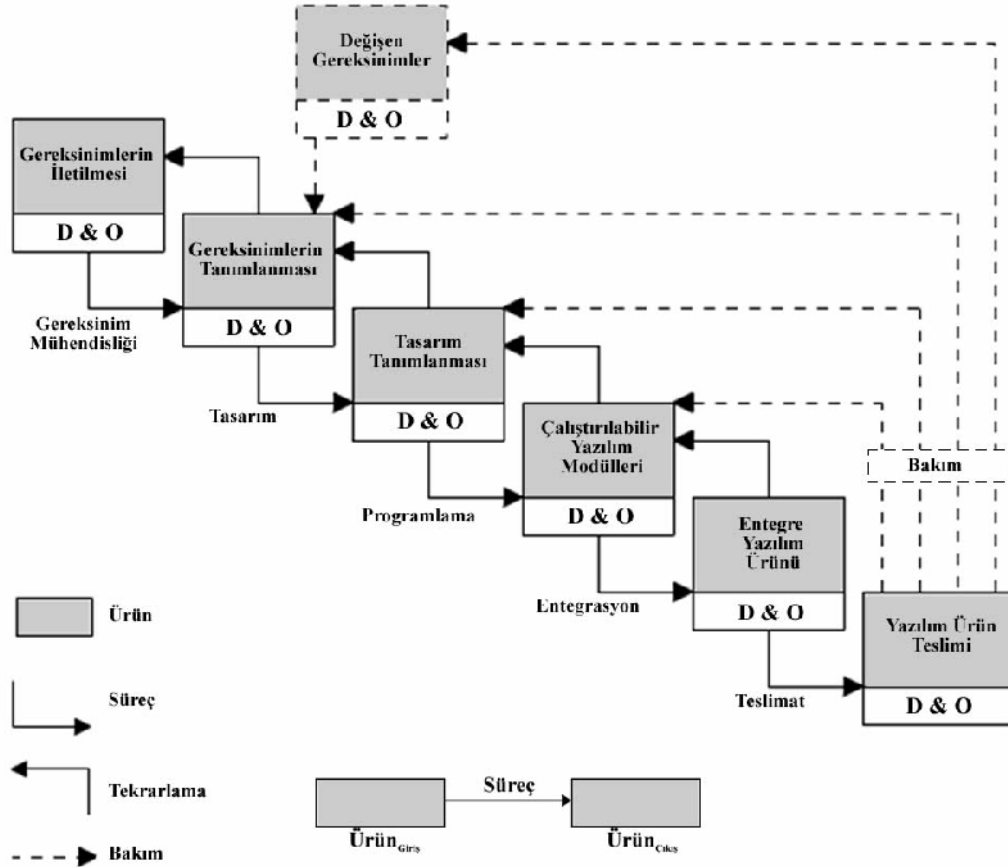
Tablo III.2 Şelale Modeli Ürünler ve Süreçler

Ürün Girdisi	Süreç	Ürün Çıktısı
Gereksinimlerin İletilmesi	Gereksinim Mühendisliği	Gereksinimlerin Dokümanda Tanımlanması
Gereksinimlerin Dokümanda Tanımlanması	Tasarım	Tasarımın Dokümanda Tanımlanması
Tasarımın Dokümanda Tanımlanması	Programlama	Çalıştırılabilir Yazılım Modülleri
Çalıştırılabilir Yazılım Modülleri	Entegrasyon	Entegre Yazılım Ürünü
Entegre Yazılım Ürünü	Teslimat	Yazılım Ürününün Teslimi
Yazılım Ürününün Teslimi	Bakım	Değişen Gereksinimler

Tablo III.2’de en sağda yer alan sütun ürün çıktısı, bir sonraki satırda alt düzeyde sürecin solundaki giriş ürünü olmaktadır. Süreçlerde ürünlerin dönüşümünün tanımlanması için genel formül Şekil III.4’de gösterilmektedir.

**Şekil III.4** Süreçlerde Ürünlerin Dönüşümü

Her ürün gri kutu ile temsil edilmektedir. Süreçler ise kutulara bağlantı yapılan ok ile temsil edilmektedir. Şekil III.5’de şelale modelinde yer alan süreçler ve ürünler daha iyi incelenmektedir.



Şekil III.5 Şelale Modeli Süreçleri ve Ürünler

Şelale modelinde, süreç ve ürünlerin yanı sıra diğer birkaç önemli geliştirme faaliyeti ve özellikler temsil edilmektedir. Bunlardan biride etkinlikler testidir. Schach’a göre, “Şelale modelinin her aşamasının doğası testtir. Test, yapılması gereken ayrı bir evre değildir. Sadece ürün inşa edilmesinden sonra her aşamasının sonunda yapılmasına gerek yoktur. Bunun dışında test, yazılım süreci boyunca devam eder.” [23]

Etkinlik testi iki adımdan oluşmaktadır. Adımlar biri doğrulama, diğeri de onaylamadır. Doğrulamada, yazılımın gerçekleştirilebilir olduğu ve yeterli uygunluk ile amaçlandığı diğer forma dönüşüm olmaktadır. Örneğin, tasarım süreci dönüşümünde, tasarım tanımlamaları dokümanı, çalıştırılabilir yazılım modülüne dönüşmektedir. Doğrulamayı bu durumda belirleyen, aslında yazılım modüllerinde tasarım belgesi ile açıklanan değerlerdir. Doğrulamayı şu soru cevaplar “Doğru ürünü üretiyor muyuz?”

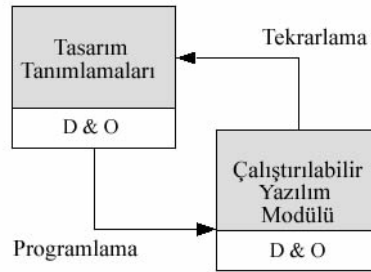
Onaylama ise yeterli doğrulukta kendi gereksinimlerinin belirlenerek yazılım fonksiyonlarının gerçekleştirilmesidir. Aynı örnek ile devam edilirse, çalıştırılabilir

yazılım modüllerini onaylayan, gereksinimlerde tanımlanan davranışlardır. Onaylamayı şu soru cevaplar: “Ürünü doğru üretiyor muyuz?”

Doğrulama testleri ve onaylama, yazılım ürün geliştirmeye paralel olarak yapılmalıdır. D & O kutuları şelale modelinde her ürün altında görünmektedir.

- Doğrulama, ürünün gerçekleştirilmesi ile ilgilenmektedir. (Yazılımcı)
- Onaylama, doğru ürünün yapılması ile ilgilenmektedir. (Müşteri)

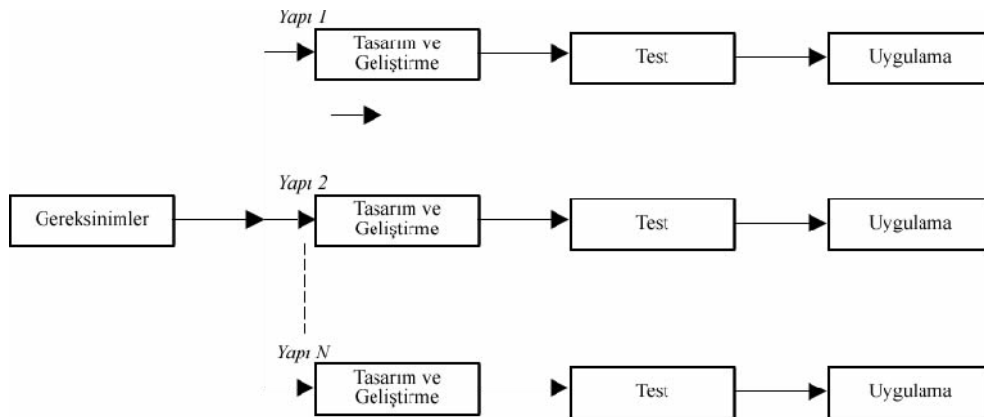
Şelale modelinin bu önemli özelliği, tekrarlama okları ile süreçlere birlikte katılmaktadır. Bu oklar bir ürüne önceki ürünlerdeki gelişme etkisinin nasıl etkilendiğini göstermektedir. Bir programlama süreci için iyi bir örnektir. Tasarım tanımlamaları dokümanı ve çalıştırılabilir yazılım modülleri döngüsünde, programlama süreci ve tekrarlama oklarının nasıl etkilendiği Şekil III.6’da gösterilmektedir.



Şekil III.6 Programlama ve Tekrarlama Süreçleri

III.3.2. Artımlı Geliştirme (Incremental Development)

Artımlı geliştirme, şelale modelinin devamı niteliğindedir. Bu model, tek döngü yaklaşımına karşı ayrı artışlarla farklı bir sistem olarak geliştirilmiştir. Kullanıcı gereksinimleri önceliklerine göre sınıflandırılarak yüksek öncelikli gereksinimler erken kesimlere dahil edilmektedir. Artımlar, sıralı döngü içinde geliştirilmektedir. Her artımla, bir sonraki döngüye yeni işlevler eklenmektedir. [28]



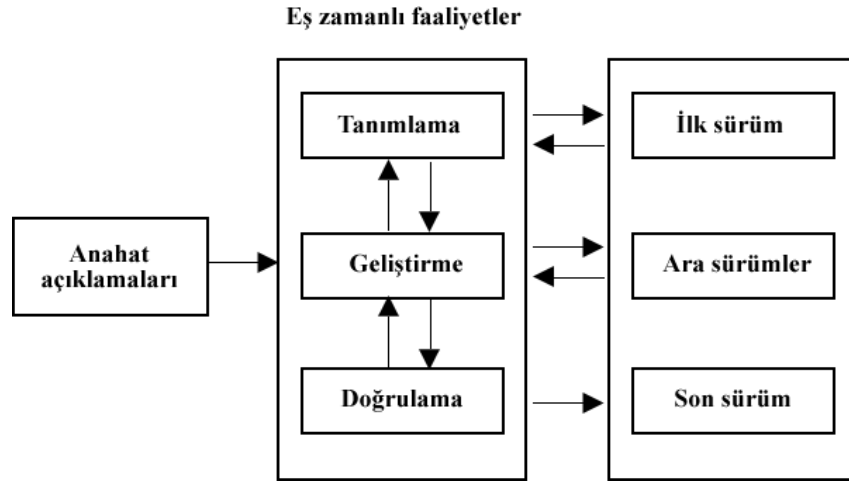
Şekil III.7 Artımlı Geliştirme Modeli [30]

III.3.3. Evrimsel Geliştirme (Evolutionary Development)

30 yıl önce ortaya çıkan şelale modelinden itibaren birçok model ileri sürülerek gösterilmiştir. Birkaç çalışma olmuş ve yeni modellerden evrimsel geliştirme gibi modeller deneysel tecrübeleri doğrulamıştır. [33]

Evrimsel geliştirme, şelale modelindeki eksiklikler düzeltilerek geliştirilmiştir. Evrimsel modelin aşamaları, işletme deneyimi tarafından belirlenen gelişime göre ürünün artımlı genişlemesidir. [28]

İlk geliştirmede, sistemin temel modülleri geliştirilmektedir. İlk ürün iskelet olarak düzenlenmeli ve kapasitesi artarak izleyen sürümlerde yeni işlevler eklenerek geliştirilmelidir. Amaç, müşteriyle çalışarak ana hatları belli ilk tanımlamalardan son sistemi geliştirmektir. [32]

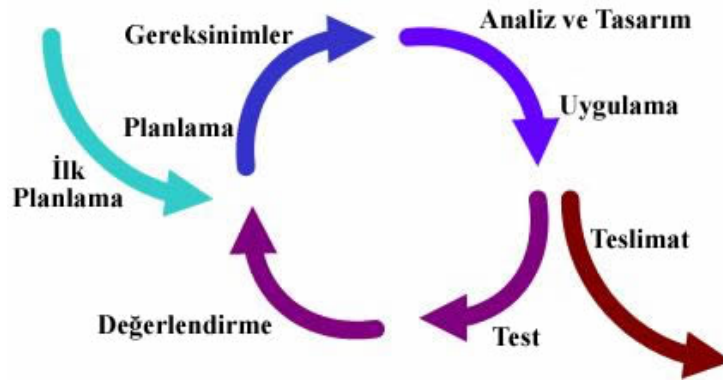


Şekil III.8 Evrimsel Geliştirme Modeli [37]

Bu modelin amacı, ilk ana hat tanımlarından son sistem durumuna kadar müşteri ile çalışmaktır. İyi anlaşılmış gereksinimler ile çalışmaya başlamak gerekmektedir. Sistem geliştirilmesi, müşterinin önerdiği yeni özelliklerin eklenmesi ile gerçekleşmektedir. [33]

III.3.4. İteratif Geliştirme (Iterative Development)

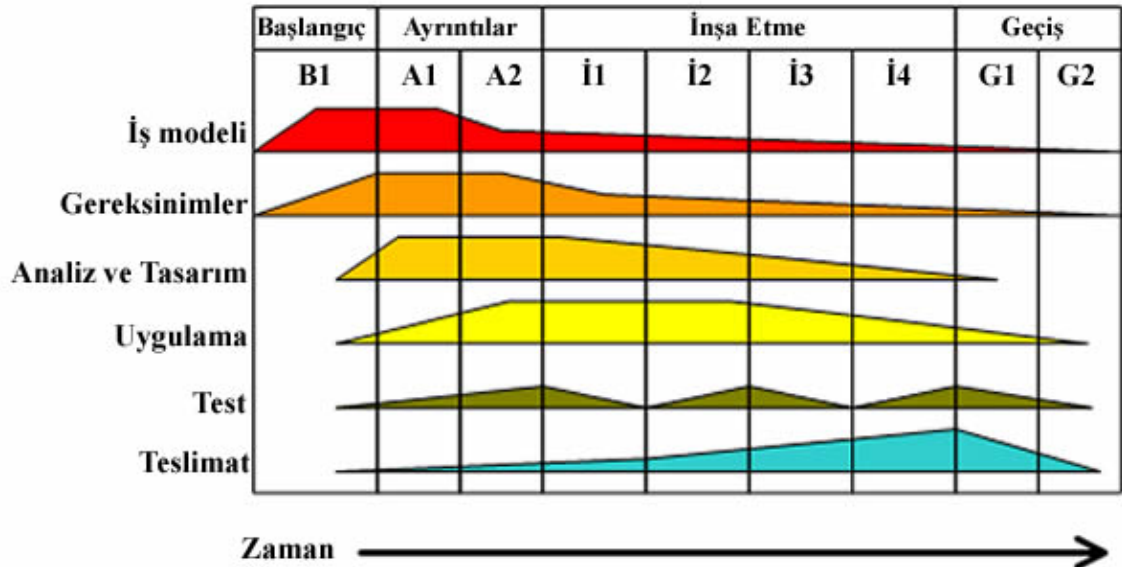
İteratif geliştirme, birbirini takip eden küçük, kullanışlı modüller halinde birçok geliştirme döngüleri kullanmaktadır. Kısa evrimsel tekrarlamalar yerine gereksinimlerin sağlandığı daha uzun tek geçişli, sıralı şelale süreci kullanılmaktadır. Artımlı geliştirmeden farkı, tekrarlayıcı yaklaşımlarda her döngü için önceden tüm gereksinimleri hazırlamak gerekmemektedir. Her iterasyon, çok kısa bir şelaleye benzemektedir. İlk iterasyon, genellikle gereksinimlerin düzenlenmesi ve tasarımı son şekline getirme gibi yüksek riskli alanlar ile temel bir özelliği uygulamaktadır.



Şekil III.9 İteratif Geliştirme Modeli [38]

İteratif geliştirme, kullanıcı ihtiyaçlarını karşılayacak uygulamaları temin etmeye yardımcı olmaktadır. Programa erken başlanarak, gerçek son kullanıcıların geri bildirimleri başarılı bir uygulama ile evrimsel gelişmeye neden olmaktadır. Örneğin, yazılım mimarisinde modüller halinde çalışma ile yüksek risk alanları daha erken tespit edilebilmektedir. Örneğin güvenlik, çıktı, yanıt süresi gibi. [28]

İteratif geliştirmede, konunun uzmanlarından sürekli geri bildirimin gelmesi, minimal yapı ve yüksek konsantrasyonlu takımla ortak bir vurgu paylaşılmaktadır. Bu süreçte ekibin tasarım ve test eksikliklerinde, bu tür metodolojilerin büyük bir yer tuttuğu görülmektedir. Tipik olarak bir proje, entegre proje ekibi ile projenin amaçlarıyla başlamalıdır.



Şekil III.10 İteratif Geliştirme [39]

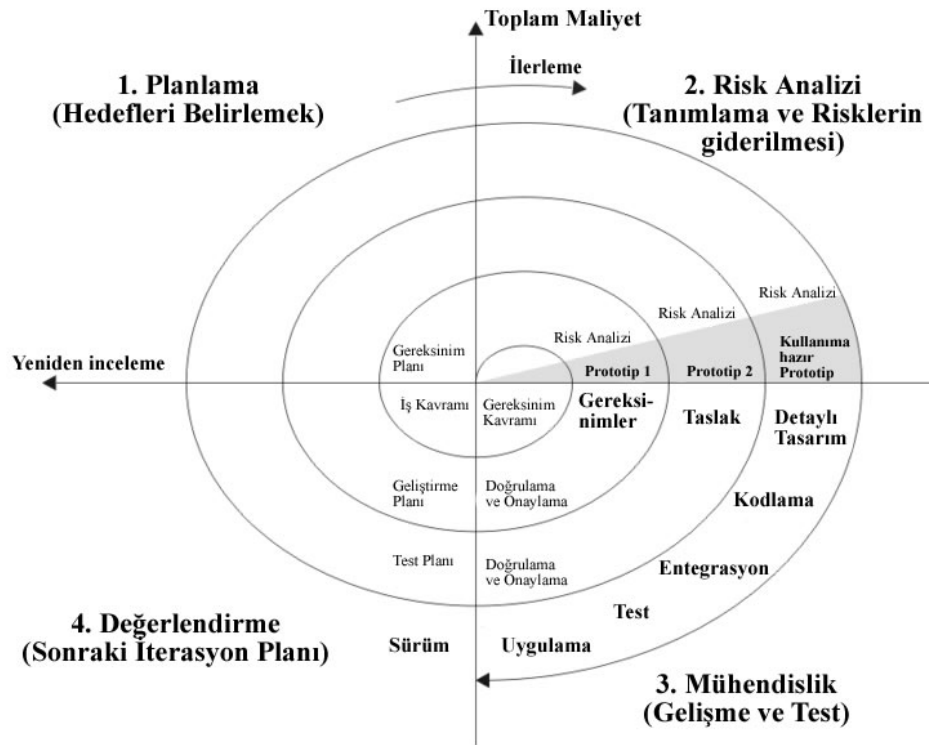
III.3.5. Sarmal Model (Spiral Model)

Sarmal model, risk analizine daha fazla ağırlık vermesiyle artımlı modele benzemektedir. Sarmal modelin dört aşaması vardır: planlama, risk analizi, mühendislik ve değerlendirme. Bir yazılım projesi, tekrarlamalar aracılığıyla bir aşamadan diğer aşamaya geçmektedir. Temel sarmal, planlama aşamasının başlangıcında gereksinimlerinin toplanması ve risk değerlendirmesi yapılmasını sağlamaktadır. Sonraki her sarmal, temel sarmala göre inşa edilmektedir. Gereksinimler, planlama aşamasında toplanmıştır. Risk analizi aşamasında bir süreç, risk ve alternatif çözümleri belirlemeyi üstlenmiştir. Bir prototip, risk analizi aşaması sonunda üretilmektedir. Yazılım, mühendislik aşamasında en son seviyede test yapılarak üretilmektedir. [34]

Modeli oluşturan her döngü,

- Amaçların, sınırların tanımlanması,
- Risklerin belirlenmesi,
- Gelişim ve geçerliliğin sağlanması,
- Bir sonraki seviyenin planlanması, aşamalarına dahil olmaktadır.

Odaklandığı nokta, bir sonraki riskleri azaltmaktır. Risk yönetimi kullanılması, diğer süreçlerden farkını oluşturmaktadır. [9]

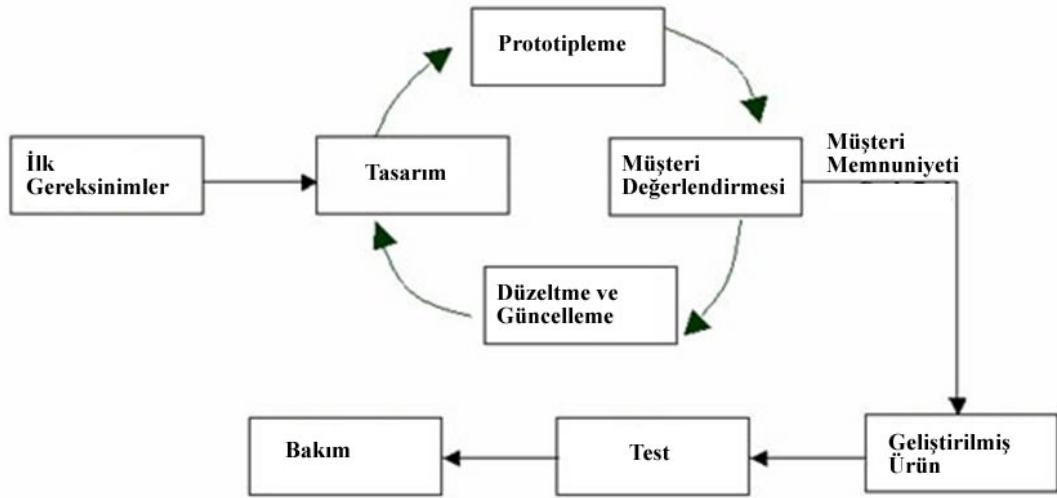


Şekil III.11 Sarmal Model [40]

III.3.6. Prototipleme (Prototyping)

Gereksinimlerin tanımlanmasında detaylı bir eksiklik olduğunda, sisteme girişte, işleme ihtiyaçları ve çıkış şartları için prototip model kullanılabilir. Bu model, ürünün çalışan bir örneği ile müşteriye etkileşim ve deneme sağlamaktadır. Gelişim süreci, müşterinin memnun olması ile en az bir kere prototip işleyerek sona ermektedir. Bu aşamada geliştirici, müşterinin gerçek ihtiyaçlarını belirlemektedir. [35]

Prototipleme ile geliştirici, yazılımda gereksinimlerin tanımlanmasını sağlayacak hızlı bir başlangıç yaparak çalışma modeli oluşturmaktadır. Amacı, temel özellikleri açıklamak ve sistemin çalışmasında görsel temsiller ile örnek çalışma oturumları kullanmaktır. Kısmi gereksinimlerin toplanması ilk önce paydaşlar ile yapılmaktadır. Prototipleme, tam ölçekli bir modeli ve sadece ilgili bölümde yeni sistemin fonksiyonel formunu üretmektedir. Hızlı bir tasarım, kullanıcıya odaklanılan yazılım yönlerinde görünmektedir. Üretilen prototip, gereksinimlerin düzeltilmesi için değerlendirilmektedir. [28]



Şekil III.12 Prototipleme Modeli [41]

III.3.7. Çevik Metotlar

Çevik terimi, bir felsefe ve yazılım mühendisliği projelerinin sürdürülmesi için kavramsal bir yapıdır. Çevik yöntemler, yazılım geliştirmenin kısa sürelerde iterasyonlar yapılarak riskin en aza indirilmesidir. Her iterasyon, kendi içinde minyatür bir yazılım projesi içermektedir. Gerekli tüm görevler ve yeni işlevler küçük artımlarla sürdürülmektedir. İterasyon sırasında, ürün sürümü yayımlamak

için yeterli işlevsellik olmayabilir. Çevik, bir yazılım projesinde her iterasyon sonucunda yeni bir yazılım sürümünün yeterli olmasını tasarlamaktadır. [36]

Çevik metotlar, son yıllarda çok desteklenen iteratif süreçler kümesidir. Bu metotlar yüksek hızlı ve geçici yazılım geliştirme için uygun olan bir "hafif" programlama değerleri dizisi olarak savunulmuştur. Çevik metotlara ait bazı örnekler aşağıdaki sıralanmıştır. [28]

- Ekstrem Programlama (EP)
- Scrum
- Dinamik Sistem Geliştirme Metodu (DSGM)
- Uyarlanabilir Yazılım Geliştirme (UYG)
- Kristal (Crystal)
- Özellik Gülümlü Geliştirme (ÖGG)
- Çevik Modelleme

BÖLÜM IV

ÇEVİK YAZILIM GELİŞTİRME

IV.1 ÇEVİK YAZILIM GELİŞTİRME TANIMI

Çevik kelimesi çeşitli kaynaklarda şu şekillerde tanımlanmaktadır.

Hız, hafiflik ile ayırt edilen, kolay taşınabilen.[43]

Atik, kıvrak, becerikli. [44]

Etkili ve manevra kabiliyetine sahip olma. [45]

Çevik yazılım geliştirme bir felsefedir. Bir yazılım geliştirme hakkında düşünme tarzıdır. Çevik bildiri bu düşünme tarzını açıklamaktadır. Çevik olunması için uygulamaya çevik değerler ve ilkeler yerleştirmek gerekmektedir. Süreç, bir çalışma biçimidir. Ne zaman bir şey yapılmak istendiğinde süreç takip edilmelidir. Çevik süreçler, çevik felsefesini destekleyen yöntemlerdir. Çevik süreçler, kendine özel olan uygulamalardan oluşmaktadır. Uygulamalar, sürüm kontrolü kullanımı, kodlama standartları, paydaşlara haftalık demolar vermeyi içermektedir. Bu uygulamaların çoğu yıllar içinde oluşmuştur. [52]

Çevik yazılım geliştirme, çeşitli iteratif ve artımlı yazılım geliştirme süreçleri için kullanılan bir ifadedir. En popüler çevik süreçler, ekstrem programlama, scrum, dinamik sistem geliştirme metodu (DSGM), özellik güdümlü geliştirmeden (ÖGG) oluşmaktadır. Çevik süreçlerin her birinde kendine özel benzersiz yaklaşımlar bulunmaktadır. Tümü ortak bir vizyon ve temel değerleri paylaşmaktadır. Süreçler, temel olarak iterasyon ve sürekli geri bildirim içermektedir. Süreç sonunda gereksinimlere en uygun yazılım teslim edilmektedir. Tüm çevik süreçler, hafif (özellikle geleneksel şelale süreçlerine göre) ve doğal olarak uyarlanabilirler. İşbirliğini güçlendirerek takım olarak hızlı ve etkin kararların alınmasına odaklanma süreçlerde görülen önemli bir özelliktir. [46]

Çevik düşünce, herhangi bir süreç ve uygulamalar kümesi kullanılarak gerçekleştirilebilir. Çevik düşüncesini anlatmanın en iyi yolu göstermektir. Bugün yazılım sektörü hızla ilerlemektedir. Sonuç olarak rakiplerden önce proje teslimi için yazılım geliştirme yaşam döngüsü ihtiyaçlarının yeterince esnek olmasını sağlamak,

organizasyonlara yetki vermek, yeni ve gelişmekte olan piyasa imkanlarını elde etmek gerekmektedir. İstenilen yeteneğe ulaşmak için cevap verme, sürekli değişim, yazılım süreci ihtiyaçlarını karşılamaya odaklanılması çok önemlidir. Çevik değerlerle, yazılım geliştirme sürecine hangi değerlerin katıldığı vurgulanmaktadır. [49]



Şekil IV.1 Çevik Değerler, İlkeler ve Uygulamalar Arasındaki İlişki [49]

IV.2. ÇEVİK SÜREÇ

IV.2.1. Çevik Sürecin Geçmişi

Çevik bildiri, çevik yazılım geliştirmeyi destekleyen ilkelerle ilgili bir açıklamadır. Bu bildiri, 11 ile 13 Şubat 2001 tarihlerinde toplanan bir grup yazılım geliştirici tarafından taslak olarak hazırlanmıştır. Burada yeni ve çeşitli metodolojilerin temsilcileri geleneksel ağır metodolojilere göre alternatif hafif metodolojileri görüşmüştür. Bu grupta yer alan onyediyi yazılım geliştirici tarafından yazılımcıların sorunlarına çözüm getirmek için Çevik birliği kurulmuştur. Bu birlik, (www.agilealliance.org) çevik yazılım üzerinde bilgi, sertifikalandırma ve diğer hizmetleri vermektedir. [50,51]

IV.2.2. Çevik Bildiri

2001 senesinde Kent Beck ve onaltı arkadaşı tarafından bir araya gelen grup, yazılım geliştirme sürecine verim sağlayacak değerleri belirlemek için yazılı bir belge hazırlamışlardır. Bu bildiri dört basit değer ile tanımlanmıştır. Bu bildiri hakkında düşünmek, yazılım projelerinde önceliklerin tanımlanmasını ve bazı

alanlarda odaklanarak diğer alanların ortadan kaldırılmasını sağlamıştır. Bu bildiri, yazılım kuruluşlarının büyük bir bölümünde olan sağdaki öğeler (süreçler ve araçlar gibi) için soldaki öğelerin (bireyler arasındaki etkileşim gibi) ihmal edildiğini işaret etmektedir. [49,50]

Tablo IV.1 Çevik Değerler [52]

Manifesto for Agile Software Development		
1. <u>Individuals and interactions</u> over process and tools		
2. <u>Working software</u> over comprehensive documentation		
3. <u>Customer collaboration</u> over contract negotiation		
4. <u>Responding to change</u> over following a plan		
That is, while there is value in the items on the right, we value the items on the left more.		
Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Kent Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brain Marick	

Türkçesi ve açıklaması:

1. Bireyler ve etkileşimler, süreç ve araçların üstündedir.
2. Çalışan program, kapsamlı dokümantasyonun üstündedir.
3. Müşteri ile işbirliği, sözleşmeler ve anlaşmaların üstündedir.
4. Değişikliklere yanıt vermek, bir planı takip etmekten daha üstündür.

Burada sağda yer alan öğeler bir önem taşıırken solda yer alan öğeler (altı çizili olanlar) bizim için daha değerlidir.

1. Bireyler ve etkileşimler, süreç ve araçların üstündedir.

Yazılım sistemleri, bireylerden oluşan takımlar üzerinde kurulmalıdır. Takımların üyeleri olan programcılar, testçiler, proje yöneticileri, modelleyiciler ve müşterilerin birlikte verimli çalışmaları gerekmektedir. En önemli faktörler,

insanlar ve onları birlikte nasıl çalıştıklarıdır. Eğer bu değer alınmazsa en iyi araçlar ve süreçlerin herhangi bir yararı olmayacaktır. Araçlar ve süreçler önemlidir, fakat bu değer tek başına birlikte ve etkili çalışmadan daha önemli değildir. Bu durumu kabul etmek yönetim için zor olabilmektedir.

2. Çalışan program, kapsamlı dokümantasyonun üstündedir.

Hangi yazılım tercih edilmelidir? Elli sayfalık dokümantasyon tanımlaması istenerek yazılımın nasıl olacağının tasarlanması ya da teslim edilen çalışan gerçek bir yazılım. Muhtemelen herkes çalışan bir yazılım tercih edecektir. Bu durumda daha mantıklı olan kullanıcıların tercihlerine göre sık sık ve hızlı çalışan yazılım üretilmesidir. Dokümantasyonun ayrı bir yeri olmaktadır. Dokümantasyon, insanların anlaması için uygun şekilde yazılan değerli bir kılavuzdur. Yazılım geliştiricinin birinci hedefi, dokümantasyon yerine yazılımı oluşturmaktır. [50]

3. Müşteri ile işbirliği, sözleşmeler ve anlaşmaların üstündedir.

Müşteriler sadece ne istediklerini söyleyebilirler. Müşterilerin tam olarak istedikleri sistemi belirtmek için becerileri olmayabilmektedir. İlk zamanlarda doğru istekler alınmayabilir. Bu durumda müşterilerin istekleri değiştirmek istenebilmektedir. Müşterilerle bir anlaşma olması önemlidir. Herkesin hak ve sorumlulukları olan bir anlayışa sahip olması için bir sözleşmenin oluşturulması gerekmektedir. Başarılı geliştiriciler, müşterileri ile yakından çalışmakta ve çaba göstermektedir. Bu geliştiriciler, müşterilerinin neye ihtiyaçları olduğunu tahmin ederek süreç boyunca müşterilerini bilgilendirirler. [50]

4. Değişikliklere yanıt vermek, bir planı takip etmekten daha üstündür.

İnsanların çeşitli nedenlere dayanarak önceliklerini değiştirebilirsiniz. Sistem üzerinde çalışmanız ilerlerken, alan problemleri ve yapısal değişiklik istekleri konularında proje paydaşlarının anlaşılması gerekmektedir. İş çevresi değişebilmektedir. Zaman içinde teknolojik değişikliklere rağmen daha iyisi her zaman olacaktır. Değişim, yazılım geliştirmenin bir gerçeğidir. Bu değişim yazılım geliştirme sürecine yansıtılmalıdır. Bununla birlikte bir proje planı esnek olmalıdır. Durum değişikliklerine göre değiştirme yapılmazsa proje planından uzaklaşılabilir. [50]

IV.2.3. Çevik İlkeleri

Çevik bildirida yer alan değerler oniki ilkeye sahip bir dizi tanımlamaktadır. Bu ilkeler çevik sürecin karakteristik özelliklerini temsil etmektedir.

Tablo IV.2 Çevik İlkeler [52]

Agile Principles
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. 2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. 3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. 4. Business people and developers must work together daily throughout the project. 5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. 6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. 7. Working software is the primary measure of progress. 8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. 9. Continuous attention to technical excellence and good design enhances agility. 10. Simplicity - the art of maximizing the amount of work not done - is essential. 11. The best architectures, requirements, and designs emerge from self-organizing teams. 12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Türkçesi ve açıklaması:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Öncelikli olan erken ve sürekli olarak yazılımın teslim edilmesi yoluyla müşteriye memnun etmektir.

Bu ilke, açık görünen ve genellikle geleneksel yazılım geliştirme kurallarının dışında yer almaktadır. Çalışan bir yazılım hazırlayarak müşterilerin hatırlanması önemlidir. Bu ilke ile müşteriden dokümantasyon düzenlemesi istenmez ve yazılım prototipi hazırlanmaz. Önceden çalışan bir yazılımın teslim edilmesi ile müşterinin memnun edilmesi sağlanabilir. Böylece proje sonunda müşterinin istediği yazılıma ulaşması sağlanmaktadır.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Sonradan gelen geliştirme ile ilgili değişiklikler hoş karşılanmalıdır. Çevik süreçler, müşterinin rekabetteki avantajını değiştirmek için kullanılır.

Müşteriler, dinamik bir pazarda rakip olarak yerini almaktadır. Bu nedenle onların pazarda rekabet avantajı sağlamaları için gereksinimlerini değiştirmek gerekebilir. Unutulmaması gereken bir nokta ise geliştiricinin değişen gereksinimleri hoş karşılamasıdır. Çevik süreçlerde, erken ve sık teslim edilen çalışan yazılımlar için sonradan değişen gereksinimler tam olarak yapılmalıdır. İteratif ve zaman kutulama (timeboxing) teknikleri, mimariye sürekli dikkat etme ve tasarım güncellemek için değişen gereksinimleri karşılamada önemlidir.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Kısa zamanlar tercih edilerek birkaç haftadan birkaç aya kadar sürelerde sık sık çalışan yazılım teslim edilmelidir.

İlk zamanlarda geri bildirim alınmaması için müşteriye yazılım gösterilmeyebilir mi? Çoğu durumda bazen küçük ama genellikle büyük geri bildirim alınabilmektedir. Beceri, yazılımın erken teslimi ve böylece erken geri bildirim alınabilmesidir. Erken geri bildirim ile süreç boyunca yeniden çalışma kaydedebilir. Bu bölüm, erken ve sık teslimde çalışma döngülerinin uzunluklarını belirtmektedir. Zaman zaman projelerde karşılaşılan dört aylık döngülerle artımlı geliştirme olarak çalışma yapılabilir. Eğer ihtiyaç var ise müşterilerin her ay değişiklikleri kabul edilebilir. Yazılım geliştirme takımı, değişimler için gelen talepler arasında eşleştirme yapabilir. Kısa geri bildirim döngüleri daha iyi olmaktadır.

4. Business people and developers must work together daily throughout the project.

İş adamları ve geliştiriciler proje süresince birlikte günlük çalışırlar.

Bu ilke müşteri olmayan dikkatli iş adamları için söylenmiştir. Çoğu durumda günlük olarak müşteri ile birlikte çalışmak mantıksız olabilmektedir. Müşteriler genellikle birden çok işi yürütmek durumunda olabilmektedir. Müşteri ile birlikte çalışma, gereksinimler hakkında herşeyi bilmek değildir. İş adamları ile genellikle iş hakkında geliştiricilerin ihtiyaçlarını karşılayan yakınlık amaçlanmaktadır. Bu yakınlığı analistler, ürün müdürleri veya program yöneticileri yapabilmektedir. Kilit nokta, bir gün bile projenin hedefinden çıkmadan gitmesini sağlamak için geliştiriciler ve iş adamları arasında sürekli olarak iletişimin muhafaza edilmesidir.

Geleneksel yazılım süreçlerinde bu durum farklıdır. Geliştiriciler ve müşteri arasında böyle bir durum olmamaktadır. Geliştiriciler ve müşteri arasında görünmeyen bir duvar mevcuttur. Proje sürecinde müşteriden gelen gereksinim değişiklikleri hoş karşılanmaz.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Projeler motivasyon sahibi bireylerle yapılmalıdır. Onların ortam ile ihtiyaç duydukları desteği sağla ve iş yapabilmeleri için güven.

Proje ekipleri, motive edilmeye ve güvenilmeye ihtiyaç duymaktadır. Motivasyon, proje üzerinde çalışan takımın iş güvenine bağlı olmaktadır. Çevik projeler, bireylerden meydana gelmektedir. Geliştiriciler, kendi has özellikleri ile takımın bir parçası olmaktadır. Geliştiriciler, proje çalışmalarında birbirlerine yardım edebilmektedir. Aralarında ast üst ilişkisi bulunmaz.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Bilgi iletiminde en verimli ve etkili yöntem bir geliştirme ekibinde yüz yüze görüşmektir.

Sohbet veya telefon asla yüz yüze iletişimin yerini tutmamaktadır. Bu bağlamda birçok e-posta ve anlık mesajlaşma üzerinden iletişimde kayıplar olmaktadır. Sözsüz iletişim ile bu durum daha kötü bir hal alabilmektedir. Aynı zamanda yüz

yüze iletişim, daha az resmi belgelerle çalışmanızı sağlamaktadır. Yüz yüze iletişim, bilginin iletilmesi esnasında en az kayba uğradığı yöntemdir.

7. Working software is the primary measure of progress.

Çalışan yazılım ilerlemenin temel ölçüsüdür.

Müşteri, öncelikle yazılımın çalışması ile ilgilenmektedir. Neden ilerlemeyi ölçmek istenir? Bugün, çoğu yazılım geliştirme çabalarında süreç, müşteri gereksinimlerine göre ölçülmektedir. Gereksinimler tamamlandığı zaman yöneticiler için projenin yüzde otuzu tamamlanmış olmaktadır. Dünya çapında bir planda bu doğru olabilmektedir. Projede, gerekli işlevselliğin yüzde otuzu kodlanmış, test edilmiş, dağıtılabilir olduğunda yüzde otuzu tamamlanmış olmaktadır. Bu dünya çapında çevik değer ile geleneksel plan arasında temel bir farktır.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Çevik süreçler sürdürülebilir gelişmeyi destekler. Sponsorlar, geliştiriciler ve kullanıcıların sabit bir tempoda beraber çalışmaları gerekir.

İnsanlara çok uzun çalışma saatleri koymak projenin ilerleme hızını yavaşlatmaktadır. Proje içinde daha fazla hata oluşmakta ve üretimi azaltmaktadır. Bir tetik ve meşgul personel, yorgun ve hantal bir personelden daha çeviktir. Proje düzeni ile birtakım yanlışların ortaya çıkması uzun çalışma saatlerinin bir yansıması olabilmektedir.

Geleneksel gelişmede, çoğu takım çalışmasında projenin sonuna doğru olan çalışmalarda geliştirici iki saatlik öğle araları alabilmektedir. Bunun nedeni ilerlemede proje etkinliklerinin, proje süresi boyunca dağıtılmasıdır. Bu durum geliştiriciler için projenin başında olmasından farklı bir şey değildir. Geliştiriciye verilen sık sık teslimat programlarıyla her şeyi omuzlarına koymak demektir.

Çevik geliştirme ekibinde, iki haftada bir teslim ya da ilk iterasyon ile gelişme başlamaktadır. Çabalar, proje yaşam çevrimi boyunca daha fazla dağıtılmış olmaktadır. Bu durum takımda sabit bir geliştirme hızına yol açmaktadır. İlave çalışma saatleri hoş karşılanmaz. Proje ilerledikçe iş azalmaktadır.

9. Continuous attention to technical excellence and good design enhances agility.

Teknik mükemmelliğe sürekli özen gösterilmesi ve iyi tasarım çevikliği kuvvetlendirir.

Başarılı bir jimnastikçi güçlü kaslara ihtiyaç duymaktadır. Benzer şekilde teknik mükemmellik, Çevik yazılım geliştirme süreci için bir temeldir. Örneğin, genişletilebilir tasarımlar ve mimariler yaparak evrimsel bir şekilde daha kolay ürün oluşturabilir. Otomatik test çevrimleri, yeniden yapılandırmada sistemin bir parçasını düzeltmede diğer parçaların etkilenmemesi için gereklidir. Sürekli entegrasyonda, yazılımın teminat altına alınması önemlidir. Bu teminat yazılımın her değişiklikten sonra çalışmasıdır.

10. Simplicity - the art of maximizing the amount of work not done - is essential.

Sadelik (Çalışma miktarının arttırılması üretkenlik getirmez) esastır.

Kod yoksa hata yok demektir. Daha fazla kod yazarak daha fazla hata koduna sahip olabilirsiniz. Bazı geliştiriciler, belli başlı ağır geliştirme çerçeveleri ile sistem altında alt yapıların oluşturulması eğilimindedirler. Sade olan kod yapısında, geliştirme ve bakım kolay olmaktadır.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

En iyi mimariler, gereksinimler ve tasarımlar kendi kendine organize olabilen takımlardan ortaya çıkar.

Geleneksel yazılım geliştirmede, analistler gereksinimleri yazmak ve mimarlar sistem mimarisi hazırlamak ile görevlidir. Gereksinimler ve mimarilerde, takım kendi içinde iletişimdedir. Çevik dünyasında, kendi kendine organize olan takımlar desteklenmektedir. Normalde kendi kendine organize, takım olarak görev almada kimin ne yapacağı belirtilmeden görevi tamamlamadır. Bu yapıda, takımda işbirliği ile aynı zamanda takım içinde bilgi iletiminin artması sağlanır.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Düzenli aralıklarla takım nasıl daha etkili olabileceğini kendi içinde sorgular. Deneyimler doğrultusunda gerekli uyum ve değişiklikler yapılır.

Sürekli iyileştirme olmaz ise kişiler ve kuruluşlarda kalan statükoya (geçmişten beri süregelen yapı) devam edilmektedir. Neyi iyi yaptığınızı ve yapılabileceğini tanımlamak gerekmektedir. Aynı şekilde neyin kötü yapıldığını tanımlamak gerekmektedir. Takım içinde toplantılar yardımıyla fikir alışverişi ve sorgulama yapılabilir. [45,49]

IV.2.4. Geleneksel Sürecin Problemleri

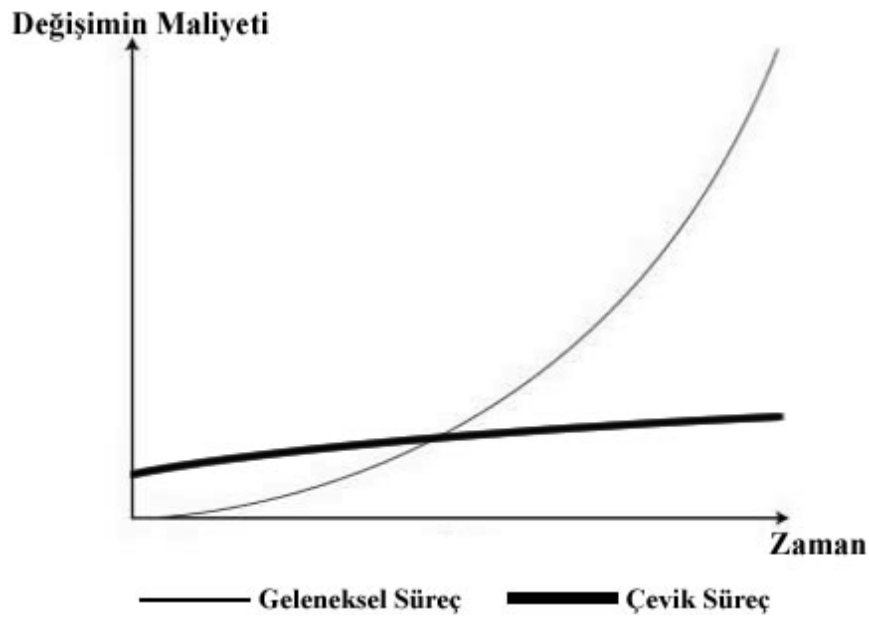
Geleneksel süreçlerden en çok kullanılan Şelale (waterfall), prototipleme (prototyping), artımlı (incremental), sarmal (spiral) modellerinde, planlama ve dokümantasyon önemlidir. Modellerin yapıları farklı olmakla birlikte yazılım geliştirme süreçleri analiz, tasarım, kodlama, test ve onarım sırasıyla gitmektedir.

Geleneksel modellerde ortak olan problemler şunlardır.

- Proje öncesi belirlenen bütçenin aşılmasına rağmen yazılım kalitesi yeterli olmamaktadır.
- İlk aşamalarda yoğun dokümantasyon nedeniyle detay ağırlıklı analiz ve tasarım çalışmalarında çok zaman kaybedilmektedir.
- Verimlilik projenin başında değilde sonunda yakalanabilmektedir.
- Proje süresince müşteriden gelen gereksinimlerle yapılacak değişiklikler projeyi riske sokabilmektedir.
- Proje ile ilgili sorunların sürecin sonlarına doğru tespit edilmesi ve düzeltme ile gelen maliyetlerin artmasına neden olmaktadır.
- Projenin başlangıcında yapılan proje planı bir süre sonra geçerliliğini yitirebilmektedir. Plana uymak için ekibin fazla mesai yapmak zorunda kalması hata riskinin artmasına ve motivasyonun azalmasına sebep olmaktadır.
- Geliştirilen yazılım, müşteri tarafından yetersiz bulunup ekibe yeni ve projenin en başında kapsamda belirtilmeyen gereksinimler gelebilmektedir. Değişiklikler sonunda yazılımda farkedilmesi zor hatalar ortaya çıkmaktadır.
- Projenin başında yapılan tasarım değişiklikleri kaldıramayarak proje sonunda kötü tasarıma sahip ve hata içeren kodlar ortaya çıkmaktadır.

- Değişikliklerin maliyeti proje ilerledikçe artmaktadır. Değişiklikleri önlemeye çalıştıkça müşteri ile çatışmalar ortaya çıkmaktadır.
- Ekip içi bilgi paylaşımı yapılamamakta ve projeden ayrılan elemanlar risk haline gelmektedir. Ekipteki roller etkin bir işbirliği yapamadığından, çekişmeler oluşmaktadır. [54]

Geleneksel süreçlerde, proje başlangıcında her ayrıntıya girilemediği için geliştirilen yazılım sistemleri müşteri gereksinimlerini karşılamamaktadır. Bunu engellemek için projenin başlangıcında analiz için çok zaman harcanarak müşteri gereksinimleri detaylı dökümantasyonla belirlenmektedir. Proje başlangıcında oluşturulan gereksinimler değişen piyasa ve rekabet koşulları gereği aynı olmayabilir. Geleneksel süreçler, bu durumları dikkate almayarak müşteri değişikliklerini en aza indirmeye çalışmaktadır. Süreç içinde gelen değişiklik talepleri maliyeti yükseltmektedir.



Şekil IV.2 Çevik ve Geleneksel Süreç [55]

IV.2.5. Çevik Sürecin Yararları

Çevik süreçler, önde gelen yazılım profesyonellerinin gerçek hayattaki proje deneyimleri ile büyümüştür. Geliştiriciler, proje üzerinde ve sonrasında geleneksel şelale geliştirmesinin sınırlamaları ve zorluklarına ait deneyimlere sahiplerdi. Çevik süreçler yaklaşımına terfi etme geleneksel yazılım geliştirme ile ilgili düzenlemelere doğrudan yanıt olmaktadır.

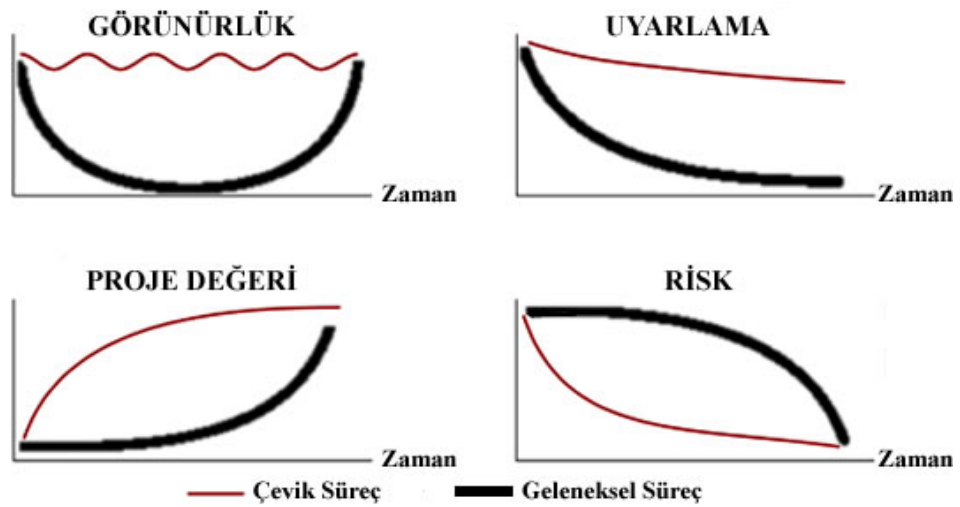
Çevik süreç, en basit formda takımlara yardım için hafif bir çerçeve sunmaktadır. Bu hafif çerçeve, sürekli gelişen fonksiyonel ve teknik manzarada iş

değeri ve hızlı teslimata odaklanmadır. Bu odaklanma sonucu olarak, kuruluşların yazılım geliştirmede toplam risk ilişkisinin önemli ölçüde azaltılması sağlanmıştır. Çevik süreç, sürekli planlama ve geri bildirim ile ilk proje değeri teslimatının hızlandırılmasını sağlamaktadır. Bu değer geliştirme süresince giderek geliştirilmektedir. Bu tekrarlayıcı planlama ve geri bildirim sonucunda takımlar süreç boyunca değişen gereksinimlerin kolayca adapte edilebilmesi için sürekli temsilat yazılımı hazırlamaktadır. Çevik sürecin ardından bir projede sonuç olarak, iş ve müşteri ihtiyaçlarını çok iyi gideren bir yazılım sistemi ortaya çıkmaktadır.

Çevik süreçlerde müşteri açısından en önemli özellikler ilk olarak bitirilmektedir. Çalışan yazılım, projedeki ilerlemenin ölçütü olmaktadır. Geleneksel yazılım geliştirmeden farklı olarak ilk sonuçlar projenin başlangıcında görülmektedir. Müşteri geliştirme sürecinde aktif bir rol alarak projenin doğru yönde ilerlemesini sağlamaktadır. Çevik süreçlerde iterasyon bazında çalışmalar sürdürülmektedir. Her iterasyon bir ile dört hafta arasında bir zamandan oluşmaktadır. İterasyonlarda, şelale modelinde yer alan adımlar bulunmaktadır. [56,57]

Şekil IV.3’de çevik ve geleneksel (şelale) geliştirme süreçleri arasındaki farklılıklar gösterilmektedir. Teslimatta çalışan, test edilmiş, dağıtabilen bir yazılımın çevik süreç ile teslim değeri artmaktadır. Çevik süreçte teslimat aşaması ile yazılım yaşam döngüsünün erken aşamalarında görünürlük ve uyarılama değeri artırılarak, proje risk değeri azaltılmaktadır.[53]

ÇEVİK SÜREÇ DEĞER İFADELERİ



Şekil IV.3 Çevik Süreç Değer İfadeleri [53]

Görünürlük:

Proje geliştirme yaşam döngüsü boyunca paydaşlarla işbirliği ve doğrulama.

Müşteri ve geliştiriciler arasında sürekli iletişim olmaktadır. Geliştiriciler sorunlarını müşteriyle birlikte kısa sürede çözüme kavuşturabilirler. Sorunların paylaşılması çevik süreçlerin değişime açık olduğunu göstermektedir. Projelerde, müşteri ile olan iletişim sayesinde proje değişerek gelişmektedir. Proje başlangıcında genellikle müşterilerin belirli bir fikirleri olmamaktadır. Müşteri fikirleri, projenin ilerlemesi ile şekillenmektedir. Çevik süreçler ile proje bittiğinde müşteri gereksinimlerini karşılayan programlar ortaya çıkmaktadır.

Proje Değeri:

Daha çok teslimat ile iş değeri ölçülebilir.

Çevik süreç içersinde her iterasyon sonunda küçük programlar oluşturulmaktadır. Kısa aralıklarla müşteriye yeni sürüm programlar sunularak geri bildirim sağlanmaktadır. Yapılan geri bildirimler ile projenin durumu hakkında bilgi verilmektedir. Proje süresince çalışan programların müşteriye sunulması müşteri memnuniyetini arttırmaktadır. Müşterilerin tercihleri konusunda yapılan araştırmalarda çalışır durumda olan tamamlanmış programların tercih edildiği tespit edilmiştir. Geliştiriciler yaptıkları testler ile geri bildirim sağlayarak uygulama bileşenlerinin durumunu belirlerler. Program durumu sürekli entegrasyon ile geri bildirim olarak elde edilmektedir.

Uyarılama:

Strateji, öncelikler ve planlardaki değişikliklere hızlı yanıt verebilme.

Yazılım geliştirme süreçlerinde iterasyonlar, proje risklerini azaltarak başarıyı arttırmaktadır. Verimlilik azalan hata oranları ile yükselmektedir. Bunun nedeni proje başlangıcında geliştirilen küçük program parçaları ile proje ekibinin eksiklikleri görebilmesidir. Proje süresinde program parçaları hızlı olarak şekillenerek proje başlangıcında görülmeyen riskler oluşmadan görünür duruma gelmektedir. Çevik süreç, değişikliklere hızlı bir biçimde uyum sağlamaktadır. Değişime uyum sağlamak rekabet için büyük bir önem taşımaktadır.

Risk:

Uyarlama, değer ve görünürlük sonucu olarak toplam proje riskini azaltma.

Değişim yazılım projelerinde kaçınılmaz bir durumdur. Projeler orta çaplı olduğunda bile başlangıç durumlarına göre yüzde otuz oranında değişime uğramaktadır. Çevik süreçler değişime karşı gelmek yerine değişimi müşteriye yönelik bir avantaja çevirmeye çalışmaktadır. Çevik süreçler, parçalı yazılımlarla her adımda müşteri ile işbirliği yaparak değişim gereksinimlerinin erken fark edilmesini sağlamaktadır.

Karmaşıklığın Yönetimi:

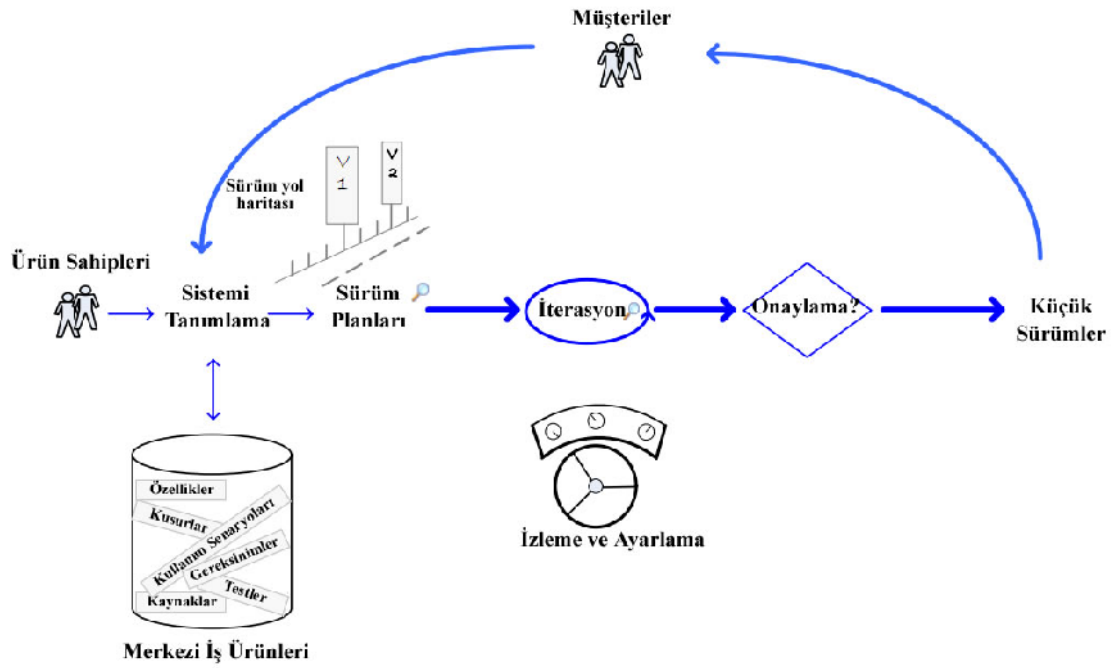
Yazılım projelerinin büyüklüğü oranında karmaşıklıklar artmaktadır. Karmaşıklık ile birlikte hata oranlarında artış olmaktadır. Çevik süreçler ile projeler küçük parçalara bölünerek daha kolay yönetilebilmektedir. Bu sayede proje büyüklüğüne bağlı olmadan projelerin karmaşıklığı azaltılabilir. Proje başlangıç seviyesinde ayrıntılı dökümantasyon ve tasarım oluşturulmaz. Geliştiriciler, test güdümlü çalışarak oluşan tasarımı testlerle gözden geçirirler. Yeterli olmayan tasarımlarda, gerekli değişiklikler ile sonraki testlerin sorunsuz çalışmasını sağlarlar.

Yüksek Kalite:

İteratif yazılım geliştirme süreçlerinde test güdümlü yazılım geliştirme anlayışı bulunmaktadır. Projenin başlangıcından itibaren testlerin yazılım geliştirme süreci içinde yürütülmesi ile hatalar büyümeden tespit edilerek hızla düzeltilmektedir. Çevik süreçlerde projenin her bir parçası kendi içinde test edilerek hata oranı azaltılmaktadır. Kod kalitesi, test güdümlü çalışma yapıldığından yüksek olmaktadır.

Geliştiriciler gün boyunca kendilerinde değişik bir takım arkadaşı seçerek (Eşli programlama) uygulamayı gerçekleştirirler. Geliştiriciler, bir süre sonra teknik olarak aynı seviye gelmektedir. Her geliştirici programın her bir bölümünde çalışacak duruma gelmektedir.

Çevik süreçlerde çalışma ortamı ve temponun iyi oluşturulması fazla mesai yapılmasına engel olmaktadır. Sürekli olan fazla mesai takım motivasyonunu kötü etkilemektedir. Projelerde meydana hatalar genelde isteksiz olarak yapılan çalışmalarda oluşmaktadır. [56,57]



Şekil IV.4 Genelleştirilmiş Çevik Süreç Modeli [61]

IV.3. ÇEVİK SÜREÇ TÜRLERİ

Çevik süreç türlerinde, süreç içinde aynı özellikler ve uygulamalar gibi benzerlikler paylaşılmaktadır. Ancak gerçekleştirme açısından her sürecin kendine özel bir yapısı, terminolojisi ve taktikleri bulunmaktadır. Bazı süreçlerde proje yönetimi ve işbirliği ağırlıklı çalışma önem kazanırken (ÖGG, Scrum, ve DSGM), diğer süreçlerde örneğin ekstrem programlamada yazılım geliştirme çalışmalarına odaklanılmaktadır.

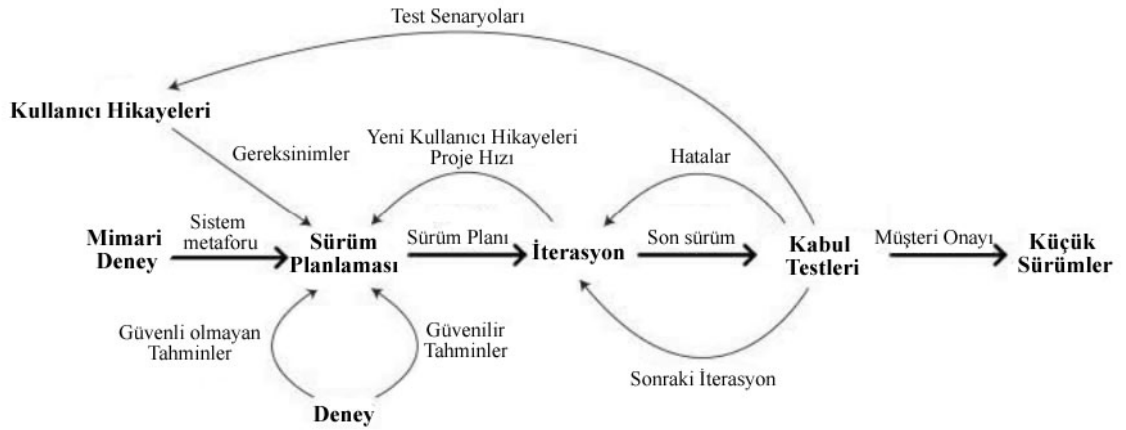
Çevik süreçlere ait bazı örnekler aşağıdaki sıralanmıştır. [31,63,65]

- Ekstrem Programlama - EP (Extreme Programming)
- Scrum
- Dinamik Sistem Geliştirme Metodu – DSGM
(Dynamic System Development Model)
- Uyarlanabilir Yazılım Geliştirme – UYG (Adaptive Software Development)
- Kristal (Crystal)
- Özellik Güzümlü Geliştirme (ÖGG) (Feature-Driven Development)

IV.3.1. Ekstrem Programlama

Çevik süreçlerden, Ekstrem Programlama doksanlı yılların sonunda Chrysler için Kent Beck, Ron Jeffries ve Ward Cunningham tarafından yapılan bir proje sonrasında oluşmuştur. En popüler ve tartışmalı çevik süreçlerden birisidir.

Ekstrem programlama, Scrum temel alınarak geliştirilen bir süreçtir. Scrum, mevcut süreçlerin tersine yazılım metotlarına odaklanmaktadır. Bu nedenle Ekstrem programlama ve Scrum bir projede beraber kullanılabilir. Ekstrem programlama hızlı, disiplinli ve sürekli olarak yüksek kaliteli yazılım teslim etme yaklaşımıdır. [15,21]



Şekil IV.5 Ekstrem Programlama Görsel Süreç Modeli [55]

IV.3.2. Scrum

Çevik süreçlerden Scrum seksenli yıllarda Kent Schwaber ve Jeff Sutherland tarafından geliştirilmiş bir süreçtir. Scrum, Rugby oyununda kullanılan bir terimdir. Scrum, kelime olarak rugby oyununda oluşturulan küçük ekiplere verilen isimdir. Oyuncular kısa bir süre için toplanarak, bir sonraki oyun hamlesi için fikir alışverişinde bulunurlar. Scrum daha çok proje yönetim metotlarına odaklanmaktadır. Projenin yapılması için gerekli bilgilerin ayrıntılı olması gerekmez. [57]

Scrum, bir işi yönetmek veya bir ürün geliştirmek için kullanılan artımlı ve iteratif bir süreçtir. Scrum, yazılım geliştirme dışında finansal ve medikal ürünlerin üretilmesinde de kullanılabilir. Proje başlangıcın dışında herhangi bir aşamada veya hata oluşan durumlarda uygulanabilir. [64]

Scrum, gereksinimleri tam olarak belirli olmayan, değişime yönelik, karmaşık yazılım projelerinin yönetimi için uygulanmaktadır. Scrum sürecinde, kullanılan

projede yapılması gereken işlemler ayrıntılı olarak belirtilmez. Bunun yerine basit kurallarla esnek bir yaklaşım sağlamaktadır.

Scrum sürecini daha iyi anlamak için bazı kavramlar aşağıda açıklanmaktadır.

Koşu (sprint): Koşu, çevik yazılım geliştirmede iterasyon kavramına karşılık gelmektedir. Scrum'da genellikle koşular otuz gün süren çalışmalarla yapılmaktadır. Koşu sonunda projeye ait alt bir ürün oluşturulur.

Müşteri (customer): Proje finanse eden kişi veya kurumdur.

Ürün sahibi (product owner): Proje ve gereksinimler hakkında takımı bilgilendiren, ürünün önceliklerini belirleyen kişidir.

Ürün talebi (product backlog): Ürün sahibi tarafından belirlenen önceliklere göre sıralı olan proje istekleridir.

Koşu talebi (sprint backlog): Bir koşuda, ürün talebi sonucunda yapılacak isteklere bağlı görevlerdir.

Scrum uzmanı (scrum master): Uygulamalar konusunda uzman olan proje ekibinin ilerlemesini destekleyen, proje ekibinde problemlere çözüm bulan kişidir.

Günlük scrum toplantısı (daily scrum meeting): Günlük olarak gerçekleştirilen ve çok kısa süren (15 dk) ekip toplantısıdır.

Koşu planlama toplantısı (sprint planning meeting):

Scrum ekibi ve ürün sahibinin birlikte ürün taleplerini tekrar değerlendirildiği ve isteklerin önem sırasına göre bir sonraki koşu seçilerek düzenlendiği toplantıdır.

Koşu değerlendirme toplantısı (Sprint review meeting): Koşu bitiminde olan koşuda üretilen yazılımın değerlendirildiği toplantıdır.

Geriye dönük koşu değerlendirme toplantısı (Scrum retrospective meeting):

Scrum ekibinde, belirli sürelerde yapılan sonraki koşular için verimliliğin artırılması için yapılan toplantılardır.

Scrum İş Akışı:

İş akışı, özetle aşağıdaki gibi sıralanabilir.

- a) Vizyonun (görünümün) belirlenmesi,
- b) Ürün talebinin oluşturulması (veya tekrar gözden geçirilmesi)
- c) Koşu talebinin oluşturulması,
- d) Koşu içerisinde günlük Scrum toplantıları,
- e) Koşu sonrasında değerlendirme toplantısı
- f) Geriye dönük koşu değerlendirme toplantısı.

Projenin bitimine kadar yukarıdaki adımlar, 2. adımdan başlanarak 6. adıma kadar tekrarlanarak sürdürülmektedir.

Vizyonun belirlenmesi, ürün sahibinin görevi olup her projede belirlenmesi gereklidir. Vizyondan sonra ürün sahibi tarafından ürün talebi belirlenerek öncelikler tanımlanmalıdır.

Scrum, 30 gün süreye sahip olan koşulardan oluşmaktadır. Ürün talebinde, her koşu öncesi scrum takımı ve ürün sahibi toplantı yaparak biraraya gelmektedir. Ürün talebindeki değişiklikler toplantıda görüşülerek vizyona uygun öncelikler belirlenmektedir.

Koşu başladıktan sonra amaç, toplantı sonucunda ortaya çıkan istekler ile koşu sonunda çalışan programın oluşturulmasıdır. Takım koşu süresince kendi içinde organize olup kendini iyi yöneten bir çalışma gerçekleştirir.

Koşu sırasında proje takımı, günlük, ayaküstü, kısa toplantılarla bir araya gelerek uyumluluğu sağlarlar. Bu toplantıda proje takımının aşağıdaki soruları cevaplayarak birbirleriyle paylaşması gereklidir.

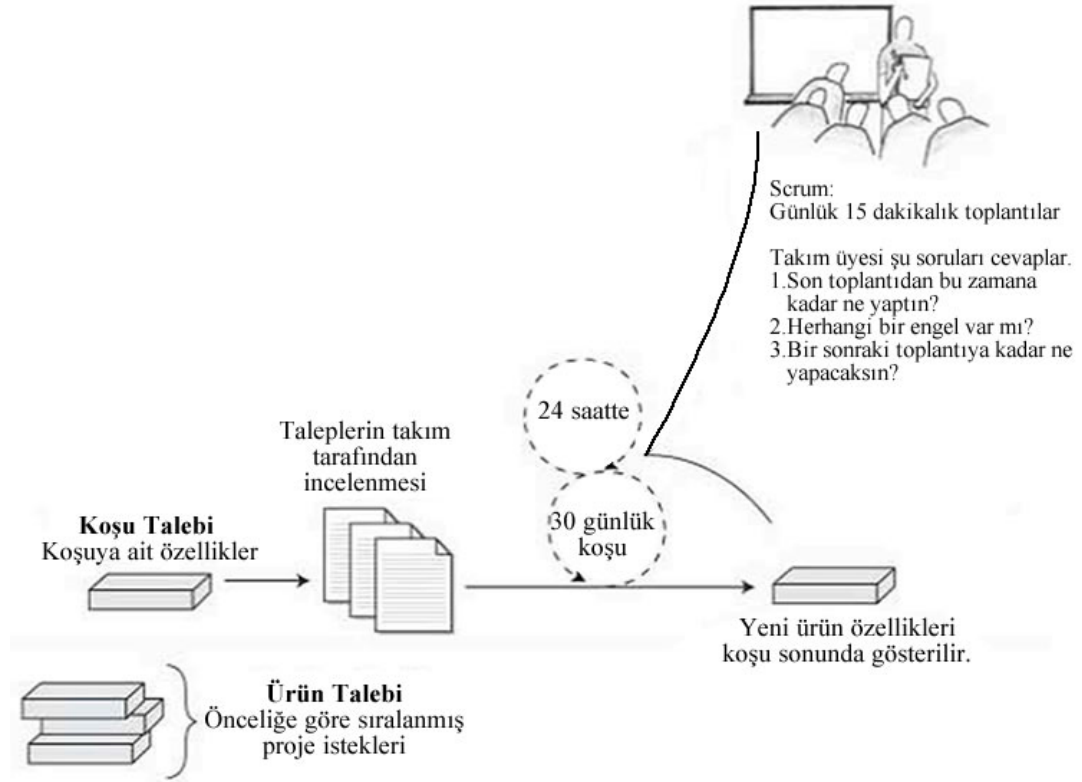
- Dün nasıl bir çalışma yaptık?
- Bugün nasıl bir çalışma yapacağız?
- Karşılaşılan ya da karşılaşma potansiyeli gördüğümüz problemler nelerdir?

Bu toplantıda proje takımında, takım uyumunun yaratılması ve problemlerin öngörülebilmesi amaçlanmaktadır. Sonraki aşamada, çalışan programı göstermek için proje takımı, ürün sahibi ve müşteri koşu sonunda değerlendirme toplantısı yaparlar. Toplantının amacı, koşu sonuçlarının paylaşılmasıdır. Projeye devam edildiği sürece, yukarıdaki adımlar yeni bir koşu planlama toplantısı ile tekrarlanır.

Scrum sürecinin getirileri:

Çevik yazılım geliştirme süreçlerinden Scrum, çevik yazılım geliştirme süreçleriyle ilgili olarak daha önceden belirtilen faydaları içinde barındırmaktadır. Aşağıdaki özellikleriyle Scrum, tüm dünyada yaygın bir şekilde kullanılmaktadır.

- a) İterasyon ile projenin parçalara ayrılarak karmaşıklığın azaltılması sağlanır.
- b) Proje geliştirme sürecinde sürekli olarak müşterinin bulunması, bilgi alışverişi ile birlikte müşteri memnuniyetini getirmektedir.
- c) Öncelikli isteklerin ürün sahibi tarafından belirtilerek bunların çözülmesi ve rekabet avantajının müşteriye verilmesi sağlanmaktadır.
- d) Önceden risklerin fark edilerek azaltılması sağlanabilir. [76]



Şekil IV.6 Scrum Görsel Süreç Modeli [55]

IV.3.3. Dinamik Sistem Geliştirme Metodu

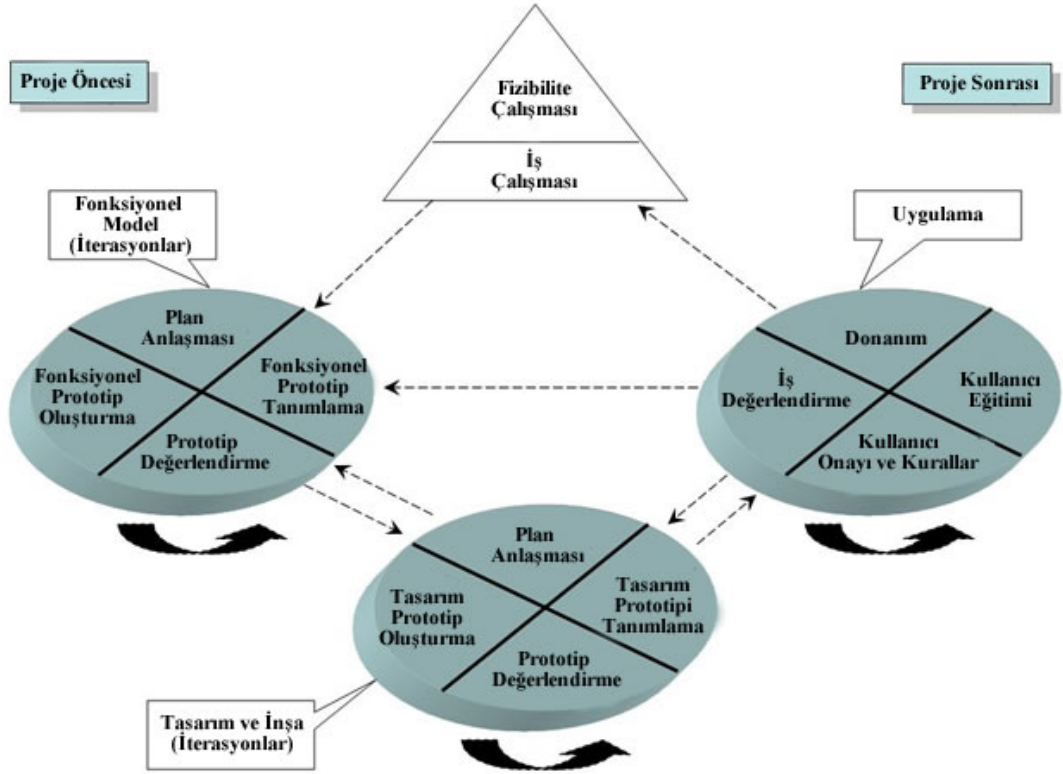
DSGM süreci, Ekstrem programlama ve Scrum gibi süreçlerle karşılaştırıldığında daha fazla ayrıntılıdır. Genelde devlet ihaleleri gibi yazılım projelerinde sıklıkla kullanılmaktadır. [54]

DSGM süreci, prototiplemeye önem veren, fizibilite ve iş alanı çalışmalarıyla başlamaktadır. DSGM sürecinin uygunluğuna fizibilite çalışmasıyla karar verilmektedir. İş alanı çalışmasıyla ise geliştirimin yapılacağı iş alanını analiz etmek amacıyla bilgi toplanmaktadır. Sonuç olarak ana hatlarıyla sistem mimarisi ve proje planı ortaya çıkarılmaktadır. DSGM sürecinin diğer bölümleri her biri iteratif ve artımlı olan 3 alt süreçten oluşmaktadır.

Fonksiyonel model çevrimi: Plan için gerekli dokümantasyon ve fonksiyonel prototipleri içermektedir.

Tasarlama ve İnşa çevrimi: Tasarım prototiplerini içermektedir.

Kodlama çevrimi: Uygulama kodları, son kullanıcı eğitimini ve kabul sürecini içermektedir.



Şekil IV.7 DSGM Görsel Süreç Modeli [47]

DSGM, karar vermede etkili bir takım, aktif kullanıcı etkileşimi, sık aralıklı teslimler, iş amaçlarına tam uyumu, gereksinimlere uygun çalışmayı, sıkı işbirliğine dayalı takım çalışmasını, tüm artımlar boyunca test yapmayı içeren prensipleri temel almaktadır. Diğer çevik süreçler gibi kısa sürelerle sınırlı 2 ile 6 hafta süren iterasyonları kullanmaktadır. Müşteri ile etkileşimin yüksek olduğu, kullanıcı grubunun net bir şekilde tanımlanabildiği, kesin zaman kısıtlarının olduğu, gereksinimlerin öncelik sırasına göre dizilebildiği, gereksinimlerin durağan ve tam tanımlı olmadığı durumlarda kullanılabilir. Küçük projelere uygulanabildiği gibi birden fazla küçük grubu içeren büyük projelere de uygulanabilmektedir. [65-68]

IV.3.4. Uyarlanabilir Yazılım Geliştirme

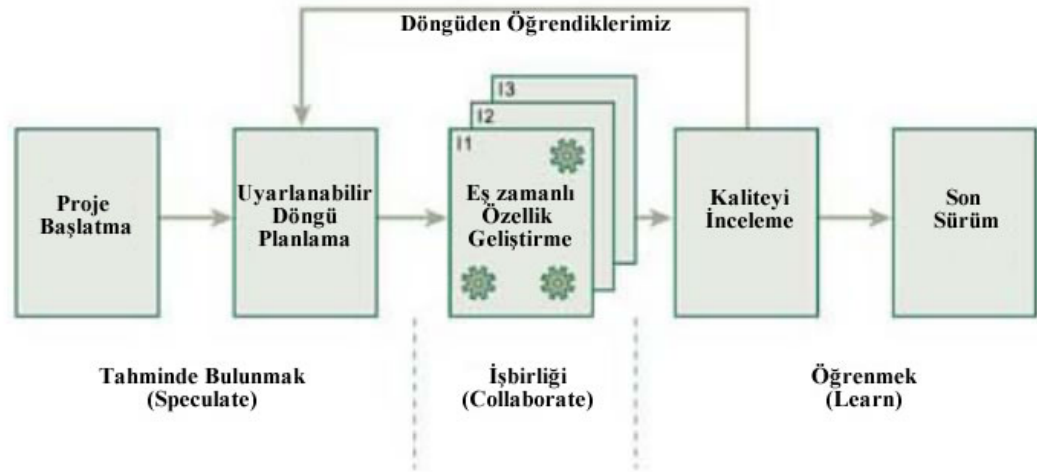
Yazılım sektörünün hızla değişmesi ve karmaşıklaşması ile başarı sağlamak için UYG sürecinin felsefesi değişimle savaşmak yerine, değişimle çalışmaktır. Bunu sağlamak için değişimleri içeren ve cevap veren, uyarlanabilen uygulamalar ile hızlı ve uyumu yüksek olan insanlara, takımlara ve organizasyonlara ihtiyaç duymaktadır. Sürece yoğunlaşmak yerine işbirliği ile yapılan ürüne yoğunlaşmayı temel alan bir yapısı bulunmaktadır.

UYG sürecinde belirsizliklere çözüm bulabilmek için insanların arasında güçlü bir işbirliğine ihtiyaç vardır. Süreç boyunca yönetim, insanları yönlendirmek yerine iletişimi güçlendirerek insanların çözümlerde yaratıcı olması sağlamayı çalışmaktadır.

Bu süreçte,

- Projenin vizyonunu, verilerini ve tanımlarını içeren ortak dokümantasyon tutulmaktadır.
- Oturumlar, müşteri odaklı gruplar ve süreç iyileştirme ve geliştirme toplantıları gibi işbirliği teknikleri uygulanmaktadır.
- İşbirliği tekniklerinin planlanmasından sorumlu bir organizatör tanımlanmaktadır.

UYG, özellikle sürekli öğrenme ve işbirliği çalışmaları üzerinde durarak, iteratif ve artımlı bir süreci desteklemektedir. [31]



Şekil IV.8 UYG Görsel Süreç Modeli [33]

İteratif geliştirme, özellik-bileşen tabanlı planlama ve müşteri odaklı grup toplantıları önemli uygulamalarıdır. UYG süreci 3 bölümden oluşur. Bunlar tahminde bulunma, işbirliği ve öğrenmedir. Tahminde bulunma bölümünde proje amacı ve geliştirmede gerekli bilgiler belirlenerek uyarlanabilir döngü planlaması yapılmaktadır. İşbirliği bölümünde eş zamanlı bileşen geliştirimi yapılmaktadır. Öğrenme bölümünde kaliteyi incelemek için toplantılar ve proje başında müşteri ve geliştiricilerin ortak uygulama toplantıları yapılmaktadır. Önemli olan sürecin nasıl geçtiğinden çok ne öğrenildiğidir. [18,65,66]

IV.3.5. Kristal

Kristal ailesi süreçleri, (Crystal Clear, Cryristal Orange ve Crystal Orange/Web) farklı projelerin farklı süreçlere ihtiyacı olduğu fikrinden ortaya çıkmıştır. Bu süreçler projelerde çeşitliliğe, projedeki kişi sayısı ve hata sonuçları olmak üzere iki farklı açıdan yaklaşmaktadır. Her süreçte, konfor (C-Comfort), isteğe bağlı kapital (D- Discretionary Money), gerekli kapital (E-Essential Money) ve hayat noktası (L-Life) olmak üzere 4 farklı seviye bulunmaktadır. [62,66,68]

Sistem Kritikliği				
	L6	L20	L40	L80
E6	E6	E20	E40	E80
D6	D6	D20	D40	D80
C6	C6	C20	C40	C80
Açık	Açık	Sarı	Turuncu	Kırmızı
				Proje Büyükliği

Şekil IV.9 Kristal Metodoloji Boyutları [66]

Crysal Clear, altı kişilik küçük gruplar için, Crystal Orange 10 ile 40 kişilik geliştirme grupları ile 1-2 sene sürecekt orta büyüklükteki projeler için kullanılmakta ve birden fazla grup içermektedir. Crystal Orange/Web ise web tabanlı projelerin geliştirilmesinde kullanılmaktadır. [31,66]

Kristal, iletişim odaklı, insan gücüne önem veren ve kendi kendine adapte olabilen süreçler topluluğudur. Bu süreçler aşağıdaki durumları temel almaktadır. [60]

- Her proje farklı birtakım kural, uyum ya da yönleme ihtiyaç duymaktadır.
- Proje çalışmaları, insan ile ilgili konularda hassastır ve bu konuların gelişimine bağlı olarak daha da gelişir; insanlar daha iyi oldukça, grup içi çalışmalar daha da iyi olmaktadır.

- c) Etkin iletişim ve sık yapılan teslimler orta seviye iş ürünlerine gereksinimi azaltmaktadır.

Kristal süreçlerin iki kuralı bulunmaktadır.

- a) 1 ile 3 ay arasında değişen en fazla 4 ay süren artımlarla geliştirim yapılmaktadır.
- b) Her bir artım öncesi ve sonrası başarılar ve başarısızlıkların dile getirildiği fikir seminerleri düzenlenmektedir. [31]

Kristal süreçlerden her biri yöntemin ağırlığına göre beyaz, sarı, turuncu, kırmızı renklerinden biri ile gösterilmektedir. Renk ne kadar koyu ise süreç o kadar ağır olmaktadır. Projenin büyüklüğü ve kritikliğine bakılarak uygun olan renkteki süreç seçilmektedir. Takım, artımda kodlanacak gereksinimleri ve iş takvimini belirlemektedir. Her artım, kodlamayı içeren yapılandırma, gösterim ve yeniden gözden geçirmeleri içermektedir. Süreç, takımın teslim ettiği ürünlere ve zamana bağlı duruma göre izlenerek takip edilebilir. Takım, kendine ait bir süreç oluşturmak için kendi kendini kontrol etmektedir. Aynı zamanda sürecin kendi kendine gelişimini sağlayacak iterasyon sonu incelemelerine ağırlık verilmektedir. [31,66]

Yazılım geliştirme süreci belirtilmediğinden EP ve Scrum ile entegre kullanılabilmektedir. Çok kritik projelerde kullanımı çok uygun değildir, aynı ofiste çalışan geliştiricilerle iletişimin fazla olduğu küçük ve orta ölçekli projelerde kullanılabilmektedir. [66]

IV.3.6. Özellik GÜDÜMLÜ Geliştirme

Jeff DeLuca tarafından doksanlı yılların sonunda geliştirilmiş bir çevik süreç türüdür. ÖGG, yazılım özelliklerine odaklı olarak çalışmaktadır. Sisteme yeni bir özellik kazandırılmadan önce, detaylı bir tasarım çalışması yapılarak bu özelliği kapsayan mimarî yapı oluşturulur. Bu yüzden ÖGG daha çok tasarım odaklı işleyen bir çevik süreçtir. [57]

ÖGG model tabanlı, küçük yenilemelerden oluşan bir süreçtir. Planlama, öncelik belirleme, tasarım ve geliştirme süreçlerinde özellik adı verilen müşteri için anlamlı, küçük ve işlevsel gereksinim parçalarını kullanılarak geliştirme yapılmaktadır. Artımlı olmayıp dinamik olan süreç, özelliklerin gerçekleştiriminin müşteri tarafından değerlendirilmesi aracılığıyla, artıma gerek kalmadan, küçük iterasyonlarla geri bildirim almayı amaçlamaktadır.

ÖGG sekiz temel uygulama içermektedir:

1. Problem alanının incelenmesi, tanımlanmasını ve özelliklerin belirlenmesini içeren *Alan Obje Modellemesi*,
2. Tanımlanmış bir dizi özellik üzerinden geliştirme ve ilerlemeyi izlemeyi içeren *Özelliklere Göre Geliştirme*
3. Bakımı ve devamlılığı kolaylaştırmak için her sınıfın bir geliştiriciye aitliğini belirten *Bireysel Kod Sahipliği*
4. Küçük ve dinamik olarak oluşturulan *Özellik Takımları*
5. Kontrol listesi tabanlı toplantıları ve kod standartlarını içeren *Denetleme*,
6. Yeni bir özelliğin eklenebileceği, her an için çalışan bir sistemi garanti eden *Düzenli Yapılar*
7. Versiyonlama takibini sağlayan *Konfigürasyon Yönetimi*
8. Tüm önemli organizasyonel yapılarına ilişkin tamamı hakkındaki bilgileri iletmeyi içeren ilerlemeleri *Rapor Etme*.

ÖGG beş süreç içermektedir. Bu süreçlerden ilk üçü proje başında yapılmaktadır.

1. Çekirdek metotların ve projenin şeklinin belirlendiği genel bir model geliştirme
2. Özellikler listesi oluşturma
3. Özelliklere göre planlama
4. Özelliklere göre tasarlama
5. Özelliklere göre uygulama

Bu süreçler içinde küçük takımlar modellemeye önem vererek özellik setleri üzerinde çalışırlar. Diğer adapte edilen yöntemler gibi gözle görülür işlevselliğe sahip teslimleri içeren iki haftalık küçük iterasyonlardan oluşmaktadır. [48,67].



Şekil IV.10 Özellik GÜdümlü Geliştirme Görsel Süreç Modeli [47]

IV.4. ÇEVİK YAZILIM GELİŞTİRMEDE ÖLÇEKLENDİRME

İlk günlerde Çevik süreç, Çevik geliştirme uygulamalarda küçük kapsamda ve nispeten basit uygulanmaktaydı. Bugün durum önemli ölçüde değiştirilmiştir. Organizasyonlar, projelerini daha geniş bir şekilde düzenlemek için çevik geliştirmeyi uygulamak istemektedirler. Çevik sürecin dolayısıyla birçok iş ve organizasyon ile uyulanmasına ihtiyaç vardır.

Önemli bulunan Çevik ölçeklendirme faktörleri şunlardır:

Coğrafi Dağılım: Bir takım nasıl dağınık olabilir? Aynı bina içinde farklı katlarda, aynı şehir içinde farklı yerlerde ya da farklı ülkelerde takım elemanları dağınık olabilmektedir. Bu durum aniden etkili işbirliğinde, takım iletişimini kötü bir duruma getirebilir.

Takım Boyutu: Çevik süreçlerde, on veya onbeş kişilik küçük takımlarla çok iyi çalışmalar yapılabilir. Eğer takım çok daha büyükse ne yapılmalıdır? Elli ya da yüz kişilik bir takımlar. Takım boyutunun artması kağıt bazlı, yüz yüze stratejilerin başlatılmasına neden olmaktadır.

Mevzuata Uyum: Mevzuat uyumu ISO 9000 ya da FDA gibi standartlara tabi olabilir mi? Bu sorunlarda önce proje gereksinimleri sağlanmalıdır. Bu standartlar kuruluşunuzun dışından empoze edilebilir.

Alan Karmaşıklığı: Bazı proje ekipleri, kendilerini bir veri giriş uygulaması geliştirme veya bilgilendirici bir web sitesi gibi çok basit bir probleme yönlendirirler. Bazen problem alanı bir biyo-kimyasal süreç izleme veya hava trafik kontrolü gibi daha karmaşık olabilmektedir. Daha karmaşık alanlarda, araştırma ve deneyimlere daha fazla önem vermek gereklidir.

Organizasyon Dağıtım: Bazen bir proje ekibi farklı bölümlerin, farklı şirketlerin ortağı ya da dış hizmet firmaları gibi üyelere oluşabilmektedir. Organizasyon uyum eksikliği proje riskini büyük oranda arttırabilmektedir.

Teknik Karmaşıklık: Bazı uygulamalar diğerlerine göre daha farklıdır. Sıfırdan başlayarak yeni bir sistemin oluşturulmasında yüksek düzeyde kaliteye ulaşma oldukça basit olabilmektedir. Ama bu basitlik tam olarak kolay değildir. Mevcut eski sistemler ve veri kaynakları ile çalışıyorsanız hatasız olma ihtimali düşüktür. Doğru bir sistemi tek bir platform kullanarak inşa etmek kolay değildir. Bir sistem çeşitli platformlarda çalışmaya yapılandırılmış ve birçok farklı teknolojiler kullanarak inşa edilmiş olabilmektedir.

Organizasyon Karmaşıklığı: Mevcut organizasyon yapısı ve kültürü, geleneksel değerleri yansıtabilir. Kurumsal politika ile başa çıkmak, gelişmeye devam etmede farklı vizyonları olan insanlarla çalışma ve geleneksel değerlerden çevik geçiş aşamaları gibi durumlar çevik takımların karşılaştığı organizasyon karmaşıklıklarından bazılarıdır.

Kurumsal Disiplin: En büyük organizasyonlar maliyeti düşürmek, pazarlama süresini azaltmak ve uyum ilerlemesi gibi ortak altyapı platformları için kaldıraç isterler. Başarmak için etkili bir kurumsal mimari, kurumsal iş modelleme, yeni strateji ve portföy disiplinleri gerekmektedir. [73]

IV.4.1. Çevik ve Takım Boyutu

Şekil IV.11’de çevik takım boyutu ile ilgili 2009 Çevik uygulamaları anketinden özet sonuçlar gösterilmektedir. Anket sonuçlarına göre çevik takımların çoğunluğu yirmi veya daha az kişilerden oluşmaktadır. Azda olsa bazı kuruluşların çok büyük çevik takımları olduğunu görmekteyiz. [74]

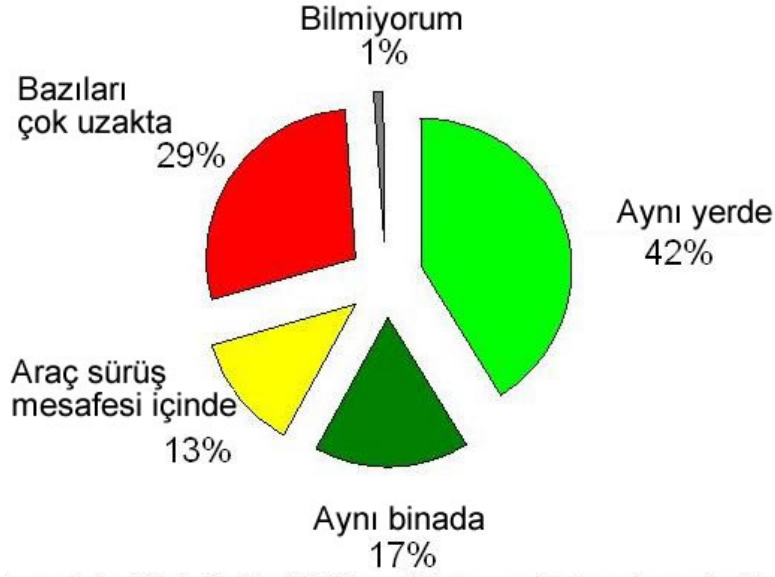


Şekil IV.11 Çevik Takımların Ortalama Boyutu

IV.4.2. Çevik ve Coğrafi Dağılım

Şekil IV.12’de çevik coğrafi dağılım ile ilgili 2009 Çevik uygulamaları anketinden özet sonuçlar gösterilmektedir. Anket sonucuna göre çevik takımların önemli bir bölümü aynı yerde bulunmaktadır.

Çevik geliştirme takımınızda BT insanları nasıl dağıtılmıştır?



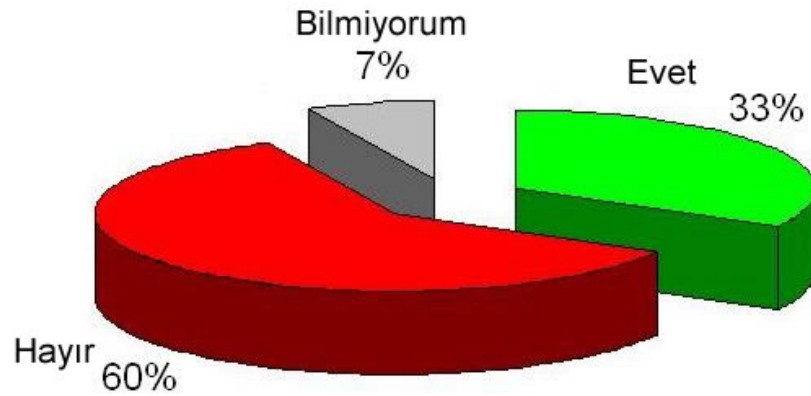
Source: Ambysoft Agile Practices 2009 Survey, <http://www.ambysoft.com/surveys/practices2009.html>
Copyright 2009 Ambysoft Inc.

Şekil IV.12 Çevik Yazılım Geliştirme ve Coğrafi Dağılım

IV.4.3. Çevik ve Mevzuat Uyumluluk

Şekil IV.13'de çevik mevzuat uyumluluk ile ilgili 2009 Çevik uygulamaları anketinden özet sonuçlar gösterilmektedir. Endüstri düzenlemelerinde genellikle dokümantasyon ve izlenebilirlik düzeylerinin fazla olmasını gerektirmektedir.

Çevik geliştirme takımınız endüstri ile ilgili düzenlemelere sahip mi?



Source: Ambysoft Agile Practices 2009 Survey, <http://www.ambysoft.com/surveys/practices2009.html>
Copyright 2009 Ambysoft Inc.

Şekil IV.13 Çevik Yazılım Geliştirme ve Mevzuat Uyumluluk

BÖLÜM V

EKSTREM PROGRAMLAMA

V.1. EKSTREM PROGRAMLAMA TANIMI

En popüler çevik süreçlerden olan ekstrem programlama, basitlik, iletişim, geri bildirim, cesaret ve alçak gönüllük değerleri üzerine kurulmuş bir yazılım geliştirme disiplini. Bu disiplini oluşturan öğelerden biri de takım çalışmasıdır. Temel işlerde takım üyeleri görevlerini yerine getirmektedir. Takıma geri bildirim iletilmesiyle, takım projede nerede olduğunu tespit etmektedir. [12]

Çevik süreçlerden, ekstrem programlama ile oluşturulan projede müşteri ve gereksinimleri ayrı bir önem kazanmaktadır. Geliştirme sırasında ekstrem programlama ile net olmayan, değişen müşteri gereksinimlerine hızla cevap verilmektedir. Bu durum geleneksel yazılım metodlarında olmayan bir durumdur. Çünkü proje öncesi müşteri gereksinimleri en son detayına kadar kâğıda dökülmüştür. Bu dokümantasyon temel alınarak, yazılım gerçekleştirilebilir. [57]

V.1.1. Ekstrem Programlama Değerleri

Ekstrem programlama değerleri basitlik, iletişim, geri bildirim, cesaret ve alçak gönüllülük değerlerinden oluşmaktadır. Çevik süreç, yazılım geliştirme açısından uyulması gereken kuralları ortaya koyarak desteklemektedir. [50]

Basitlik (Simplicity)

Yazılım kitaplarının çoğu basitlikten söz etmektedir. Kitap içeriğine bakıldığında, yazılım geliştirmenin karmaşıklaştığı görülmektedir. Genel olarak yapılan yanlışlar şu şekilde sıralanabilir.

Erken uygulanan karmaşık yapılar: Karmaşık yapıların ihtiyaç dışında modellenmesi, yazılım geliştiricilerin fazla mesai yapmasına neden olmaktadır. Bu yapıların, parçalanarak önceliklere göre küçük bölümler halinde modellenmesi gereklidir. Bu durum önem taşıyan bölümlerle ve en az hata ile müşteriye ilk

sürümün teslim edilmesini sağlamaktadır. İlk sürümden sonra müşteri gereksinimleri daha iyi tespit edilerek karmaşıklığın önlenmesi sağlanır.

Proje içinde fazladan modelleme ve kodlama yapmak: Yazılım geliştiriciler, sonradan olacak değişikliklere yanıt verebilmek veya her isteğe yanıt verebilecek en iyi yazılımı yapmak için daha fazla modelleme ve kodlama yapma eğiliminde olabilirler. Hedef, müşteri gereksinimlerini tespit ederek, en basit, en verimli ve en ucuz çözümü sunmak olmalıdır. Gelecekteki problemler, gelecekte çözülmemelidir. Bugünden en basit çözüm üzerinde çalışılarak, gelecekte yeni bir fonksiyon eklendiğinde sistemin daha basit olması sağlanabilmektedir.

Karmaşık altyapılar geliştirmek: Proje ekiplerinin ilk hatası gelecekte kullanılmak üzere yaptıkları modüller, sınıf kütüphaneleri ve iskelet yapılarıdır. Gerektiğinde kullanım için hazır olmasını sağlamak amacıyla yapılırlar. Fakat bu durum önemli zararlara neden olabilmektedir. İlk olarak görülebilir bir sürüm teslim etmeden müşteri kaynakları kullanılmış olmaktadır. Müşteri, temel gereksinimlerini yapabileceği bir sistem isterken, geliştiriciler gelecek için hazırlık yapmış olmaktadır.

İletişim (Communication)

Projeyi oluşturan tüm kişiler arasında çok iyi bir iletişim oluşturulmalıdır. Proje başarısı için bu değer çok önemlidir. İletişim, kişiler arası belli işaret, hareket veya sembollerle bilgi alışverişi yapılan genel sistemin adıdır. İletişim iki yönlüdür. Her iki tarafta bilgi sunar ve kazanır. İletişimde sorunlar ortaya çıktığında problemler de ortaya çıkmaktadır. Örneğin, bir yazılım geliştirici sorumlu olduğu alanın tam olarak bitmediğini iş arkadaşlarına söylememesi başka bir yazılım uzmanının bu problemi ortaya çıkartmak için ekstra zaman harcamasına neden olabilmektedir.

Geliştiriciler yazacakları sistemin modelini (prototip) müşteriye sunarlar. Müşteriye teslim edilen yazılımın model olduğu bildirilmez. Modelleme işlemi, aslında iletişimi arttırmak ve geliştirmek için kullanılır. Bir konuda oluşturulan bir model, o konu hakkında okumaktan veya tartışmaktan daha fazla katkı sağlar. Modelleme, fikirlerin daha rahat anlaşılmasına, diğer kişilerin fikirlerinin daha rahat anlaşılmasına ve genel olarak tüm iş hakkında fikir sahip olmayı sağlamaktadır.

Geri Bildirim (Feedback)

Geliştirilen sistemin doğruluğunu anlamının yolu farklı kişilerin geliştirilen sistem hakkında test yaparak sonuçları bildirmesidir. Test sonuçlarını doğru zamanda alarak kısa sürede çözümleri üretmek önemlidir. Modellemeyi takım halinde yapmak gerekmektedir. Yazılım geliştirme işi bireysel bir çalışma değildir. Tek başına yapılması sorunlara yol açabilmektedir. Beraber çalışılan diğer kişiler ile sonuçlara daha kısa zamanda ulaşma sağlanabilir. Böylece zaman kaybı önlenmiş olmaktadır.

Modeller müşteri ile beraber, ayrıntılı tasarım modelleri ise programlamayı yapacak kişiler ile yapılmalıdır. Proje süresince aylık veya haftalık toplantılar düzenlenmelidir. İhtiyaç halinde takvim dışına çıkarak kısa süren toplantılar yapılması geliştirme sürecine katkı sağlamaktadır. Modelin uygulanmasında toplantı düzenleyemiyorsa, modeli kodlayarak ilk sürümden sonraki sonuçlar beklenmelidir. İlk sürümün müşteri gereksinimlerini yansıtması gerekmektedir. Müşterinin kabul testleri ile gereksinimleri değerlendirmesinden sonra geri bildirimlerle model test edilmiş olmaktadır.

Bir takım halinde çalışıldığında, geri bildirimler saniyeler ya da dakikalar içinde olabilmektedir. Gayri resmi toplantılarda ise geri bildirim dakikalar ya da saatler alabilmektedir. Uygulama yapılıp ilk sürüm verildiğinde geri bildirim saatler ya da günler içinde olmaktadır. Kabul testlerinden sonra geri bildirim birkaç hafta ya da ay içerisinde gelmektedir. Takım halinde çalışmanın en büyük yararı geri bildirimlerin hızlı olmasıdır.

Cesaret (Courage)

Genel durumu kabul etmek ve bir şeyleri geliştirmeyi, düzeltmeyi denememek ya da birisinin çıkıp hataları düzeltmesini beklemek kolay bir durumdur. Yazılım endüstrisinin başarısızlıklarında cesaretsizliğin büyük payı vardır. Çevik metodolojisi diğer insanlarla beraber çalışmayı, onlara güvenmeyi ve kendimize güvenmemizi öğütlemektedir. Bu durum cesareti arttırmaktadır. Ekstrem programlama, yapabileceğiniz en basit modeli yapmanızı söylemektedir. Ekstrem programlama, gerçekten dokümantasyona ihtiyacınız olduğunda doküman oluşturulmasını önermektedir. Modellemede, beyaz tahta ya da not defteri gibi en basit araçlar kullanılabilir.

En yüksek verimin elde edilmesi istendiğinde karmaşık yazılım araçlarının kullanılması gerekmektedir. Cesaretli ekipler, denemekten ve yanılmaktan korkmayarak sonuçlara daha hızlı ulaşıp daha çok yol alabilirler.

Alçak Gönüllülük (*Humility*)

Yazılım geliştirici her şeyi bilmediğini kabul edecek kadar alçak gönüllü olmalıdır. Gelmiş geçmiş en iyi yazılımcı olunabilir fakat her detayın biliniyor olması mümkün değildir. Programlama yapan kişi, proje ekibindeki herkesin bir uzmanlık alanı olduğunu bilmeli ve diğer kişilerin yardımı ile kendi işlerinin başarılı bir biçimde biteceğini görmelidir.

Alçak gönüllülük, diğer kişiler ile birlikte çalışmayı imkan dahilinde kılmaktadır. Ekip üyesi olan kişi, diğer proje elemanlarının farklı deneyimlerinin olduğunu, kişisel pek çok farklılıklar olduğunu bilerek saygı ile hareket etmelidir. [50]

V.1.2. Ekstrem Programlama İlkeleri

Ekstrem programlama değerleri, tek başına soyut kaldığından proje davranışları için doğrudan rehber olarak kullanılamaz. Ekstrem programlama değerleri temel alınarak birtakım ilkeler oluşturulmuştur.

Hızlı Geri Bildirim (*Rapid Feedback*)

Sık ve hızlı geri bildirim elde etmek, projenin sürecini olumlu yönde etkilemektedir. Geri bildirim ile yanlış anlaşılmalara ve hatalara ortadan kaldırılmaktadır.

Basitliği Tercih Etmek (*Assume Simplicity*)

Basit çözümler, kısa zamanda kolaylıkla uygulanarak oluşturulabilir. Bu durum, geri bildirimin hızlı bir şekilde gerçekleşmesini sağlamaktadır. Basit çözümlerin kavranması ve anlatılması daha kolay olmaktadır. Ekstrem programlama, programcılardan o anki gereksinimi karşılamak için basit çözümü beklemektedir. Programcı, gelecekte oluşabilecek eklemeleri ve değişiklikleri düşünmeyerek, sadece kendisinden beklenen en basit haliyle uygulamalıdır.

Artımlı Değişiklik (*Incremental Change*)

Basit çözümler uygulandığında bile yazılım sistemleri zaman içinde karmaşık bir yapıya dönüşebilmektedir. Yapılan bir değişiklik, sistemin düşünülmeyen bir bölümü üzerinde hata oluşmasına sebep olabilmektedir. Oluşabilecek bu hataları kontrol altında tutabilmek için değişikliklerin küçük çapta olması gerekmektedir. Büyük değişiklikler beraberinde büyük sorunları getirebilir.

Değişimi İstemek (*Embracing Change*)

İlerleyebilmek için bir yön tayin etmek ve yeniliklere açık olunmalıdır. Yeniliklere açık olmak cesaret gerektirir. Bilinmeyenle uğraşmak, rahatsız edici olabilirken, başarıyı elde edebilmek için değişimi istemek gerekmektedir.

Kaliteli İş (*Quality Work*)

Ekstrem programlama projelerinde kaliteli işin ortaya konabileceği bir ortamın oluşturulması gerekmektedir. Hiçbir programcı hatalı program yazmak istemez. Çalışma ortamının etkisiyle yüksek kalitede yazılım yapmak programcının özgüvenini artırmaktadır. Bu durum müşteriye memnun edici ürünlerin ortaya konmasını sağlamaktadır.

Öğrenmeyi Öğret (*Teach Learning*)

Tecrübeli programcılar bilgilerini daha az tecrübeli programcılarla paylaşarak takım arkadaşlarıyla aynı teknik seviyeye gelebilir. Programcılara komutlar vererek iş yaptırmak yerine, kendiliğinden bazı şeyleri öğrenerek, görevlerini yerine getirmeleri sağlanmalıdır.

Az Başlangıç Yatırımı (*Small Initial Investment*)

Ekstrem programlama en modern ve pahalı araç gereçlerle projeye başlanmasını beklemes. Başlangıç giderleri ne kadar düşük tutulabilirse, projenin iptali durumunda kayıplar o oranda az olacaktır.

Kazanmak İçin Oyna (*Play to win*)

Ekstrem programlama takımları kazanmak için oynamalıdır. Hedef, programı tamamlamak ve müşteriye teslim etmek olmalıdır.

Somut Denemeler (Concrete Experiments)

Verilen kararların sonuçlarını kontrol edebilmek için denemeler yapılmalıdır. Çünkü alınan kararlar her zaman doğru olmayabilir. Bazı somut denemeler yardımıyla yazılım sistemlerinin hangi durumda olduğu tespit edilebilir. Örneğin testler hazırlayarak, oluşturulan mimari ve tasarım kontrol edilebilir.

Açık ve Samimi İletişim (Open, Honest Communication)

Projenin başarılı olabilmesi için bireyler arasında açık ve samimi bir iletişimin olması gerekmektedir. Projelerde çoğu zaman bireylerin korkuları, deneyimsiz olmaları ya da kendilerini çok beğenmeleri açık ve samimi bir iletişim ortamının oluşmasını engellemektedir.

Takımın İçgüdülerini Kullan, Onlara Karşı Koyma (Work with people's instincts, not against them)

Bireysel içgüdü yanında, bireylerin oluşturduğu takımların da içgüdüsi bulunmaktadır. Takımın içgüdüüne kulak verilmelidir. Bunun göz ardı edilmesi, proje için olumsuz sonuçlar doğurabilmektedir.

Sorumluluk Üstlenmek (Accepted Responsibility)

Sorumluluk birilerine verilmeyerek bireyler kendileri sorumluluk üstlenmelidir. 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 hızlandırmaktadır. 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)

Amaç ekstrem programlamayı tamamen uygulamak değildir. Amaç kısa bir zamanda projeyi başarılı bir sonuca ulaştırmaktır.

Az Yükle Yolculuk Yapmak (Travel Light)

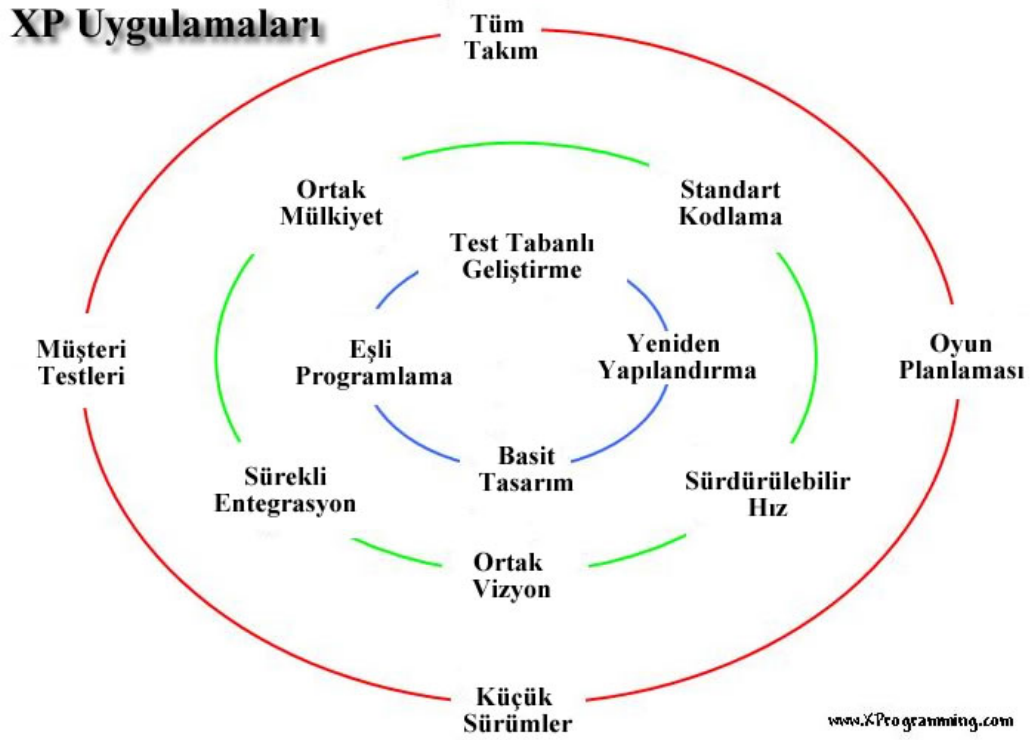
Projede hızlı ilerleyebilmek için fazla bir yükle yola çıkılmaması gerekmektedir. Beraber çalışmayı kolaylaştırmak için kullanımı kolay araç ve gereçler seçilmelidir. Formalitelerden uzak durulmalıdır.

Doğru Ölçüm (Honest Measurement)

Proje sürecini kontrol edebilmek için değişik türde ölçümlerin yapılması gerekmektedir. Örneğin, hazırlanan birim testleri ile sınıfların işlevleri kontrol edilebilir. Yapılan ölçümler doğru yapıldığı takdirde kontrol mekanizması olarak kullanılan ölçümlerin bir anlamı olacaktır. Programcılar tarafından doğru yapılmayan ölçümler projenin sürecini olumsuz yönde etkileyecektir. [57]

V.1.3. Ekstrem Programlama Uygulamaları

Ekstrem programlamada yer alan beş değerden onüç temel uygulama ortaya çıkarılmıştır.



Şekil V.1 Ekstrem Programlama Uygulamaları [12]

Tüm Takım (Whole Team)

Ekstrem programlama projelerine katkısı olan herkes takımın birer üyesi olarak birlikte çalışırlar. Bu takımda proje içinde olan iş temsilcisi yer almaktadır. Bu iş temsilcisi, müşteri gereksinimleri ortaya koyarak, öncelikleri tanımlar ve projeyi yönetir. Müşteri, projeye ne kadar hakim olursa süreç o kadar iyi işlemektedir. Takım içindeki testçiler, müşteri ile yapılması gereken testleri belirlemektedir. Takımda üyelerinden olan koç, projenin takibini yaparak, işlemlerin kontrolünü sağlamaktadır.

Yönetici ise kaynakları temin ederek, bağlantı kurulması gerekli işleri yaparak çalışmaların koordinasyonunu sağlamaktadır. Ekstrem programlama takımında yer alan kişiler, yetenekleri ölçüsünde katkıda bulunmaktadır. Takım anlayışı olarak uzmanların yer aldığı takım yerine sadece özel yetenekleri olan kişiler vardır.

Oyun Planlaması (*Planning Game*)

Ekstrem programlamada planlama, yazılım geliştirmede önemli olan iki faktör üzerinde ilerlemektedir. Bu faktörler, nelerin hangi tarihle bitirileceğini tahmin etmek ve sonrasında hangi işlerin yapılması gerektiğine karar vermektir. Planlamada önemli olan projenin plana bağlı olarak yürütülmesidir. Planlamadaki her adım gerçeğe aktarılamayabilir. Ekstrem programlamada planlama yapılırken bu faktörlere dikkat edilerek plan oluşturulmalıdır.

Plan oluşturma, müşterinin gereksinimlerini geliştiriciye sunduğu ve geliştiricilerinde bunların zorluk derecelerini belirlediği bir etkinliktir. Müşteri proje planını, maliyet tahminleri ve geliştirilecek programın özelliklerinin önem derecelerine göre şekillendirmektedir. Başlangıçta elde edilen planlar yeterli olmayabilir. Çünkü projede gerçek anlamda başlayıncaya kadar işlerin öncelikleri ve maliyetler tam olarak belli değildir. Buna rağmen ilk plan, bazı kararları vermek için yeterli olabilmektedir. Sonrasında plan, belirli sürelerde güncelleştirilmelidir.

İteratif planlama, takımın her iki haftada yeni bir çalışma yapması üzerine kurulu planlamadır. Ekstrem programlama takımları iki haftalık sürelerde programı derleme işlemini yapmaktadır. Böylece iki haftada bir yeni özellikler eklenmiş gereksinimleri karşılayan programlar elde edilir. İteratif planlamada, müşteri sonraki iki haftalık sürede neler yapılması gerektiğini belirtir. Belirtilen bu gereksinimlere göre geliştiriciler görevleri ve alt görevleri oluşturur. Maliyetlerle ilgili tahminler yapılarak bir önceki iterasyonun ne kadar başarılı olduğu, takımda hangi işlerin sonraki yenilemede yapılması gerektiğine karar verilir. Bu planlar basit olmalarına rağmen müşterilerin sağladıkları bilgiler yardımıyla projenin iyi bir şekilde yönetilmesini sağlarlar. İki haftada yapılan işler, herkes tarafından görülerek anlaşılabilir duruma gelmektedir. Ekstrem programlamada, işin tamamlanması için yüzde yüzünün bitirilmesi gereklidir. İstenen iş yapılmış ya da yapılmamıştır.

Müşteri Testleri (Customer Tests)

Müşteri tüm gereksinimlerine ait özellikleri belirterek, takım testlerini ortaya koymaktadır. Bu testler, kabul testleri olarak adlandırılmaktadır. Programda olması gereken özellikler, müşteri tarafından bildirilerek test edilmektedir. Bu testler, geliştiricilerin ve müşterilerin onayından geçmektedir. Proje içinde zaman faktörü önemlidir. Bu nedenle tek tek test yapmak yerine işleri otomatikleştirerek test yapmak daha hızlı sonuçlar vermektedir. En iyi ekstrem programlama takımları testleri kendileri hazırladıkları gibi onların çalıştırılmasını da kendileri gerçekleştirmektedir. Takım, test çalışmaya başlayınca sonuç alınıncaya kadar testin doğru çalıştırılmasından sorumludur.

Küçük Sürümler (Small Releases)

Ekstrem programlama takımları, iki yöntemle küçük sürümler üretmektedir. Birincisi, takım her iterasyonda müşterinin seçtiği değer ve özelliklere göre test edilen çalışan sürümler üretmektedir. Müşteri, üretilen deneme sürümünü herhangi bir amaç için kullanabilir. Önemli olan, yazılımın her iterasyonda sonundaki sürümlerinde müşteri tarafından kolay anlaşılabilir olmasıdır. Müşteri, herhangi bir sürüm ile ilgili sorunu olmamalıdır. Bu sayede her şeyin açık ve problemlerin çözülmesi sağlanmaktadır.

İkinci yöntem, ekstrem programlama takımları son kullanıcılara mümkün olduğu kadar sık sık sürümler oluşturur. Ekstrem programlama hafif projelerde günlük olarak sürüm üretmektedir. Kapsamlı projelerde, aylık ya da kısa aralıklarla sürümler üretilmektedir. Projenin önemi ve büyüklüğü arttıkça sürüm üretme sıklığı orantılı olarak artmaktadır.

Ekstrem programlama takımlarında herkes aynı zamanda sürüm üretiminde çalışmaktadır. Ekstrem programlama takımında yer alan kişiler, bir işi sadece kendisinin yapmadığının farkındadır. Sürümlerin üretiminde bu bilince sahiptirler. Üretilen sürümler, her zaman müşteri testlerinde geçtiği için ekstrem programlama takımı sürümleri güvenilir kabul etmektedir.

Devamlı Entegrasyon (Continuous Integration)

Ekstrem programlama takımları, projenin daima bir bütün olarak kalmasını sağlarlar. Bu nedenle bazen günde bir kez yapılan derlemelerde yetersiz kalabilirler. Gün içinde birden fazla derleme yapmak gerekebilir. 40 kişilik bir ekstrem

programlama takımı günde en az 8 ya da 10 kez derleme yapabilmektedir. Çalışmayan proje geliřtirmelerinde ise haftalarca sistem entegre edilmediğinden problemler ortaya çıkmaktadır. Sonradan gerçekteřtirilen derleme ve sürümlerle proje başarı olmayabilir. Ekstrem programlamada devamlı entegrasyon ile bu tür sorunlar engellenmektedir.

Proje üzerinde sık yapılmayan entegrasyon çalışmaları çeřitli problemlere neden olabilmektedir. Birinci problem olarak, kritik önemi olan bir çalışmada entegrasyon son ana kadar yapılmayabilir. Takım tecrübesi yetersiz olabilir. Bu işin son anda hatasız yapılabilme ihtimali düşüktür. İkinci problem ise seyrek aralıklarla entegre edilen kodlar daha fazla hataya sahiptir. Problemler, tasarım zamanında entegre olamayan sistemin test edilmemesinden bulunamayarak gizlice sistem içine yerleşirler. Üçüncü problem ise yetersiz entegrasyon süreci kodların değışmez olmasına neden olmaktadır. Kod değışmezliğı řu şekilde açıklanabilir. Uzun süre çalışılarak bitirilen program özelliklerinin sisteme entegre edilmesinde diğerk bölümlere uyum sağlanması gereklidir. Bu durum zaman tüketimi arttırmaktadır.

Ortak Mülkiyet (Collective Ownership)

Ekstrem programlama projesinde yer alan takım geliřtiricileri herhangi bir takım üyesinin yazdığı kodu geliřtirebilir. Bu durum yazılan her kodlamanın takım üyelerince kontrol edilmesini sağlamaktadır. Sonuç olarak daha kaliteli ve az hatalı kodlar üretilmektedir.

Her bir kodlamanın sorumlusu bir geliřtiricisi olduğunu düşünürsek bu durumda geliřtiricilerden birisi kod yazarken diğeri geliřtirici sorumluluğı dışında olan kodlamaya karışmayarak yapılmasını bekleyebilir. Bu bekleme süresinde hiçbir şey yapılamayabilir. Ekstrem programlama geliřtiricilere beklememeyi, gereksinim duyulan bir kodlama hazır değılse, sorumluluk dışında bile olsa kod yazma yetkisini geliřtiriciye vermektedir. Bu tür kod yazımında en önemli noktalardan biride anlaşılmayan kodlar üzerinde değışlik yapılmamasıdır. Ekstrem programlama, bu tür problemlerin çıkmaması için iki yöntem üzerinde odaklanmaktadır. Bunlar programcı testleri ve eşli programlamadır. Bu yöntemlerle hatalar yakalanarak kodların düzeltilmesi sağlanır.

Kodlama Standardı (Coding Standart)

Ekstrem programlama takımları genel bir kodlama standardı kullanmaktadır. Bu sayede projedeki tüm kodlar tek tip olmaktadır. Standardın detayları önemli olmayıp önemli olan kodlamada bir bütünlüğün sağlanmasıdır.

Ortak Vizyon (Metaphor)

Ekstrem programlama takımları, programın geliştirilmesinde ortak vizyona sahiptir. Ortak vizyon, programın nasıl geliştirildiği ile ilgili tarif olmaktadır. Tarif olarak yapılırsa arı kovana sisteminde arının polen toplamaya gitmesi ve topladığı polenleri kovana getirmesi gibi bir örnek verilebilir. Bazen yeterli ortak vizyon ortaya konulamayabilir.

Sürdürülebilir Hız (Sustainable Pace)

Ekstrem programlama takımları uzun dönemlerle çalışmayı düşünürler. Bu dönem içerisinde hızlı ve sıkı çalışırlar. Takımlar etkili ve verimli olduklarında normalden daha fazla çalışarak diğer zamanlardaki yavaşlıklarını giderirler. Başarısız projelerin en önemli eksiklikleri yazılım kalitesinin yetersiz ve üretim sürecinin uzun olmasıdır.

Basit Tasarım (Simple Design)

Ekstrem programlama takımları, basit tasarımlar ile projeleri gerçekleştirirler. Projede, programcı testleri ve tasarım geliştirme aşamalarından sonrada basit tasarım korunmaktadır. Takım ihtiyaç duyulan özelliklere göre tasarımı şekillendirmektedir. Yazılım her zaman bir sonraki yapılandırma ya da sürüm değişimi için hazır bulunmaktadır. Ekstrem programlamada tasarım bir defa yapılmayarak geliştirme süresince tasarım şekillendirilir. Sürüm planlama ve iterasyon planında tasarım adımları tespit edilmektedir Takım, hızlı tasarım çalışmaları yaparak gözden geçirme toplantıları düzenlemektedir. Ekstrem programlama gibi artımlı ve iteratif yazılım geliştirme metodolojilerinde tasarım iyi olmalıdır. Bu nedenle EP yazılım geliştirme sürecinde tasarım yapılır.

Eşli Programlama (Pair Programming)

Ekstrem programlamada proje süresince tüm yazılım eşli gruplar ile yapılmaktadır. Böylece yapılan her kodlama, kodu yazan kişi dışında başka birisi

tarafından kontrol edilmektedir. Sonuç olarak tasarım, test ve kodlama açısından kalite artmaktadır. Bir programcı yerine iki programcının aynı işi yapması verimsiz bulunabilir. Tam tersine akademik çalışmalara göre bir kodun iki kişi tarafından yazılmasının, tek kişi tarafından yazılmasından daha iyi sonuçlar çıkardığı tespit edilmiştir. Eşli programlamayı uygulayan geliştiricilerin yüzde doksanı bu şekilde çalışmaya devam etmektedir. Eşli programlama, kaliteli kod yazımı sağladığı gibi takım içi iletişimi de geliştirmektedir. Eşler arası değişim geliştiriciler arası bilgi birikiminin paylaşılmasını sağlamaktadır. Yeni bilgiler öğrenen geliştiriciler kendilerini geliştirerek takım içinde değerli bir üye olmaktadır. Eşli programlama, herkes için faydası olan bir çalışmadır.

Test Tabanlı Geliştirme (Test Driven Development)

Ekstrem programlanın en önemli özelliklerinden biride geri bildirimin iyi sağlanmasıdır. Geri bildirim iyi olması yapılan iyi testlerin sonuçlarına bağlı olmaktadır. Ekstrem programlama takımları test tabanlı geliştirme yaklaşımı tercih ederler. Bu yaklaşımda yeni testler, kısa çalışmaların ardından hazırlanarak çalıştırılmaktadır. Kodlamanın tamamına yakını kolaylıkla test edilmektedir. Testleri yazmak yeterli olmayıp onları çalıştırmak da gerekmektedir. Geliştirici testleri ve birim testleri bir araya getirilerek, bir sonraki sürümde her bir testin çalıştırılması sağlanmaktadır. Testlerin tamamı gerçekleştirilmektedir. Böylece geliştiricilerin yapılan işle ilgili tüm geri bildirimleri anında alınması sağlanmaktadır. Testlerin geliştirmesi tasarım gelişimine katkı sağlamamaktadır.

Yeniden Yapılandırma (Refactoring)

Ekstrem programlama ile yazılım geliştirmede her iterasyon önemli bir değere sahiptir. Her iterasyonda, yeni özelliklerin eklenmesi, kod kalitesi ve benzeri değerler üretilmektedir. Bunun sağlanması tasarımın iyi olmasına bağlıdır. Ekstrem programlamada devamlı olarak tasarımın iyileştirilmesi için yeniden yapılandırma kullanılmaktadır. Yeniden yapılandırma yaklaşımı, kod tekrarlarını ortadan kaldırmayı, program parçaları içinde bağılılığı güçlendirmeyi amaçlamaktadır. Sonuç olarak, ekstrem programlama takımları iyi ve basit bir tasarımla işe başlarlar. Takım, sürekli olarak tasarımın yazılım için basit ve iyi olması sağlamaktadır. Böylece proje çalışma temposu korunmaktadır.

Yeniden yapılandırma, proje ilerledikçe ve tasarım değıştikçe sonradan problem çıkmaması için güçlü testlerle desteklenmektedir. Müşteri ve geliştirici testleri aşılması gereken önemli testlerdir. Ekstrem programlama çalışmaları hepsini destekleyerek müşteri ve geliştirici testlerinin birlikte değerlendirilmesi sağlamaktadır. [50]

V.1.4. Ekstrem Programlama Takım Roller

Çevik yazılım projelerinde takım üyelerini tanımlamak için roller belirlenir. Her rol, bazı sorumluluk ve haklar sağlamaktadır. Roller, sabit olarak kalmayıp değışebilmektedir. Takım içinde farklı kişilere roller verilerek daha sonra rol değışikliği yapılabilmektedir. Proje gereksinimleri ile yeni roller tanımlanabilir.

Müşteri (Customer)

Müşteri, projenin var olma sebebidir. Müşteri, gereksinimlerine yanıt verebilecek bir yazılım sistemi için yatırım yapan kişidir. Proje süresince, programlamada nelerin yapılması gerektiğini bildirmektedir. Müşteri, yapılması gerekenleri kullanıcı hikayeleri ile ifade etmektedir. Geliştiriciler, müşteriye proje süresince yardım ederler. Müşteri, kullanıcı hikayelerinin oluşturulmasından sorumludur. Her kullanıcı hikayesi yazılım sisteminde bir özelliğı tanımlamaktadır. Geliştiricilerin, uygulamayı gerçekleştirebilmesi için kullanıcı hikayelerini anlamaları gereklidir. Bu durumlarda uygulama için tahminlerde bulunabilirler. Ayrıca kullanıcı hikayelerinin test edilebilir durumda olması gerekmektedir.

Müşteri, çalışma alanı hakkında bilgi sahip olan kişidir. Geliştiriciler, sorunlar karşısında müşteriye soru sorabilirler. Bu sorulara en iyi cevabı müşteri verebilir. Hangi kullanıcı hikayelerinin uygulamasının yapılacağını müşteri karar vermektedir. Burada müşteriye bir sınırlama getirilmemektedir. Müşteri seçim yaparak geliştiricilerin uygulama yapmasını sağlamaktadır. Uygulamaya geçirilen her kullanıcı hikayelerini kontrol etmek için müşteri kabul testleri tanımlaması yapmaktadır. Bu testler, geliştirici ya da testçi tarafından uygulamaya konulmalıdır. Kabul testleri, kullanıcı hikayelerinin doğru uygulanıp uygulanmadığını kontrol eden bir mekanizmadır.

Programcı

Ekstrem programlama takımında programcıların, hikaye tahminleri, özellikleri uygulama, testleri yazma, geliştirme sürecini otomatikleştirme ve yavaş yavaş sistem tasarımını geliştirme gibi görevleri bulunmaktadır. Programcılar birbirleri ile teknik işbirliği yaparak ve kodlamaları eşleştirerek takım içi iletişimin gelişmesini sağlarlar. Müşteri tarafından belirlenen kullanıcı hikayelerinin uygulama süreleri programcılar tarafından tahmin edilmektedir. Geleneksel projelerde bu tahminler teknik bilgisi yetersiz olan yöneticiler tarafından yapılmaktadır. Bu yüzden tahminler gerçekçi olmamaktadır.

Her programcı, test tabanlı geliştirme için bir takım arkadaşıyla birlikte çalışmaktadır. Eşli programlama, programcılar arası iletişimi kuvvetlendirmektedir. Test tabanlı geliştirmede bakım ve geliştirme işlemleri kodun sade olmasını sağlamaktadır. Ekstrem programlama projelerinde, programcılar kodlama yapmadan önce gerekli test sınıfları oluşturarak uygulamaya başlamaktadır. Programcılar, yaptıkları testler ile her gün bir veya daha fazla program parçalarının entegrasyonunu gerçekleştirirler. Sürekli entegrasyon, programcı tarafından oluşturulan parçaların entegre edilmesini sağlamaktadır. Her programcı bu süreçten geri bildirim sağlayarak, yaptıkları çalışmanın entegre olma durumunu ölçerler.

Proje Yöneticisi

Ekstrem programlama takımında proje yöneticisi müşteriler, tedarikçiler, organizasyonun geri kalanıyla olan iletişimi koordine etmek ve takım içi iletişimi sağlamaktan sorumludur. Proje yöneticisi, süreç durumu ile ilgili takıma hatırlatmalarda bulunur. Proje yöneticisinin, proje bilgilendirme sunumlarında takımın diğer üyelerine göre daha yaratıcı olması gereklidir.

Proje menajeri, müşteri ve programcılar bir araya getirerek beraber çalışabilecekleri ortamın oluşmasını sağlar. Ekstrem programlama proje menajeri tek başına proje planlamasından sorumlu değildir. Programcılar kendi başına seçim yapmasını sağlayarak sorumluluk almalarını kolaylaştırmaktadır. Toplantı ve diğer buluşmaları düzenleyerek takımın sorunlarına çözüm getirmek için gerekli işlemleri yapmaktadır.

Koç

Koç, çevik süreci tanıyarak ve nasıl uygulanması gerektiğini bilmektedir. Görevi, proje başlangıcında takım oluşturmak ve onlara bir süre rehberlik yapmaktır. Koç, projede sorun çıktığında veya takım ekstrem programlama yöntemlerine uymadığında müdahale etmektedir. Bazen uygulamalarda aktif olarak bulunmaktadır. Örneğin, birim testlerinin nasıl bir yapıda olacağını diğer programcılara gösterebilir.

Testçi

Ekstrem programlama takımında testçi, uygulama öncesinde sistem düzeyinde testler hazırlamak ve müşterinin oluşturduğu kabul testlerini uygulamakla görevlidir. Aynı zamanda birim ve entegrasyon testlerinin uygulanmasında takım arkadaşlarına yardım etmektedir. [57,59]

V.1.5. Haklar ve Sorumluluklar

Projede takım üyeleri bazen içgüdüsel bir korkuya kapılabilmektedir. Bunun en önemli sebebi belirsizliktir. Nelerin, nasıl yapılacağı bilinmediği zaman takım içinde huzursuzluk çıkmaktadır. Proje başarısı bu durumdan olumsuz etkilenebilmektedir.

Projenin başarılı olması için çalışanların korkusunun azaltılması ya da yok edilmesi gereklidir. Ekstrem programlama bu konuda proje çalışanlarına takım haklarının verilmesini sağlamaktadır. Hangi rollerin, hangi haklara sahip olduğu aşağıda açıklanmaktadır.

Müşteri Hakları

Müşterinin sahip olduğu haklar şu şekilde özetlenebilir.

- Müşteri, bütçe ve zaman planlaması için öncelikleri belirleme hakkına sahiptir.
- Müşteri, gereksinimlerde değişiklik yapma ve gereksinimlerin uygulamasını isteme hakkına sahiptir.
- Müşteri, programcılarının sağlayabileceği en yüksek verim ve değeri elde etme hakkına sahiptir.
- Müşteri, projedeki gelişmeleri çalışan yazılımlarla görme hakkına sahiptir. Bu durum kısa aralıklarla oluşturulan yeni sürüm ve kabul testlerine iyi yanıt veren uygulamalarla sağlanmaktadır.
- Müşteri, sonraki sürüm için uygulanacak kullanıcı hikayelerini seçme hakkına sahiptir.

- Bütçe durumunu gözeterek öncelikler ve içerikte değişiklikler yapma hakkı vardır.

Programcı Hakları

Programcının sahip olduğu haklar şu şekilde özetlenebilir:

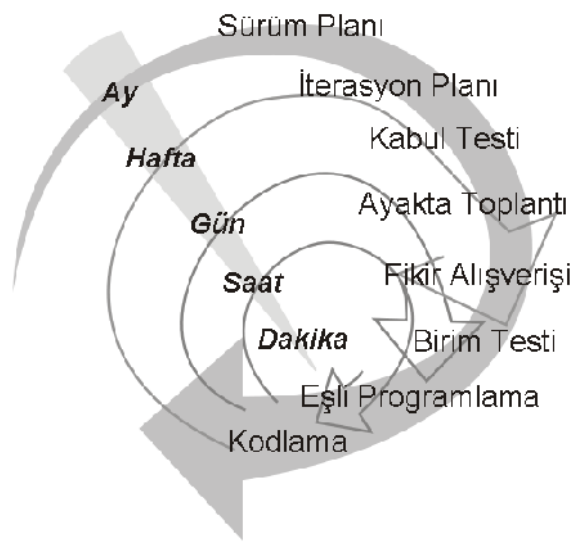
- Programcı, takım arkadaşlarına soru sorma ve cevap alma hakkına sahiptir.
- Programcı, kullanıcı hikayeleri için uygulama zaman tahmini yapma hakkına sahiptir.
- Programcı, tahminler üzerinde değişiklik yapma hakkında sahiptir. Programcı, kendisine görev verilmesi yerine sorumluluk alma hakkında sahiptir.
- Programcının her zaman yüksek kalitede ve teknik mükemmellikten ödün vermeden çalışmaya hakkı vardır.
- Programcının hangi öncelik sırasına göre neyi yapması gerektiğini bilme hakkı vardır.
- Projedeki gelişim süreci ile ilgili rapor verme hakkına sahiptir.
- Takımın kendi içinde organize olabilme, daha iyi çalışabileceği yolları seçme hakkı vardır.

Süreç İşleyişi

Ekstrem programlama projelerinde projenin ilerleyişi genel olarak aşağıdaki şekilde özetlenebilir.

- Müşteri, gereksinimlerini bildiren kullanıcı hikayelerini oluşturur. Bunlara öncelik sırası ataması yapar. Programcılar, kullanıcı hikayeleri için uygulama 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 kullanılabilir durumdadır.
- Müşteri, ilk iterasyon (1 hafta) için gerekli kullanıcı hikayelerini tahminlere göre seçme işlemini yapar.
- Programcılar, iterasyon için seçilen kullanıcı hikayelerini uygulamasını yaparlar. Soru sorulması gereken durumlarda müşteriye danışılarak cevap alınır.
- İterasyon sonunda programcılar müşteriye çalışır bir sistem sunarlar. Müşteri sistemi değerlendirerek programcılara geri bildirim sağlar.

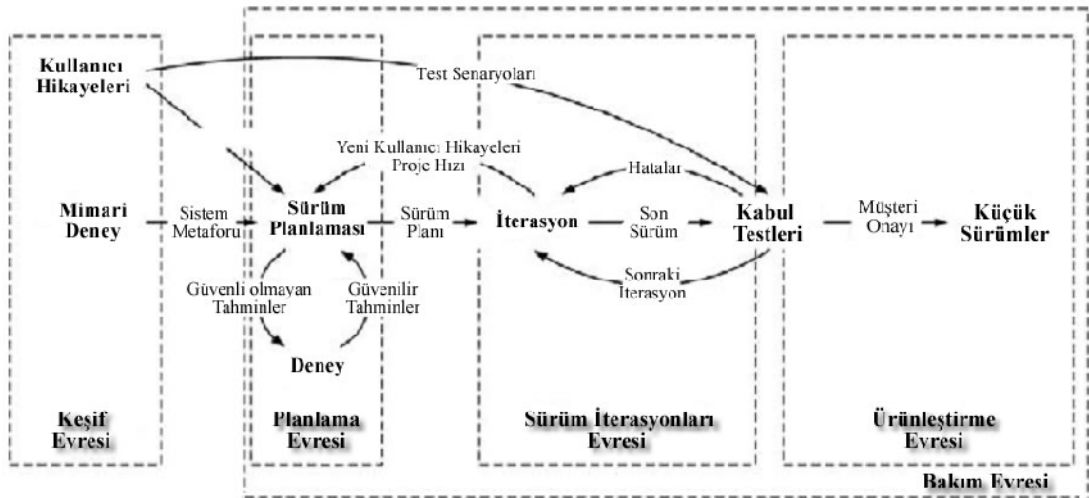
- Elde edilen deneyimlerle bir sonraki iterasyon planlanır. Müşteri, yeni gereksinimlerin uygulanması istediğinde, tekrar kullanıcı hikayeleri ve tahminler yapılır. Yeni kullanıcı hikayeleri yoksa, kullanıcı hikayelerinden en öncelikli olanlar seçilir ve bir sonraki iterasyon uygulanır.
- İlk sürüm sonunda yazılım müşteri tarafından kullanılabilir. Diğer sürümlere sonraki iterasyonlarla devam edilir. [54,57]



Şekil V.2 Ekstrem Programlama Zaman Döngüsü

V.1.6. Ekstrem Programlama Proje Evreleri

Ekstrem programlama projesi farklı evrelerden oluşmaktadır. Her evre, kendine özel etkinlikler içermektedir. Şekil V.3’de ekstrem programlama proje evreleri ve etkinlikler gösterilmektedir.



Şekil V.3 Ekstrem Programlama Proje Evreleri [50]

Keşif Evresi (Exploration Phase):

İlk evre, ekstrem programlamada bir proje deneyimlerini keşfetme evresidir. Mimari deney ve ilk kullanıcı hikayelerinin gelişimini içermektedir. Her projenin bir kapsamı vardır. Bu kapsam genellikle sistemin ilk gereksinimlerinin elde edilmesi ile tespit edilmektedir. Kullanıcı hikayeleri, ekstrem programlamada birincil önceliklidir. Çünkü sistem için üst düzey gereksinimleri sağlamaktadır. Bu kritik aşama, daha sonra planlama sürecine dahil edilmektedir. Bu evrede müşteri, kullanıcı hikayelerini oluşturmaktadır. Programcılar, teknik altyapı için deney ve araştırmalarda bulunurlar.

Planlama Evresi (Planning Phase):

Planlama evresinin amacı, küçük ve değerli olan kullanıcı hikayelerinin uygulanması için programcılar ile müşterilerin sürüm iterasyon planları oluşturmalarıdır. Kullanıcı hikayeleri için beyin fırtınası görevleri oluşturulmalıdır. Bazen görevler, görev kartı yerine kısa bir model kullanılarak tanımlanabilir. Her bir görev için bir görev kartı yazılırken, programcılar kullanıcı hikayesi hakkında doğru tahminlerde bulunmak için çaba göstermelidir.

Sürüm İterasyonları (Iterations to Release Phase):

Kullanıcı hikayelerinin uygulaması, sürüm iterasyonları evresinde gerçekleşmektedir. Müşterinin belirlediği kabul testleri ile uygulamanın kontrolü yapılmaktadır. Bu testler programcılar ve testçiler tarafından uygulanmaktadır. Her iterasyon sonunda müşteriye çalışan bir yazılım teslim edilmektedir. Böylece müşteri görüşleri alınarak geri bildirim sağlanmaktadır. İterasyon bittikten sonra çalışma temposunu tespit etmek için önceki iterasyondan edinilen deneyimler kullanılmaktadır.

Bakım Evresi (Maintenance Phase):

Yazılım, bakım ve geliştirilmesi bu evrede yapılmaktadır. Bu evre, planlama, sürümde iterasyonlar ve ürünleştirme evrelerini kapsamaktadır. Bundan dolayı sistem destek ve operasyon gibi üretim etkinliklerini de içermektedir. Yazılım için küçük çaplı eklemeler ve hataların düzeltilmesi için işlemler yapılmaktadır. Müşteri gereksinimlerine göre bir sonraki büyük sürüm için çalışmalar yapılabilir. Bu aşamada tekrar keşif evresinden başlayarak proje evrelerinin uygulanması gereklidir.

[50,57]

KEŞİF	PLANLAMA	İLK SÜRÜM İTERASYONLARI	ÜRÜNLEŞTİRME	BAKIM
Amaç: - İlk sürüm için yeterli miktarda hikaye kartı tahmini yapılmalıdır. - Fizibilite sağlanmalıdır	Amaç: - İlk sürüm için hikayelerin ve tarihlerin kararlaştırılması.	Amaç: - Sürüm için bir test sistemi uygulanır.	Amaç: - Operasyonel Dağıtım	Amaç: - Geliştirme, Düzeltme - Sonraki sürümün oluşturulması
Etkinlikler: - Prototipler - Teknolojik altyapı keşfi - Hikaye kartları yazma ve tahmin etme	Etkinlikler: - Sürüm planlama oyunu - Hikaye kartları yazma ve tahmin etme	Etkinlikler: - Test ve Programlama - İterasyon planlama oyunu - Görev yazma ve tahmin etme	Etkinlikler: - Dokümantasyon - Eğitim - Pazarlama -	Etkinlikler: - Diğer sürümler, bu aşamaları yeniden uygulayabilir.

Şekil V.4 Ekstrem Programlama Proje Evreleri [58]

V.2. PROJE YÖNETİMİ

Proje yönetimi, bir projenin gereksinimlerini karşılamak üzere bilgi, beceri, araç ve tekniklerin tüm etkinliklere uygulanmasıdır. Projenin iyi yönetilmesi başarılı olmasındaki faktörlerden biridir.

V.2.1. Planlama Evresi

Proje planı, geliştirme sürecinde hangi sürede neler yapılacağını tanımlayan bir dokümandır. Proje öncesinde plan oluşturularak geliştirme adımları belirlenebilir. Yazılım hedeflerine gereksinimler oluşarak, gereksinimlere göre uygulamalar yapılmaktadır. Uygulama, gereksinimlerin program koduna çevrildiği aşamadır. Programcılar, müşteri gereksinimlerini uygulamaya geçirirler. Gereksinimleri müşteri tespit etmektedir.

V.2.2. Sürüm Planlaması

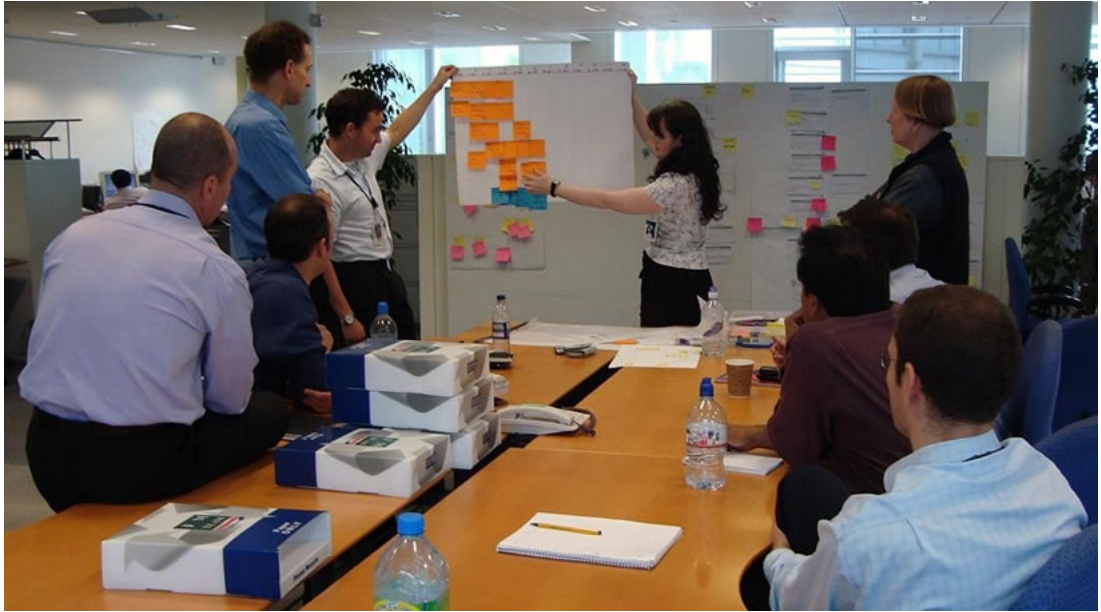
Ekstrem programlama projelerinde, sürüm planlaması ile proje planı gerçekleştirilir. Tanım olarak sürüm, yazılım sisteminde bir veya daha fazla özelliğin uygulamasını içeren kodlamadır. Sürümler, yazılım sürecinde bir ile üç aylık sürelerde meydana gelen özellikleri içermektedir. Her sürüm müşteri tarafından kullanılabilir durumdadır.

Sürüm planlama, müşteriye bir sonraki sürümde hangi gereksinimlerin olması gerektiğini hakkında destek olmaktadır. Programcılar, sürüm planlaması ile uygulama için gerekli teknik analiz yaparlar. Tahminleri programcılar yaparak sürüm planının oluşturulmasında görev alırlar. Sürüm planlama, bir ile iki hafta gibi sürede gerçekleşmektedir. Projede, sürüm planı yol göstericidir. Planda özelliklerin hangi

sıralamaya göre uygulanacağı ve yeni sürüm tarihleri yer almaktadır. Ekstrem programlama projelerinde sürüm planları sürüm planlama oyunlarında oluşturulmaktadır.

V.2.3. Sürüm Planlama Oyunu

Sürüm planlama oyununda, müşteri ve programcılar sonraki sürüm planlaması için toplanmaktadır. Oyunda amaç, yardımlaşma ile oyunu sonlandırmaktır. Oyun, sürüm planı içeren doküman oluşturularak sonlanmaktadır. Sürüm planlama oyununa başlamadan önce müşteri tüm kullanıcı hikayelerini oluşturmaktadır. Sürüm planlama oyununda hikaye kartlarına yazılan hikayeler temel alınmaktadır.



Şekil V.5 Sürüm Planlama Oyunu [13]

Kullanıcı Hikayeleri

Sürüm planlama oyunu, hikaye kartlarına yazılan kullanıcı hikayeleri ile oynanmaktadır. Kullanıcı hikayeleri, müşteri tarafından oluşturularak birkaç cümle ile yazılım özelliklerini tanımlamaktadır. Kullanıcı hikayelerinin, programcılar anlaşılabilir olması gereklidir. Çünkü programcılar, kullanıcı hikayelerini temel olarak uygulama gerçekleştirirler. Kullanıcı hikayeleri A5 yada A6 formatında olabilir.

Hikaye No: <u>001</u>	Bağlılık: <u>Yok</u>
Hikaye Başlık: <u>Sisteme giriş</u>	
Hikaye Açıklama: <u>Kullanıcı adı ve şifresi kullanılarak sisteme giriş yapılması</u>	
Hikaye Uygulama Süresi: {5, 10, 15, 20, 30, 40, 60, 90 , 120} dakika	
Hikaye Öncelik Değeri: 1	
Hikaye Tahmini: 2	

Şekil V.6 Kullanıcı Hikaye Kartı [16]

Hikaye kartları, müşteri ve programcılar arasında bilgi alışverişi sağlamaktadır. Programcılar, kullanıcı hikayeleri üzerinde konuşarak uygulama sürelerini ile ilgili tahmin yaparlar. Kullanıcı hikayesi ile ilgili sorular müşteriye iletilmektedir. Kullanıcı hikayelerinde müşteri öncelik değeri belirlendikten sonra hikaye kartı tahmin yapılmak üzere programcılara verilmektedir.

Programcı Tahminleri

Programcılar, kullanıcı hikayelerini detaylı bir şekilde öğrenmek zorunda değildir. Programcı için bu durum zaman kaybı olabilmektedir. Çünkü kullanıcı hikayesinde yer alan gereksinim, müşterinin ilk sürümü görmesiyle değişikliğe uğrayabilir. Programcıların, uygulama için gereken zamanı tahmin edebilecek kullanıcı hikayesini öğrenmeleri yeterlidir. Programcılar, kullanıcı hikayelerinin uygulama süreleri için yaptıkları tahminde planlama pokerini kullanırlar.



Şekil V.7 Planlama Poker Kartları [70]

Planlama pokeri olmadan yapılan tahminler iyi sonuçlar vermeyebilir. Bir programcının tahmini diğer programcılar etkileyebilir. Doğru tahminler yapmak için planlama pokeri oynanır. Her programcı Şekil V.5'deki kartların setine sahip olmalıdır.

Planlama pokeri oyunu şu şekilde oynanmaktadır.

Bir takım üyesi, ilk kullanıcı hikayesini okuyarak uygulama süresinin ne olması gerektiğini sormaktadır. Programcılar bir süre düşünerek planlama poker kartlarından birisini seçerek gösterirler. Kartlardaki değerler çoğu zaman farklı olabilir. En uzun ve en kısa süre tahmini yapan programcılardan açıklama yapılması istenmektedir. Açıklamalara göre ortak bir değer aralığı seçilmektedir. Hikaye puanları tahminler için kullanılmaktadır. Örneğin 1 hikaye puanı 1 işgünü(8 saat) olabilmektedir. Programcılar kullanıcı hikayelerini karşılaştırarak tahminde bulunmaktadır. Örneğin kullanıcı hikayesi A için 2 hikaye puanı tahmin edilmişse, kullanıcı hikayesi B'de buna göre tahmin değeri verilmektedir. Kullanıcı hikayesi B büyüklük olarak A hikayesinin üç katı ise B hikayesi için tahmin değeri $2 \times 3 = 6$ hikaye puanı olmaktadır.

Yük Faktörü

Bir kullanıcı hikayesinin ideal şartlarda uygulanabilmesi için gereken zaman ile normal şartlarda uygulanması için gerekli zaman farklı olmaktadır. Programcılarının gün boyunca yazılım dışında meşgul oldukları alanlar bulunmaktadır. Bir programcının sekiz saatlik iş gününde sekiz saat program yazması ideal zaman olarak adlandırılmaktadır. İdeal zamandan toplantı ve diğer işlerde geçirilen zamanlar çıkarıldığında normal zaman elde edilmektedir. Kullanıcı hikaye tahminlerinde ideal ve normal zaman dilimlerinin dikkate alınması gerekmektedir.

Tahminlerde başarılı olabilmek için yük faktörü olarak bilinen değer kullanılmaktadır. Bu değer, kullanıcı hikayesinin uygulanması için kullanılan zaman ideal zamana bölünerek elde edilmektedir. Örneğin, bir kullanıcı hikayesi için 1 iş günü (8 saat) tahmin edildiğini ve programcı kullanıcı hikayesini 2 iş gününde tamamladığını kabul edelim. Bu durumda yük faktörü $16 / 8 = 2$ olarak hesaplanır. Yük faktörü için 2 ile 5 arasında bir değer normaldir. Tahmin yapılırken tahmin edilen uygulama zamanı yük faktörü ile çarpılmaktadır.

Örneğin, programcı bir kullanıcı hikayesini 2 iş gününde uygulayabileceğini düşünüyorsa kullanıcı hikayesi için tahmin süresi 2 değil, 2 iş günü x 2 yük faktörü = 4 olmalıdır. Bu şekilde daha gerçekçi tahmin elde edilmektedir.

Programcı Denemeleri

Programcılar, bir kullanıcı hikayesi için tahminde bulunmazsa küçük bir demo uygulaması yapılmaktadır. Uygulama süresi tahmin edilmeye çalışılmaktadır. Ekstrem programlama süreçlerinde bu işleme deney çözümü (spike solution) adı verilmektedir. Programcılar denemeler yardımıyla kullanıcı hikayesi için tahmin yapabilecek durumda olmaktadır.

Müşteri Kullanıcı Hikayesinin Bölünmesi

Her sürüm birden fazla iterasyondan meydana gelmektedir. Her iterasyon bir ile iki haftalık zamanları içermektedir. Seçilen kullanıcı hikayeleri bir iterasyon içinde uygulanabilir durumda olmalıdır. Bu durum uygulamanın bir iterasyon içinde yapılması için gereklidir. Aksi takdirde müşteriden kullanıcı hikayesini iki ya da daha fazla kullanıcı hikayesine bölmesi talep edilebilir. Müşteri, kullanıcı hikayesini bölerek, yeni oluşan kullanıcı hikayelerini tekrar tahmin edilmek üzere programcılara vermektedir.

Müşterinin Kullanıcı Hikayelerinin Sırasını Belirlemesi

Her kullanıcı hikayesi için programcılar tahminleri yapıldıktan sonra, müşteri kullanıcı hikayelerini program açısından sahip oldukları değerlere göre sıralamaktadır.

Tablo V.1 Kullanıcı Hikayeleri ve Değerlilik Sırası

Değer	Kullanıcı Hikayesi
Yüksek	A (3) B (2) C (1)
Orta	D (3) E (3) F (2)
Düşük	G (2) F (1) H (1)

Müşteri tarafından gruplanan kullanıcı hikayeleri Tablo V.1’de yer almaktadır. Parantez içinde kullanıcı hikayeleri için tahminler belirlenmiştir.

İterasyon Süresinin Belirlenmesi

Her sürüm birden fazla iterasyondan meydana gelmektedir. Her iterasyon bir ile iki haftalık zamanları içermektedir. İterasyon sonunda test edilerek çalışan sistem müşteriye teslim edilmektedir. İterasyon süresi bellidir. Bu süre bir, iki ya da dört hafta olabilmektedir. İterasyon için seçilen kullanıcı hikayeleri bu iterasyon süresinde uygulanabilir olmalıdır. Uygulamanın tamamlanması için iterasyon süresi dinamik olarak değiştirilmez. Yetişmeyen kullanıcı hikayelerinde, uygulama yarım bırakılır ya da hiç başlanmadan bir sonraki iterasyona aktarılmaktadır. Bir iterasyonda kaç tane kullanıcı hikayesinin uygulanacağı programcıların çalışma hızına (velocity) bağlıdır.

Programcı Çalışma Hızının Belirlenmesi

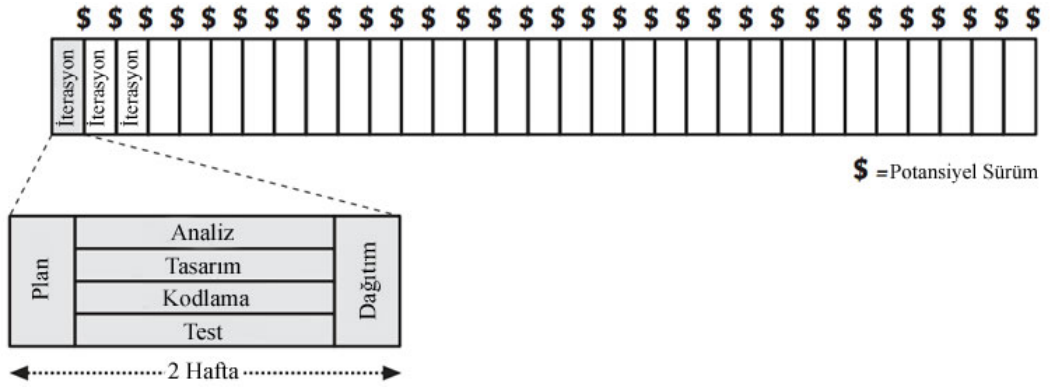
Programcıların bir iterasyon içinde uygulayabildikleri kullanıcı hikayelerinin hikaye puan toplamı, takımın o iterasyondaki hızıdır. Örneğin, takım bir iterasyonda 3 kullanıcı hikayesini uygularsa, kullanıcı hikayelerinin hikaye puan toplamı 5 ise ($2+2+1$), iterasyon çalışma hızı 5 olmaktadır. Sürüm planı yapılırken bu hızın bilinmesi gereklidir. Her iterasyon süresi belli olduğu için, bu iterasyon süresini aşacak şekilde kullanıcı hikaye seçimi yapılamaz. Eğer bir önceki iterasyonda çalışma hızı 5 hikaye puanı ise, bir sonraki iterasyon içinde 5 hikaye puanında kullanıcı hikayesi seçilmelidir. Eğer ilk iterasyon için planlama yapılıyorsa, o zaman programcıların çalışma temposunu tahmin etmeleri gerekmektedir.

Tablo V.2 Sürüm Planı

İterasyon	Hikaye Puanı	Kullanıcı Hikayesi
1	3	A
	2	B
2	1	C
	3	D
	1	H
3	3	E
	2	F
4	2	G
	1	F

V.2.4. İterasyon Planlaması

Sürüm planı, projenin uzun süreli planlanmasını içermektedir. Daha kısa sürelerde projeyi planlamak için iterasyon planları hazırlanmaktadır. Yazılım sistemini tek hamlede uygulamak yerine ekstrem programlama projelerinde yapılan işe göre uzunluktaki iterasyonlara bölmek gerekmektedir. Her iterasyon iki ile dört haftalık zamanları içermektedir. İterasyon süresi sabittir.



Şekil V.8 Ekstrem Programlama İterasyon ve Sürüm [52]

İterasyon öncesi iterasyon plan toplantıları yapılmalıdır. Bu toplantılar bir ile dört saat arası bir sürede gerçekleştirilmektedir. Toplantıda iterasyon içinde uygulanmak üzere seçilmiş kullanıcı hikayeleri kontrol edilmektedir. Diğer iterasyonlarda elde edilen deneyim ile tahminler tekrar kontrol edilmektedir. İterasyon toplantısına, müşteri, programcılar, testçiler, tasarımcılar ve diğer proje üyeleri katılmaktadır.



Şekil V.9 İterasyon Plan Toplantısı [71]

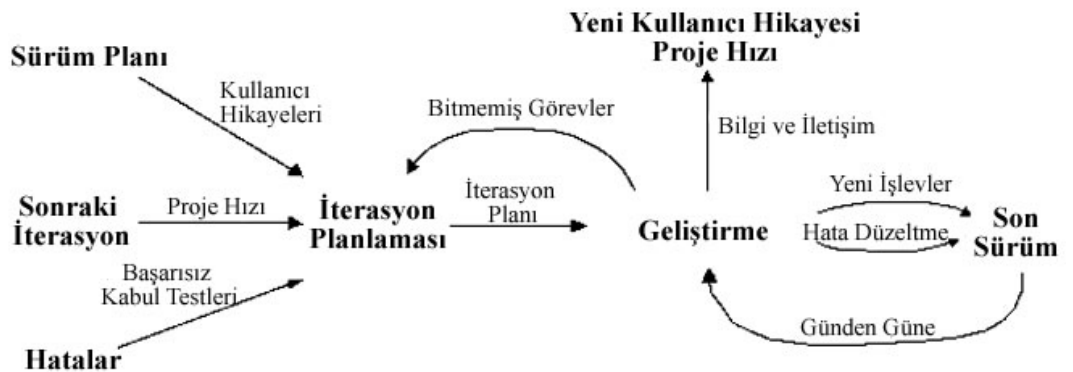
Programcılar, kullanıcı hikayelerini kontrol ederek görev listesi hazırlamaktadır. Hazırlanan görevler görev kartlarına yazılır. Görevlerin uygulama süresi poker oyunu ile tahmin edilmektedir. Tahminler saat üzerinden yapılmalıdır. Görev kartı, ilgili kullanıcı hikayesinin hikaye kartıyla gruplandırılmalıdır.

Hikaye	Yapılacaklar	İşlemde olanlar	Doğrulama	Tamamlanan
As a user, I... 8 points	Code the... 9 Code the... 2 Test the... 8	Test the... 8 Code the... 4 Test the... SC 8	Test the... SC 6	Code the... SC Test the... SC Test the... SC Test the... SC Test the... SC 6
As a user, I... 5 points	Code the... 8 Code the... 4	Test the... 8 Code the... 6		Test the... SC Test the... SC Test the... SC 6

Şekil V.10 Görev Kartlarının Kullanıcı Hikayesine Göre Gruplanması [72]

Örnek görev listesi:

1. LoginController sınıfı işletme (business) katmanında bulunan LoginManager sınıfını kullanarak login işlemini gerçekleştirir. LoginManager sınıfını oluşturulur.
2. LoginManager sınıfı LoginDao üzerinden bilgibankası işlemlerini gerçekleştirir. LoginDao isminde bir sınıf oluşturulur.
3. Üye bilgileri bilgibankasında customer isimli tabloda tutulur. Bu tablo yeterli mi, kontrol et.



Şekil V.11 Ekstrem Programlama İterasyon Görsel Modeli

V.2.5. Kabul Testleri

Kabul testleri, görevlerle karşılaştırıldığında daha somut yapıda olmaktadır. Kabul testleri, programcının işini kolaylaştırmaktadır. Kabul testlerinde, testler yazılım sistemindeki özellikleri gösterdiğinden ilerlemede ölçü olmasını sağlamaktadır. Bu yüzden iterasyon planlamasında görev yerine kabul testleri oluşturulması daha iyi olmaktadır.

Örnek kabul test listesi

1. Kullanıcı login sayfasına gider. Email adresi ve şifre alanlarını boş bırakarak login butonuna tıklar. Kullanıcıya “Lütfen email adresinizi ve şifrenizi giriniz!” hata mesajı gösterilir.
2. Kullanıcı login sayfasına gider. Email adresini girer ve şifre alanını boş bırakarak login butonuna tıklar. Kullanıcıya “Lütfen şifrenizi giriniz!” hata mesajı gösterilir.
3. Kullanıcı login sayfasına gider. Email adres alanını boş bırakarak, şifresini girer ve login butonuna tıklar. Kullanıcıya “Lütfen email adresinizi giriniz!” hata mesajı gösterilir.
4. Kullanıcı login sayfasına gider. Email adresi ve şifreni girer ve login butonuna tıklar. Email adresi ve şifre doğrudur. Login işlemi gerçekleşir. Üye, hoş geldiniz sayfasına yönlendirilir.
5. Kullanıcı login sayfasına gider. Email adresi ve şifreni girer ve login butonuna tıklar. Email adresi geçersizdir. Kullanıcıya “Email adresiniz geçersizdir, lütfen tekrar ediniz!” hata mesajı gösterilir.
6. Kullanıcı login sayfasına gider. Email adresi ve şifreni girer ve login butonuna tıklar. Email adresi geçerli olmasına rağmen şifre hatalıdır. Kullanıcıya “Şifre hatalı, lütfen tekrar deneyiniz!” hata mesajı gösterilir.

V.2.6. Dijital Hikaye Kartları

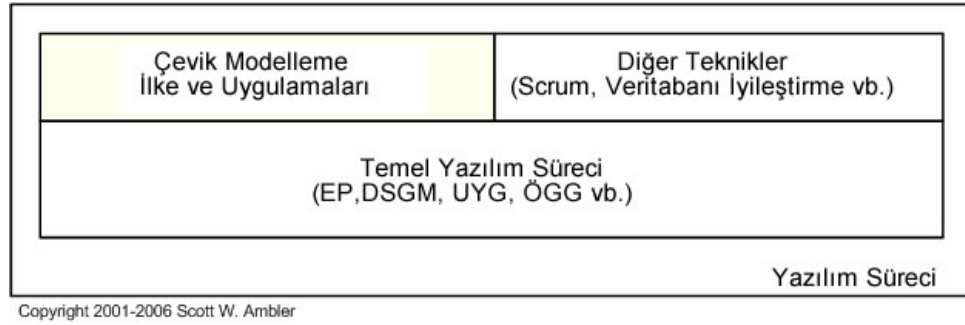
Bir bilgisayar programı yardımıyla hikaye kartları yapılabilir. Bu işlem için web tabanlı programlar kullanılabilir. Web tabanlı proje yönetimi için kullanılan programlarda iterasyonlar içinde yer alan dijital hikaye kartları oluşturulabilir. Her dijital kart bünyesinde görev listesi oluşturulabilir. Ayrıca bu kartlar üzerine birim ve kabul testleri not edilebilir. Açık kaynak kodlu Xplanner programı proje yönetimi kullanılan programlardan biridir. [57]

BÖLÜM VI

ÇEVİK MODEL TABANLI GELİŞTİRME

VI.1. ÇEVİK MODELLEME TANIMI

Çevik Modelleme, yazılım sistemlerinde etkili modelleme ve dokümantasyon için uygulamaya dayalı bir metodolojidir. Çevik modelleme, yazılım modellemesi için değerler, ilkeler, uygulamaların bir toplamıdır. Bir yazılım geliştirme projesi üzerinde etkili ve hafif bir şekilde uygulanabilmektedir. Çevik modelleme, ekstrem programlama, UYG gibi metodolojileri kendine katarak bir yazılım geliştirme süreci oluşturmaya olanak sağlamaktadır. Projeye, ekstrem programlama ya da kullanılan bir temel süreç ile başlanmalıdır. Yazılım süreci, çevik modelleme yanında ihtiyaçları yansıtan diğer tekniklerle değişime uğramaktadır. Böylece ihtiyaçları karşılayan süreçler oluşturulabilir. Çevik modellemede hedef, etkin ve hafif modellemeye ait olan değerler, prensipler ve uygulamalar bütünü tanımlama ve uygulamaya nasıl geçirileceğini göstermektir.



Şekil VI.1 Çevik Modellemenin Diğer Yazılım Süreçlerini Geliştirmesi [75]

Çevik modelleme, ekstrem programlamada yer alan basitlik, iletişim, geri bildirim, cesaretlendirme ve alçak gönüllük değerlerini benimsemektedir. Modellemenin başarı anahtarı, etkili iletişime sahip olarak tüm proje paydaşları arasında basit bir çözümün geliştirilmesi için gayret edilmesidir. Bu tüm ihtiyaçları karşılamaktadır. Çevik Modelleme, sabit bir tarifi olan süreç değildir. Modelin nasıl yaratılacağı tanımlama yerine modellemede etkin olabilme üzerinde durulmaktadır. Çevik Modelleme, modellemenin azaltılması değildir. Çevik Modelleme, çevik yazılım geliştirme değerlerinin projede etkin olması için anahtar görevi yapmaktadır.

VI.2. ÇEVİK MODELLEME İLE MODEL OLUŞTURMA

Modelleme yaparken sadeliğin önemi ve çalışırken taleplerin zaman içinde değişiminden dolayı değişime bağlı kalmak gerekmektedir. Zaman içinde sistem, artan değişikliğe yanıt imkanı vererek gereksinimlerin karşılanmasını sağlamaya çalışmaktadır. Değişime yanıt vermede sürece yönelik bir modelleme yapılabilmektedir. Verim sağlamak için geliştirme araçlarınızda çok yönlü modellere ihtiyaç olabilmektedir. Etkin bir modelleme için kullanılan araçların tanınması ve modellerin iyi bilinmesi gereklidir. Çevik modelleme, her şeyi içeren büyük bir modeli bir anda yapmak yerine, küçük adımlar atarak modellemeyi yapılmasını sağlamaktadır. Modelleme, yazılımın soyut tasviri olduğundan her zaman doğru olmayabilir. Fikrinizin pratikte de uygulanabilir olduğunu ispatlamaya çalışmak için kod yazarak çaba gösterilmelidir. Proje katılımcılarının aktif katılımı modelleme çalışmaları için önemlidir. Çünkü proje katılımcıları ne istediklerini bilerek gerekli olan geri bildirimi kolaylıkla yapabilirler.

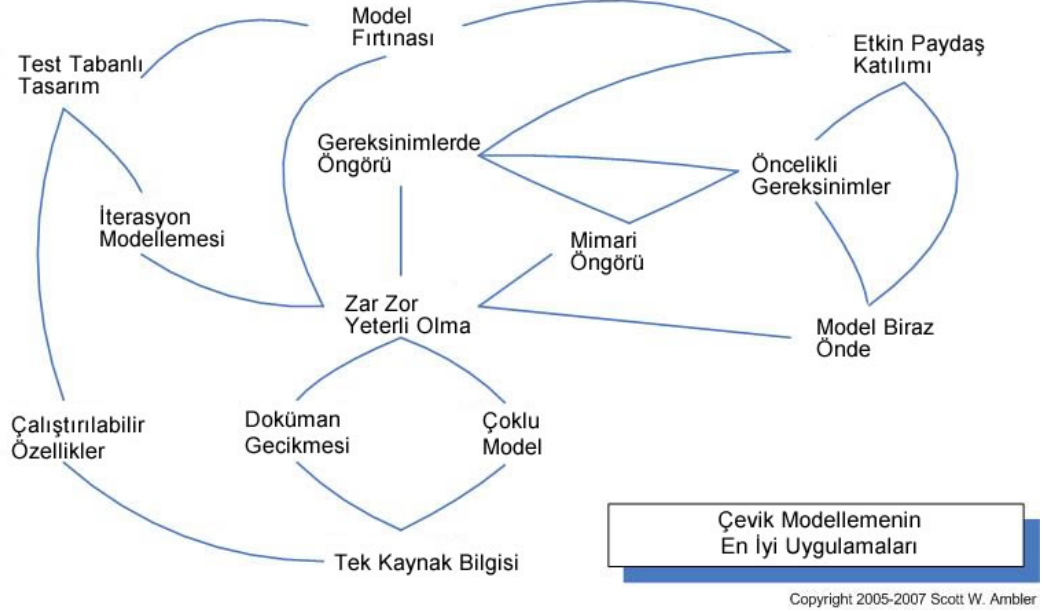
Modeller yaratmanın iki temel nedeni vardır. Bunlar bir kavramı (mesela bir sistemi kimin nasıl dizayn edeceğini anlama) anlama ya da takımın neler yaptığını/yapacağını anlatmak için modelleme yapmaktır. Çevik modellemede, temel ve tamamlayıcı bir takım ilkelerin tanımlanması gereklidir.

Tablo VI.1 Çevik Modelleme İlkeleri

Temel İlkeler	Tamamlayıcı İlkeler
Basitliği tercih etmek	- İçerik tasarımıdan daha önemlidir. - Açık ve dürüst iletişim
Değişimi kabul etmek	
Sonraki çalışmalara başlamak ikinci hedeftir	
Artımlı değişim	
Maksimum paydaş yatırımı	
Bir amaç için modelleme	
Birden fazla modelleme	
Kaliteli çalışmak	
Hızlı geri bildirim	
Çalışan yazılım birinci hedeftir.	
Hafif hareket etmek	

VI.3. ÇEVİK MODELLEME UYGULAMALARI

2000 yılı sonbaharında başlayan bir çalışma ile Çevik modelleme, yazılım geliştirme projelerinde modelleme ve dokümantasyon için değerler, ilkeler ve uygulamaların toplamına odaklanmıştır.



Şekil VI.2 Çevik Modelleme Uygulamaları [75]

Çevik modellemede yer alan en iyi uygulamaların açıklaması aşağıda verilmiştir.

Etkin Paydaş Katılımı: Paydaşların zamanında bilgi vermesi gerekmektedir. Paydaşlar, geliştirme sürecini kapsayan araçlar ve teknikleri kullanarak dahil olması gerekmektedir.

Mimari Öngörü: Çevik projenin başında, çözüm için uygulanabilir teknik strateji belirlemede ilk olarak üst düzey mimari modellemenin yapılması gerekmektedir.

Doküman Gecikmesi: Mümkün olduğu kadar geç doküman oluşturulmasıdır. Değişim olasılığına karşı proje istikrarı lehine bir durumdur.

Çalıştırılabilir Özellikler: Çalıştırılabilir müşteri testlerinde gereksinimler belirtilmelidir. Çalıştırılmayan statik dokümanlar yerine çalıştırabilir geliştirici testleri ile tasarım yapılabilir.

İterasyon Modellemesi: Her iterasyon başında, iterasyon faaliyetlerinin bir parçası olarak biraz modelleme yapılmalıdır.

Zar Zor Yeterli Olma: Bir model veya belge ihtiyacının bundan sonra mevcut durum için yeterli olmasıdır.

Model Biraz Önde: Bazen gereksinimlerde, öncelik yığınlarında karmaşıklık olabilmektedir. Toplam riski azaltmak için iş ögesi yığını öncelik olarak yukarı almak gereklidir. Böylece model daha ayrıntılı tasarlanabilir.

Model Fırtınası: Bir yenileme boyunca gereksinimlerin arkasındaki detayları keşfetmek için ve bir tasarım sorunu ile ilgi düşünmede birkaç dakika olacak sürelerde model fırtınası olabilmektedir.

Çoklu Model: Her türdeki modelin güçlü ve zayıf yönleri olabilmektedir. Etkili bir geliştirici, bir dizi modelde mevcut durum için en uygun şekilde doğru modele başvurulmalıdır.

Öncelikli Gereksinimler: Çevik takımlarında öncelik sırası proje paydaşları tarafından tanımlanarak gereksinimler uygulanmalıdır

Gereksinimlerde Öngörü: Çevik projenin başında, projenin kapsamını belirlemek ve gereksinimler için ilk öncelik yığını oluşturmak için bazı gereksinimlere ihtiyaç duyulmaktadır

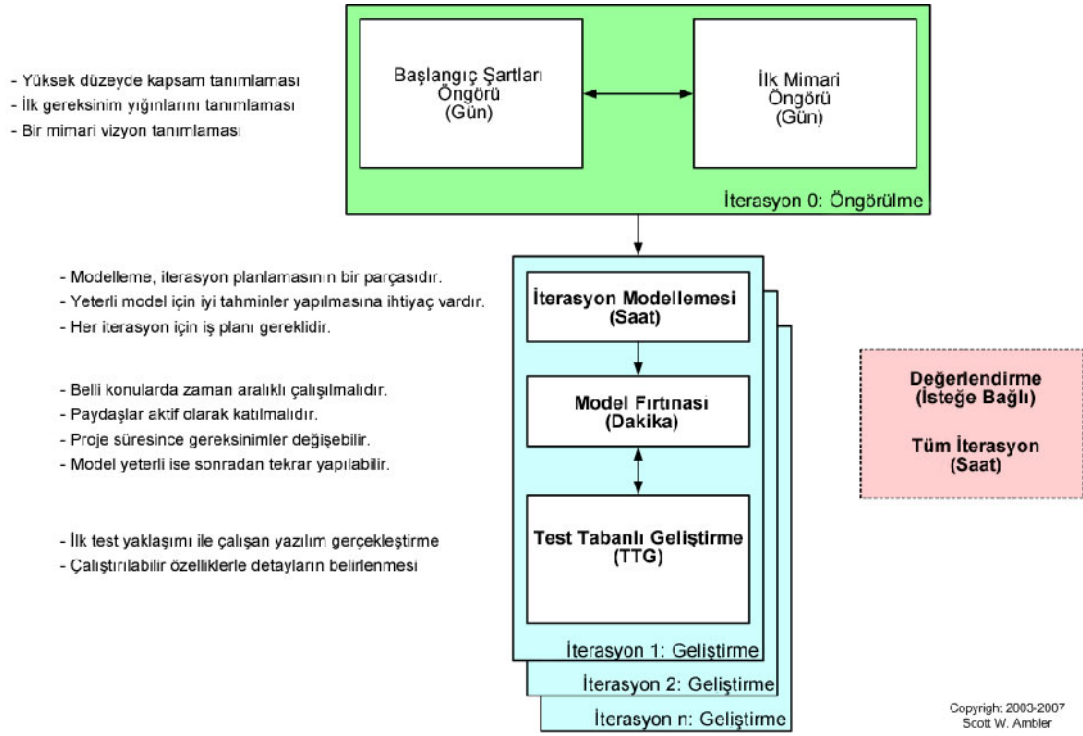
Tek Kaynak Bilgisi: Sadece bir yerden bilgiyi almak için gayret göstermektir.

Test Tabanlı Tasarım: Gereksinimler ve tasarım düzeyinde bir tane test yazılmalıdır. Bu test ile gereken kodlar karşılanmaktadır. Test tabanlı tasarım, doğrulama testi ve detaylı gereksinimlerin belirlenmesi için bir yaklaşımdır.

VI.4. ÇEVİK MODEL TABANLI GELİŞTİRME YAPISI

Adından anlaşıldığı gibi ÇMTG, model tabanlı geliştirmenin çevik versiyonudur. Model tabanlı geliştirme yaklaşımı, yazılım geliştirmede kapsamlı modellerin önceden kaynak kodu yazılarak oluşturulmasıdır. Model tabanlı geliştirmede iteratif yaklaşımlar bulunmaktadır. ÇMTG'nin farkı kapsamlı modeller yaratmak yerine kaynak kodu yazılmadan önce yeterli olan çevik modeller oluşturulmasıdır. ÇMTG, küçük ve etkin takım yaklaşımlarının ötesinde çevik yazılım geliştirme ölçeklenmesinin kritik bir stratejisidir.

Şekil VI.3'de ÇMTG'de bir sistemin sürümü için yüksek düzeyde bir yaşam çevrimi gösterilmektedir. Yaşam döngüsünde her bir kutu bir geliştirme etkinliğini temsil etmektedir. Öngörüler iki ana alt faaliyeti içermektedir. Bunlar başlangıç şartları ve ilk mimari öngörü etkinlikleridir. Bu etkinlikler sıfıncı iterasyon ile iterasyon döngüsünde bir defa olmak üzere yapılmalıdır. Sıfıncı iterasyon, ilk iterasyonlar için ortak bir terimdir. Parantez içinde yer zaman bilgileri oturumlar için ortalama süre uzunluğunu vermektedir.



Şekil VI.3 ÇMTG Yaşam Döngüsü - Proje Süresince Faaliyetlerin Modellenmesi

Öngörü:

Öngörü çabası, mimari tahmin ve sistemin nasıl kapsamda olmasını belirlemek için genellikle projenin ilk haftasında yapılmaktadır. Amaç özelliklerin detaylı olarak yazılması değildir. Amaç gereksinimleri keşfetmek ve proje için genel bir strateji oluşturulmasıdır. Bu amaç, kısa projeler için ilk birkaç saat içinde yapabilmektedir. Uzun projelerde ise iki haftalık bir zamanın ayrılması gereklidir.

İlk Gereksinimleri Modelleme:

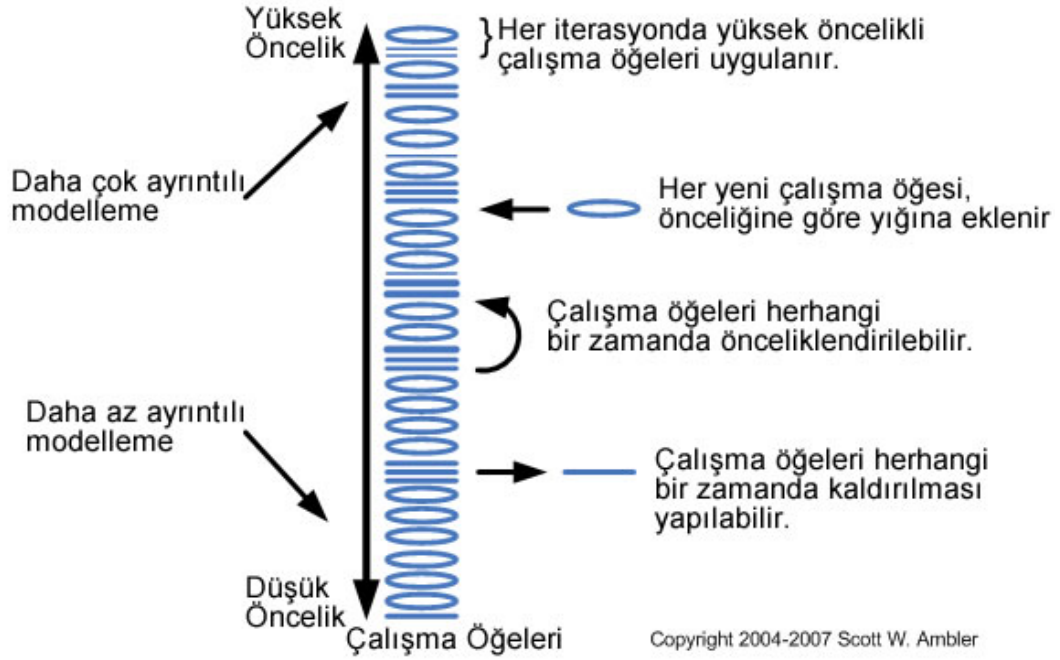
Bir sistemin ilk sürümü için, sürüm kapsamı gibi bazı üst düzey gereksinimleri belirlemek için birkaç güne ihtiyaç duyulmaktadır. Hedef, projenin tümü hakkında iyi bir fikir edinebilmektir.

İlk Mimari Modelleme:

İlk mimari modelleme çalışmasının hedefi, bir mimari belirlemeye çalışmaktır. Bu hedefe, teknik yönlendirmelerle geçerli ayarlama ve yeterli bilgi sağlamak için, mimari etrafında takım organizesi yapılmaktadır. Mimari modellemede beyaz tahta üzerine etki alanı modeli çizilmesi modellemeye katkı sağlayacaktır.

İterasyon Modellemesi:

Başlangıçta her iterasyon yapılırken takımın bir çalışma planı doğrultusunda iterasyonu gerçekleştirmesi sağlanmaktadır. Çevik takımları öncelik sırasına göre gereksinimleri uygular. Bu etkinlikte, öncelik tahmini için planlama pokeri oyunu oynanmalıdır. Doğru tahmin etmek için uygulamada gereksinimlerin anlaşılması gerekmektedir.



Şekil VI.4 Çevik Gereksinim Değişimi Yönetim Süreci [75]

Model Fırtınası:

Bir sorunu tanımlamada birkaç takım arkadaşı ile beraber hangi çözümün uygulanacağını tartışılması faydalı olabilmektedir. Modellemelerde birkaç kişi ile birlikte kağıt ve beyaz tahta üzerinde taslak çizimlerle sorun tartışılmalıdır. Model fırtınası oturumlarında genellikle bir proje takım üyesi doğaçlama olayları ve diğer üyelerde model üzerine sorular sorup beş ya da on dakika süren tartışmalar gerçekleşmektedir. Modelleme araçları etrafında takım üyelerinin toplanarak sorunun keşfedilmesi ve anlaşılması motivasyonu arttırmaktadır. [75]

BÖLÜM VII

ÇEVİRİMİÇİ ZAMAN ÇİZELGESİ SİSTEMİ

VII.1. GİRİŞ

Bu çalışmada, yazılım endüstrisi için çevik yazılım geliştirme süreçlerinin proje üretiminde yarattığı etkileri tespit etmek amaçlanmıştır. Yazılım endüstrisinde çeşitli müşteri ihtiyaçlarının, problemlerinin ya da süreç optimizasyonlarının çözümü için yeni bir yazılım projesi başlatılmalıdır. İlk aşamada problemlerin ve ihtiyaçların tespitleri yapılmalıdır. Bu tespitler, proje başlangıcında yapılan toplantılarda gereksinimlerin iyi tanımlanması ile olmaktadır.

Çevrimiçi zaman çizelgesi sistemi ile örneğin bir çalışan veya danışmandan hangi saatte kimin çalıştığı bilgisi alınabilir. Örnek proje zaman çizelgesi bir kullanıcı arayüzü dışında bazı arka plan işlemlerine de yapabilmelidir. Kullanıcı arayüzü web tabanlı ve bazı ekranlara sahip olmalıdır. Çalışılan saatin girilmesi, zaman çizelgesinin onaylanması, yönetim raporlarının oluşturulması gibi daha fazla özellik bu ekranlardan yapılabilirdir. Arka plandaki işlemler, toplu yapılan haftalık işlemleri içermektedir. Bu işler hatırlatıcı e-posta gönderilmesi gibi otomatik olarak çalıştırılan işlemler olabilir.

VII.2. İŞ GEREKSİNİMLERİ

Her zaman ne gibi iş sorunlarının çözülmüş olduğunu tanımlamak iyi bir fikirdir. Çözüm, müşteri için önemlidir. Buna ek olarak ne gibi müşteri beklentileri olduğunu anlamak gereklidir. Örneğin, çözüm ne zaman gerekli ve proje kapsamı ne olmalı gibi.

Çalışanları saat bazında yönetmek amacıyla zaman çizelgesi sistemi oluşturmak için hayali bir organizasyon yaratmalıyız. Başlarken önce problem tanımı ile başlayabiliriz. Bir kağıt kullanarak, zaman çizelgesinde çalışanların haftalık çalışma saatinin kontrol edilmesi yorucu olup hataya sebep olabilmektedir. Bu durumda otomatikleştirilmiş bir çözüm gereklidir. Bu çözüm, elektronik zaman çizelgesi şeklinde olabilmektedir.

Çevrimiçi zaman çizelgesi ile çalışma saatlerinin gönderilmesi, bu saatlere göre ödeme ve onaylama işlemlerinin yapılmasını sağlamaktadır. Proje, zaman çizelgesi durum değişikliklerine, otomatik bildirim ve haftalık hatırlatma gönderilmesine, zaman çizelgelerinin onaylanmasına izin vermelidir.

Bu problemler dikkate alındığında aşağıdaki özellikleri ayarlamak veya iş gereksinimleri içine almak gereklidir.

- Saatlik çalışmalar yapan kullanıcılar için web uygulamasında oturum gerekmektedir.
- Kullanıcılar, kendi zaman çizelgelerini her hafta göndermesi gerekmektedir.
- Bir kullanıcının yöneticisi, zaman çizelgesinin başarıyla gönderildiğini bildirmelidir.
- Yönetici, zaman çizelgesini onaylamalı veya reddetmelidir.
- Zaman çizelgesini göndermeyen kullanıcılara bir haftalık e-posta hatırlatması gönderilmelidir.

VII.3. YAZILIM GELİŞTİRME METODOLOJİSİ

Büyük ya da küçük her proje temel yapı ve süreçlere sahip olması gereklidir. Doğru dengeyi bulmak müşteri ihtiyaçlarına ve projenin büyüklüğüne göre değişmektedir. Projede daha az süreci olan, yeterli olan, az doküman gereksinimine sahip süreç kullanılacaktır. Bu projede, temel yazılım geliştirme süreci ekstrem programlama ve ÇMTG'ye dayalı olarak sağlanacaktır. Ekstrem programlama ve ÇMTG etkili ve hızlı yazılım uygulamaları oluşturmak için bazı temel yönergeleri sağlamaktadır. Ekstrem programlama ve ÇMTG tamamlayıcı yöntemlerdir. Çünkü ekstrem programlama disiplinli, tam yaşam döngüsüne sahip yazılım geliştirme sağlamaktadır. Diğer taraftan ÇMTG, genellikle modelleme ve dokümantasyon için etkili uygulamalar sağlamaktadır.

Keşif Evresi

Projenin bu evresinde aşağıdaki bazı örnek faaliyetler yer alabilmektedir.

Etki alanı modeli: Etki alanı modeli, temel iş kavramlarını (varlıklar) ve aralarındaki ilişkilerin tanımlanmasına yardımcı olmaktadır.

Kullanıcı arayüzü prototipleri ve hikaye panosu: Proje ile ilgili fikir edinilmesini için müşteri uygulamasında ilk ekran örnekleri oluşturulmaktadır. Hikaye akışları panoda yer almaktadır.

Kullanıcı hikayeleri: Birkaç kullanıcı hikayesi, projeye başlarken ilk sürümün düzenlenmesinde kullanılır.

Kapsam tanımı: Bu projenin ön kapsamını tanımlamak önemlidir. Çünkü neyin geliştirilmesi veya neyin ertelenmesi konusunda bilgi sağlanmaktadır. Kapsam tanımı aynı zamanda müşteri beklentilerini açıklamaktadır.

Analiz: Mimari şema taslağının beyaz tahtaya aktarılması, bir sözlük ve daha fazlası kombinasyonu içerebilmektedir. Çevrimiçi zaman çizelgesinde mimari yapı java teknolojileri temel alınarak gerçekleştirilmiştir.

Planlama Evresi

Sürüm planı: Sistemde bir sonraki sürüm için plan yapılmasıdır. Kolaylıkla elektronik tablo, kelime işlem programı ve/veya HTML tablo ile yapılabilir. Tüm kullanıcı hikayeleri, birkaç gruplandırılmış iterasyon ile son sürüm içine dahil olmaktadır. Sürümlerin uzunluğu sabittir. Genellikle bir ile üç ay arasındadır.

İterasyon planı: İterasyon öncesi geliştirilen bir plandır. Sonraki iterasyonda müşterinin uygulanmasını istediği kullanıcı hikayelerini içermektedir. İterasyonlar sabit uzunluktadır. Bir ile üç hafta arasında değişmektedir.

Standartların tanımlanması: Herhangi bir geliştirmeye başlamadan önce kodlama kurallarının tanımlanması, veritabanı adlandırma kuralları, süreç (inşa, entegre, teslimat) ve daha fazlasının standartlaştırması iyi bir adım olacaktır.

İterasyon ve Sürüm Evresi

İteratif geliştirmede, her iterasyon tasarım, kodlama, kullanıcı onayı ve geliştirilmeye hazır kodların teslimatını içermektedir. Bu üretim ortamına dağıtılabilir. Her iterasyon aşağıdaki faaliyetleri içermektedir

- Geliştirici görevleri, geliştirici tahminleri ve bir sonraki iterasyon için plan
- Geliştirici ve müşteri arasında genel görüşme
- CRC kartları, UML diyagramları tasarlamak
- Kod testi, ilk refactor kodu / veritabanı / mimari olarak gereklidir, daha sonra optimize edilir.

- Kullanıcı kabul testleri
- Üretim için iterasyon teslimatı. Bu adım küçük bir sürüm olarak da adlandırılabilir.

Bu şekilde tekrarlama yapılarak artımlı uygulamada sonraki sürümü oluşturabilirsiniz.

Proje Kapsamı

Projenin kapsamı, farklı biçimlerde tanımlanabilmektedir. Proje kapsamı, iki satırlık paragraf ile müşteri tarafından açıklanabilir ya da diyagramlar kullanılarak daha fazla yapılandırılabilir. İşlevsel ve işlevsel olmayan gereksinimler için projenin ayrıca tanımlamasını yapmak gereklidir.

Tablo VII.1 Örnek Proje Kapsam Tablosu

Kapsam	İşlevsellik
Dahil etme	Zaman çizelgesi, kullanıcılara giriş ve onay desteği vererek, çalışma saatlerine göre ödeme sağlayacaktır.
Erteleme	Zaman çizelgesinde, ücret kontrolünde devlet vergi ve sağlık giderleri gibi kesinti hesapları olmayacaktır.
Erteleme	Zaman çizelgesi, hastalık izni veya tatilleri izlemeyecektir.

Bakım Evresi

Bu bölümde uygulama, bakım durumuna geçmektedir. Bu evre kullanıcıya gerekli küçük düzeltme ve geliştirmeler için gerekli eğitimi (kullanıcı hikayeler şeklinde) içerebilmektedir. Bu evrede müşteri başka bir sürüm yapmak isteyebilir. Bu durumda keşif evresinden itibaren yeniden işlemlere başlanmalıdır.

VII.4. PROJEYE EP ve ÇMTG UYGULANMASI

Genel bir yazılım geliştirme sürecinde bazı arka planda yürütülen işlemler vardır.

VII.4.1. Etki Alanı Modeli

Etki alanı modeli, tüm veri varlıkları ve ilişkilerini bağlayarak hangi esaslara sahip olduklarını göstermektedir. Etki alanı modeli, iş kullanıcıları ve analistler gibi alan uzmanları ve iş bilgisi olan insanlar ile yapılan taslak bir çalışmadır.

Şekil VII.1’de zaman çizelgesi projesi için etki alanı modelini göstermektedir. Projeyi ileri götürmesi için basit bir etki alan modeli ile başlamak iyi olacaktır.



Şekil VII.1 Zaman Çizelgesi Etki Alanı Modeli

Kullanıcı: Çalışanları modellemek için kullanılmaktadır.

Zaman Çizelgesi: Zaman çizelgesi sistemini modellemek için kullanılmaktadır. Kullanıcı ve zaman çizelgesi arasında 1 – 0..* ilişkisi bulunmaktadır. Buna göre bir kullanıcı sıfır, bir ya da birden fazla zaman çizelgesine giriş yapabilmektedir. Her giriş sadece bir kullanıcıya ait olmaktadır.

Durum: Her zaman çizelgesi işleminde bir durum değeri bulunmaktadır. Zaman çizelgesi ve durum arasında 0..*-1 ilişkisi bulunmaktadır. Buna göre zaman çizelgesinde gerçekleşen işlemde sadece bir durum meydana gelebilmektedir. Bir durum içinde sıfır, bir ya da birden fazla zaman çizelgesi işlemi olabilmektedir.

Zaman: Zaman çizelgesinde, zaman aralıkları olduğundan zaman adında bir sınıf oluşturulur. Zaman çizelgesi ve zaman arasında 1-0..* ilişkisi bulunmaktadır. Buna göre zaman çizelgesinde sıfır, bir ya da birden fazla zaman aralığı olabilmektedir. Her zaman aralığı bir zaman çizelgesinde yer almaktadır.

Departman: Zaman aralıkları değişik departmanlara ait olabilmektedir. Zaman ile departman arasında 0..*-1 ilişkisi bulunmaktadır. Buna göre departmanda sıfır, bir ya da birden fazla zaman aralığı olabilmektedir. Bir zaman aralığı sadece bir departman içinde yer alabilmektedir.

VII.4.2. Kullanıcı Arabirimi Prototipi

Projede müşterinin web uygulamasını test etmesini sağlamak için hızla gerçek olmayan prototip ekranlar hazırlanabilmektedir. Erken geliştirilen prototip ekranlarda, uygulamanın yüzü uygulama inşasına motivasyon ve heyecan sağlamaktadır. Şekil VII.4’de saat girişlerinin ekranlarda nasıl gösterileceği yer almaktadır.

Basit bir HTML sayfası olarak hazırlanarak müşteriye uygulama ara yüzü gösterilebilmektedir. Böylece Java web sunucusu çalıştırmak yerine yerel bir tarayıcıda ara yüz incelenebilir.

Oturum Giriş

Lütfen kimlik doğrulama bilgilerinizi giriniz.

Kullanıcı Id: Parola:

Şekil VII.2 Zaman Çizelgesi Oturum Giriş Ekranı

Zaman Çizelgesi Listesi

Yeni bir zaman çizelgesi listesi eklemek için [tıklayınız](#) veya Aşağıdaki listeden seçiniz.

Bitiş Süresi	Saat	Zaman Çizelgesi Id
Şubat 22, 2010	39.50	1234
Şubat 15, 2010	43.00	1239
Şubat 08, 2010	40.00	1242
Şubat 01, 2010	40.00	1299

Şekil VII.3 Zaman Çizelgesi Liste Ekranı

Saat Girişi

Bitiş Süresi: Şubat 22, 2010

Departman	Pzt	Sa	Çrş	Prş	Cm	Cmt	Pa	Toplam
Bilgi Teknolojileri ▼	4.00	8.00	4.00	0.00	0.00	0.00	0.00	16.00

Şekil VII.4 Zaman Çizelgesi Saat Girişi Ekranı

Kullanıcı: Erdem ATAY				Bitiş Süresi : Şubat 22, 2010				
Departmant	Pzt	Sa	Çrş	Prş	Cm	Cmt	Pa	Toplam
Bilgi Teknolojileri	8.0	8.0	8.0	7.5	6.5			38.0

Şekil VII.5 Zaman Çizelgesi Saat Çıktı Ekranı

Zaman Çizelgesi Onaylama

Kullanıcı	Haftalık Saat	Onaylama
Erdem ATAY	65.00	☐ Evet ☐ Hayır
Hilal DEMİR	40.00	☐ Evet ☐ Hayır
Levent GELMEZ	35.00	☐ Evet ☐ Hayır
Sema LEYLEK	37.50	☐ Evet ☐ Hayır
TOPLAM	177.50	

Şekil VII.6 Zaman Çizelgesi Onaylama Ekranı

Ödeme İşareti

Kullanıcı	Haftalık Saat	Ödeme İşareti
Erdem ATAY	65.00	☒
Hilal DEMİR	40.00	☐
Levent GELMEZ	35.00	☒
Sema LEYLEK	37.50	☐
TOPLAM	177.50	

Değişiklikleri Kaydet

Sıfırla

Şekil VII.7 Zaman Çizelgesi Ödeme İşareti Ekranı

Rapor: Kullanıcı Saatleri

Kullanıcı	Tip	Haftalık Saat
Erdem ATAY	Personel	65.00
Hilal DEMİR	Yönetim	40.00
Levent GELMEZ	Personel	35.00
Sema LEYLEK	Personel	37.50
Ortalama Saat		39.00
TOPLAM		177.50

Yazdır

İptal

Şekil VII.8 Zaman Çizelgesi Kullanıcı Saatleri Ekranı

Rapor: Genel Özet

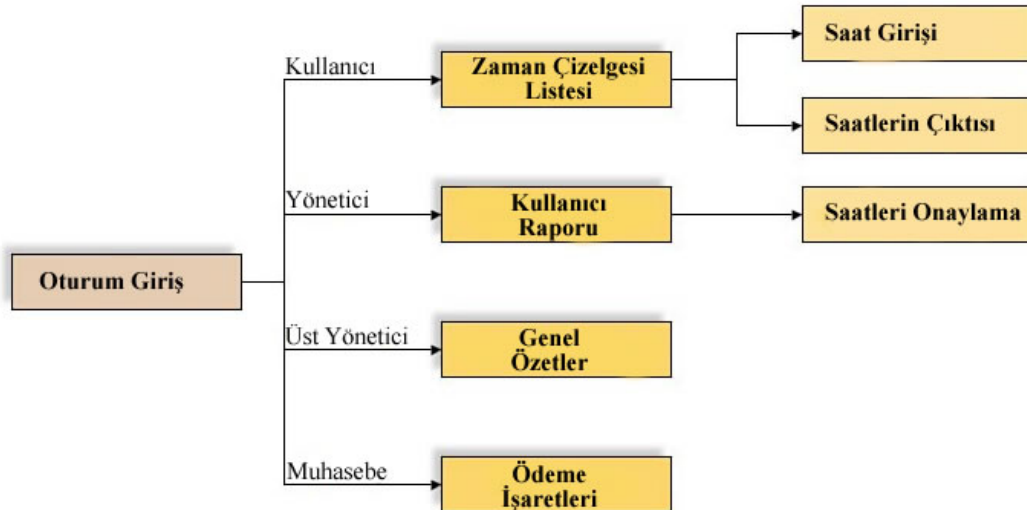
Yönetici	Haftalık Saat	Durum
Ekrem ACAR	320.00	12 ödenmiş 0 ödenmemiş 0 onaylanmış 0 onaylanmamış
Esra DEMİRLİ	367.00	6 ödenmiş 3 ödenmemiş 2 onaylanmış 0 onaylanmamış
Tuvara ÖZTÜRK	312.00	7 ödenmiş 4 ödenmemiş 1 onaylanmış 1 onaylanmamış
Mustafa ZENBİL	225.00	2 ödenmiş 1 ödenmemiş 3 onaylanmış 0 onaylanmamış
Ortalama Saat	300.00	
TOPLAM	1300.00	

Yazdır İptal

Şekil VII.9 Zaman Çizelgesi Genel Özet Ekranı

VII.4.3. Hikaye Panosu

Bir hikaye panosu, diğer adlarıyla kullanıcı arayüzü akış şeması veya web sitesi haritası, çeşitli uygulama ekranlarına ait temel bir navigasyon haritasının gösterilmesidir. Şekil VII.10'da proje için hikaye panosu gösterilmektedir. Kullanıcılara ait personel ve yönetici rollerine ait işlevlere ilgili ekranlardan ulaşılabilmektedir.



Şekil VII.10 Zaman Çizelgesi Hikaye Panosu

VII.4.4. Kullanıcı Hikayeleri

Zaman çizelgesi projesinde, verilen iş gereksinimleri ve kullanıcı arayüzü prototipleri tanımladıktan sonra kullanıcı hikayelerini tanımlayabilirsiniz. Kullanıcı hikayeleri sadece gereksinimleri tanımlamak amacıyla kullanılmayacaktır. Aynı zamanda java sınıfları adında, kullanıcı hikaye adı kullanımında ve kabul testleri oluşturmada kullanılmaktadır. Zaman çizelgesi uygulaması için kullanıcı hikayeleri, öncelikler ve tahminler (puan) Tablo VII.2’ de verilmiştir.

Tablo VII.2 Kullanıcı Hikayeleri, Öncelikler ve Tahminler

#	Hikaye Adı (Etiket)	Hikaye Açıklaması	Öncelik	Puan
1	Saat Girişi	<i>Kullanıcı çalışma saatlerini girip ve kaydedebilir.</i>	1	2
2	Zaman Çizelgesi	<i>Kullanıcı listesi, zaman çizelgesi listesinde görülür. Önceden girilmiş ve değiştirilebilir olanlar tıklanır.</i>	1	1
3	Oturum Giriş	<i>Kullanıcı, geçerli bir id kimlik ve parolayla sisteme giriş yapabilir.</i>	2	1
4	Oturum Çıkış	<i>Kullanıcılar sistemden mevcut oturumu kapatarak çıkabilir.</i>	2	1
5	E-posta hatırlatma: Kullanıcı	<i>Bir hatırlatıcı e-posta, kullanıcılara her Cuma saat 2 de gönderilir.</i>	2	1
6	Zaman Çizelgesi Baskı	<i>Kullanıcılar, tarayıcıda en iyi biçimlerle ve otomatik yazıcı ekran penceresini kullanarak zaman çizelgesi yazdırılabilir.</i>	3	1
7	Rapor: Benim Saatim	<i>Haftalık saatlerin özetini yazdırmak /görüntülemek için “Benim Saatim” isimli bir rapor çalıştırabilir.</i>	3	1
8	Zaman Çizelgesi Teslimi	<i>Kullanıcılara, saatleri girilmiş zaman çizelgesi gönderebilir.</i>	3	1
9	Rapor: Kullanıcı Saatleri	<i>Yönetici, altında çalışan tüm personel için belirli bir hafta saatinin özetinin görüntülediği kullanıcı raporu isimli bir rapor çalıştırabilir.</i>	4	1
10	Rapor: Genel Özet	<i>Üst yönetici, Şirketteki tüm yöneticiler için belirli bir hafta saatinin özetinin görüntülediği genel özet isimli bir rapor çalıştırabilir.</i>	4	2
11	Zaman Çizelgesi Onay	<i>Yönetici zaman çizelgesini onaylar ya da onaylamaz. Bildirimler, Muhasebe departmanı ve kullanıcıya e-postayla gönderilir.</i>	5	1
12	Zaman Çizelgesi Ödeme	<i>Muhasebe, kullanıcılara ödeme durumlarını gösterebilir.</i>	5	1
13	E-posta hatırlatma: Yönetici	<i>Bir hatırlatıcı e-posta, tüm yöneticilere her Cuma saat 4 de gönderilir.Onay bekleyen zaman çizelgeleri var mı?</i>	5	1

Geliştirici, kullanıcı hikayeleri geliştikçe iterasyon başında daha doğru tahmin yapabilmektedir. Çünkü kullanıcı hikayeleri bölünebilmektedir. Kullanıcı hikaye girişlerini tamamlamak için, geliştirme görevlerinin içine en iyi zamanda bölünmesi gereklidir. Puan, yapılan projede göreceli ölçü birimidir. Örneğin 1 puan, 1 normal iş gününe, 1 ideal iş günü, 1 hafta veya bir başka döneme eşit olabilmektedir. Müşteri ve geliştirici bu konuda bir anlaşma yapmaktadır.

VII.4.5. Sürüm ve İterasyon Planı

Müşterinin (son kullanıcı ve/veya iş analisti) kullanıcı hikayeleri tanımlamasından sonra, müşteri ve geliştirme proje yöneticisi (ve/veya geliştirici), sonraki sürüm ve uygulama versiyonu için birlikte bir sürüm planı ortaya koymalıdır. Bir sürüm planı, aslında her sürümde çeşitli sistem sürümleri ve tarihleri listeleyen proje planıdır. Tablo VII.3’de projenin birinci sürümü artımlı olarak yapılmaktadır.

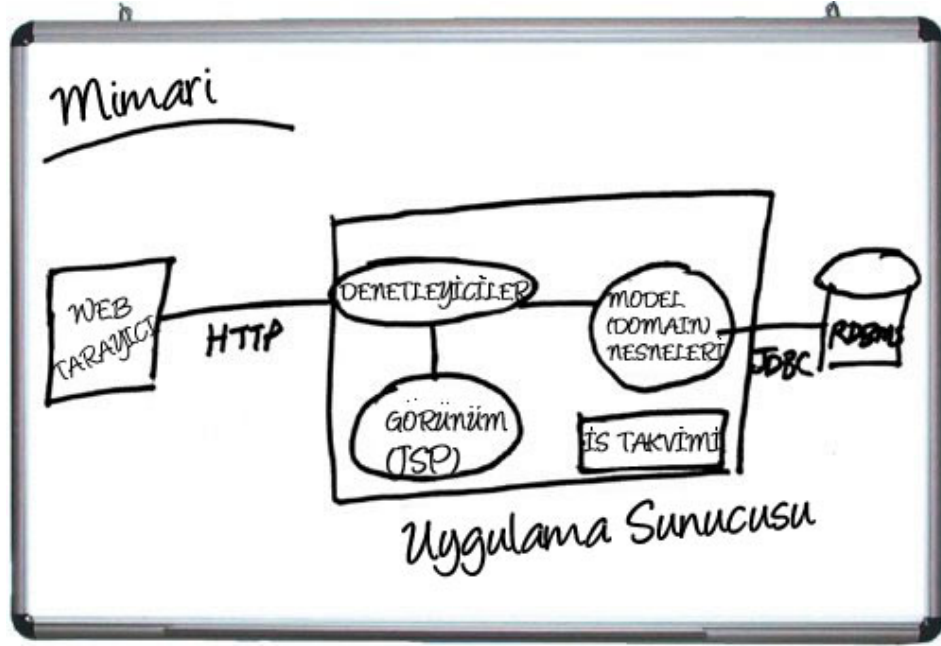
Tablo VII.3 Sürüm Plan 1.0

İterasyon	Özellikler	Yayın Tarihi
0	Çalışma ortamı kurulumu (JDK,Ant,JUnit) ve Hibernate kullanarak veri tabanı bağlantı demosu	14.12.2009
1	Küçük sürüm bütün önceliklerde 1 kullanıcı hikayeleri	28.12.2009
2	Küçük sürüm bütün önceliklerde 2 kullanıcı hikayeleri	11.01.2010
3	Küçük sürüm bütün önceliklerde 3 kullanıcı hikayeleri	25.01.2010
4	Küçük sürüm bütün önceliklerde 4 kullanıcı hikayeleri	08.02.2010

Sürüm planı, çeşitli iterasyonlar içeren yazılımın sonraki sürümü için tanımlanmış olmaktadır. Bundan sonra geliştirici ilk iterasyon üzerinde çalışmaya başlayabilmektedir.

VII.4.6. Beyaz Panoda Mimari

Yeterli bilgi elde edildiği için resmi olmayan mimari bir şema oluşturulabilir. Üst düzey bir mimari şema Şekil VII.11’de gösterilmektedir. Bu diyagram ile geliştiriciler ve müşteri önemli olan teknolojiler konusunda anlaşmaktadır. Örneğin, java, veritabanı, web/uygulama sunucusu gibi.



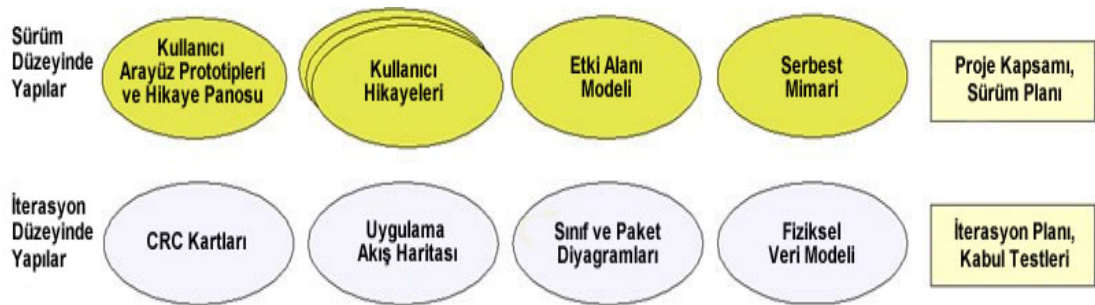
Şekil VII.11 Beyaz Panoda Taslak Mimari

VII.5. EP, ÇMTG TABANLI MİMARİ ve TASARIM MODELLEME

Bu aşamada, mimari ve tasarımda mümkün olduğu kadar detaya girmeden popüler teknolojilerin kullanımı ile zaman çizelgesinde proje süreçleri anlatılacaktır.

VII.5.1. Tasarım Yaklaşımı ve Yapı Seçimi

Şekil VII.12’ de gösterilen bazı yapıları, sürüm ve iterasyon düzeylerinde elde edebilirsiniz. Sürüm düzeyinde yapılar, yeni sürüm öncesinde üretilenlerdir. İterasyon düzeyinde yapılar, her iterasyon öncesi üretilenlerdir. Bunların hepsi her proje için zorunlu değildir.

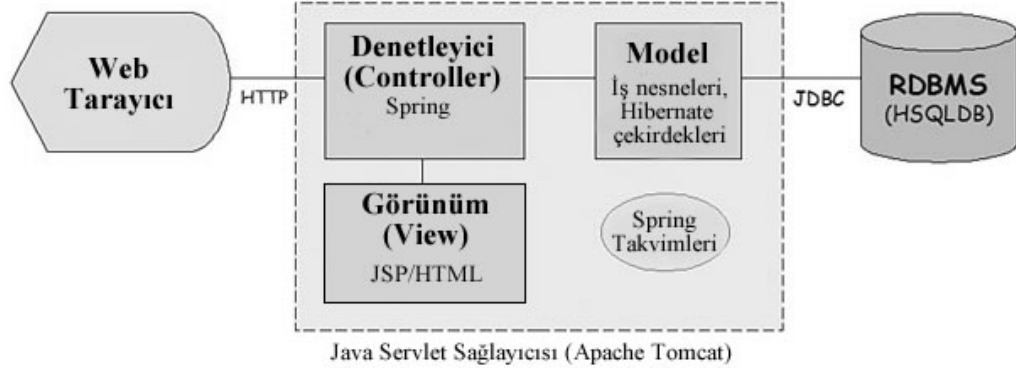


Şekil VII.12 Sürüm ve İterasyon Düzeylerinde EP/ÇMTG Tarzı Seçimler

Kullanılan metodolojiler ile ilgili iyi uygulamalar olmalıdır. Geliştiricilerin bu uygulamaları sevmesi önemlidir. Bu yöntemlerin sonucunda birçok başarılı proje yapılmıştır. Şekil VII.12’ de üretilmesi gereken yapılar gösterilmektedir.

VII.5.2. Serbest Mimari Diyagramı

Şekil VII.13’de örnek uygulama için üst düzey mimariyi göstermektedir. Burada beyaz panodaki taslak mimari elektronik tabanlı bir hale dönüştürülmüştür. Elektronik biçime çevirme zorunlu değildir.



Şekil VII.13 Zaman Çizelgesi İçin Üst Düzey Mimari

Mimari oldukça sade olarak tasarlanmıştır. Burada üç katmanlı bir mimari bulunmaktadır. Bunlar İstemci katman (Web Tarayıcı), orta katman (uygulama sunucusu) ve veri katmanı (veritabanı) katmanlarıdır. Ayrıca uygulama standardı olarak model görünümlü denetleyici (MGD) tasarım modelleri kullanılmaktadır. Günümüzde birçok java tabanlı web çerçevelerinde model görünümlü denetleyici kullanılmaktadır. Denetleyici, model ve görünümün denetlendiği HTTP giriş / web isteği noktasıdır. Görüntülemeye java sunucu sayfaları (Java Server Pages - JSP) kullanılmaktadır.

VII.5.3. CRC Kartları Kullanarak Sınıfları Keşfetmek

MGD’nin model kısmında, zaman çizelgesi etki alanı modelinde bazı varlık tanımlamaları bilinmektedir. Denetleyici bölümünde, kullanıcı hikayeleri isimleri bilinmektedir. Bunlar dikkate alındığında CRC kartları kullanarak ilk sınıf tasarımına devam edilebilir. CRC kartlarında, sınıf için sorumluluklar ve ortak çalışanlar olmalıdır. Tablo VII.4’de örnek CRC kart düzeninde üç bileşeni bazı açıklamalarla göstermektedir. CRC kartlar, sınıflar arasındaki etkileşimi keşfetmek için bir resmi olmayan nesneye dayalı teknik sağlamaktadır. Bilgisayar olmadan geliştiriciler veya kullanıcıların nesneleri keşfetmesi için resmi olmayan oturumda kullanılabilirler. CRC kartları resmi sınıf diyagramlarını geliştirmek için kullanılabilir. Gerçek sınıflar için örnek CRC kartları Tablo VII.4-VII.7 aracılığıyla gösterilmiştir.

Tablo VII.4 Örnek Bir CRC Kart Düzeni

Sınıf Adı (İsim)	
Sorumluluklar (Bu sınıfın yükümlülükleri, işletme yöntemleri, sıra dışı işlem, güvenlik yöntemleri gibi özellikler / değişkenler)	Ortak çalışanlar (Diğer sınıflar, üst düzey bir gereksinimde tam çözüm sağlamak için gereklidir)

Tablo VII.5 Zamancizelgesi Sınıfı İçin Örnek CRC Kart

Zamancizelgesi
Bitiş süresini bilir. Saati bilir. Departman kodunu bilir.

Tablo VII.6 ZamancizelgesiYoneticisi Sınıfı İçin Örnek CRC Kart

ZamancizelgesiYoneticisi	
Veritabanından veri çizelge(leri) alınır. Veritabanına zaman çizelgesini kaydeder.	Zamancizelgesi

Tablo VII.7 zclistedenetleyicisi Sınıfı İçin Örnek CRC Kart

ZclisteDenetleyicisi	
Zaman Çizelgeleri listesini görüntülemek için denetleyici.	ZamancizelgesiYoneticisi

VII.5.4. Uygulama Akış Haritası

Tablo VII.8’de yer alan biçim uygulamada kendi içinde gelişimi göstermektedir. Uygulama akış haritası gereklidir. Çünkü bir kullanıcı arayüzünde adım adım işlevlerin nasıl olacağını göstermektedir. Bu teknikte, kullanıcı hikayelerinin görünümü iyice planlanmaktadır. Uygulama akış haritasında CRC kartları veya sınıf diyagramları gibi teknikler karşılaştırılmaktadır.

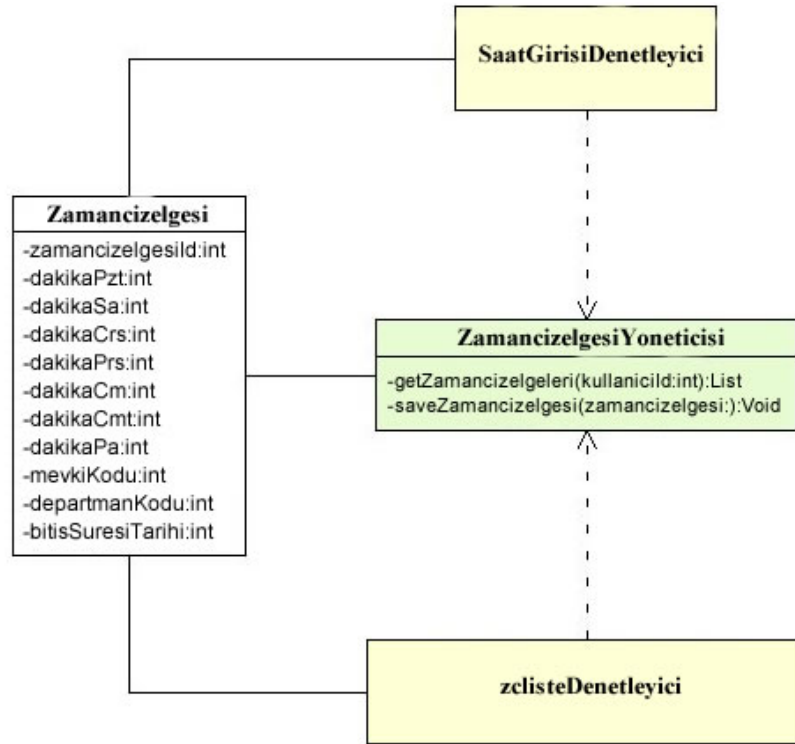
Tablo VII.8 Örnek Uygulama Akış Haritası

Hikaye Etiketi	Görünüm	Denetleyici Sınıfı	Ortak Çalışanlar	Etkilenen Tablolar
Zaman Çizelgesi	Zaman Çizelgesi Listesi	zclistedenetleyicisi	ZamancizelgesiYonet icisi	Zaman Çizelgesi
Saat Girişi	Saat Girişi	SaatGirisiDenetley ici	ZamanCizelgesiYone ticisi	Zaman Çizelgesi Departmanı

VII.5.5. UML Sınıf Diyagramı

UML(Unified Modelling Language), yazılım sistemlerinin belgelendirilmesi veya görsel hale getirilmesi için kullanılan bir dildir. Bu dilin kullanımı ile soyut modeller tasarlanabilir, karmaşık sistemler basitleştirilebilir, analiz edilebilir ve sonrasında yazılım geliştirme ortamlarına dönüştürülebilir.

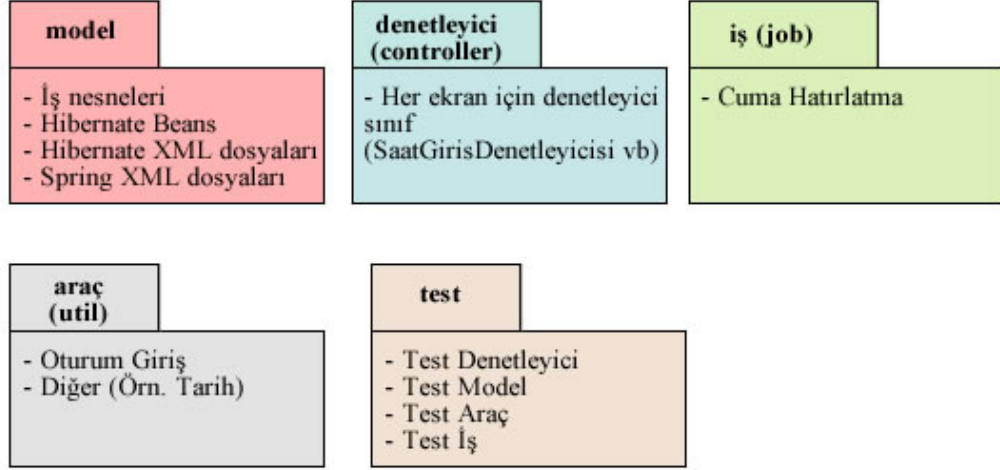
Yeterli bilgi ve kodlama için hareket geçmeyi CRC kartları ve Uygulama Akış Diyagramları sağlamaktadır. Fakat sınıf diyagramları uygun bir şekilde kullanıldığında proje sürecine katkılar sağlayabilir. Şekil VII.14’de sınıflar için sade bir sınıf diyagramı örnekte gösterilmektedir.



Şekil VII.14 Zaman Çizelgesi İçin Örnek Sınıf Diyagramı

VII.5.6. UML Paket Diyagramı

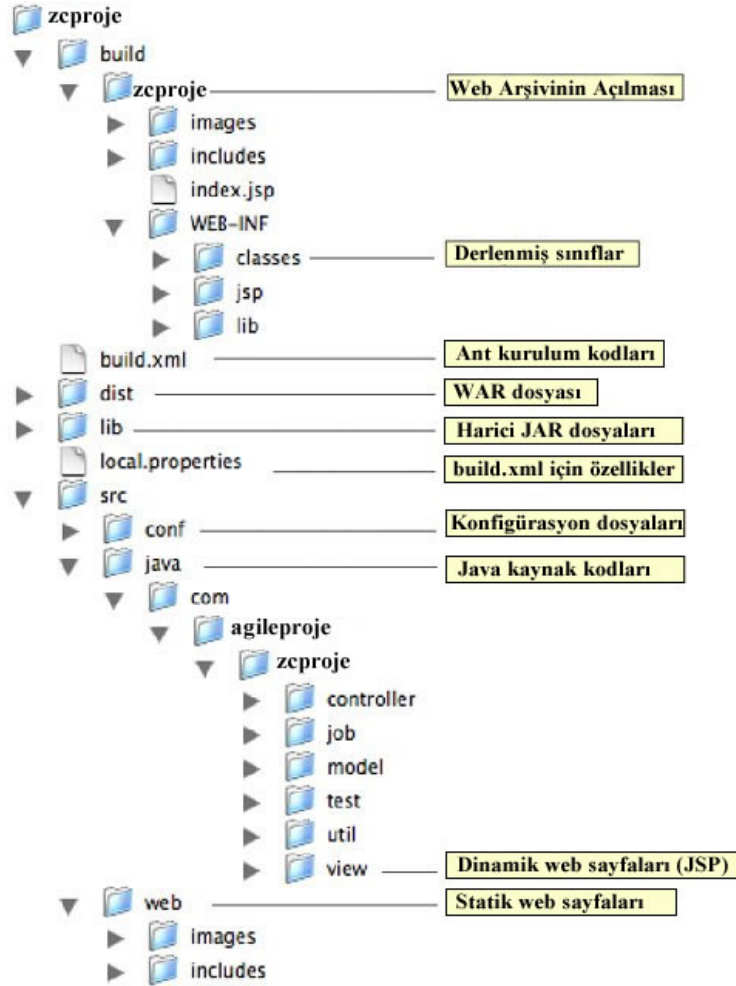
Paket diyagramı, yapısal diyagramdır ve modellenen sisteme belli bir açıdan bakışı göstermektedir. Gösterim genel olarak paketler, paket birleşmelerini içermektedir. Denetleyici paketi, sınıflar ile ilgili denetlemeden sorumludur. İş paketi, e-posta hatırlatmalarını sağlamaktadır. Araç paketi, ortak ve/veya yeni kodlamaları içermektedir. Test paketi, birim test kodlarını içermektedir.



Şekil VII.15 Zaman Çizelgesi İçin UML Paket Diyagramı

VII.5.7. Dizin Yapısı

Şekil VII.16’da örnek uygulama için dizin yapısı gösterilmektedir.



Şekil VII.16 Zaman Çizelgesi İçin Dizin Yapısı

Örnek dizin yapısı oluşturulduktan sonra sınıflar ve bazı örnek dosya isimlerini belirleyebiliriz. Örneğin zaman çizelgesi liste ekranı için daha öncede belirtilen isim kullanılabilir.

zcproje dizin yapısında zcproje/src/java/com/agileproje/zcproje altında oluşturulacak dinamik dosyaların örnek isimleri aşağıda verilmiştir.

- controller/zclistedenetleyici.java
- model/Zamancizelgesi.java
- model/ZamancizelgesiYoneticisi.java
- test/zclistedenetleyiciTest.java
- test/ZamancizelgesiYoneticisiTest.java
- view/zamancizelgesiListesi.jsp

Dizin yapısında önemli olan dizinler aşağıda açıklanmıştır.

src dizini : Bu dizin geliştirilecek tüm java, html/javascript, XML ve diğer tüm giriş/kaynak dosyalarını içermektedir.

build dizini : Kurulum dosyalarını içeren dizindir. Burada derleme, kütüphane ve web dosyalarının kopyaları vb. yapılar bulunmaktadır.

lib dizini: Uygulamayı çalıştırmamız için gerekli harici tüm JAR dosyaları bu dizinde yer almaktadır.

dist dizini: Web arşiv dosyası (.war), derlenmiş sınıf dosyaları (.class), Jar kütüphane dosyaları ve ilgili diğer web dosyalarını içermektedir.

VII.5.8. Kabul Testleri

Çevik yazılım projelerinin çoğunda olduğu gibi kabul testleri, gereksinimlerin ayrıntılı olarak sunulmasıdır. Bu testlere örnek olarak bir ekran görüntüsünde kullanıcının yapacağı işlemlerin listesinin belirtilmesi verilebilir. Gereksinimlerin tespitinde kabul testleri kullanılabilir. Çünkü bu testler ile müşteri uygulamadan beklentilerini listelemektedir. Birim testlerinde, bu testler kullanılmaktadır. Aşağıdaki bölümlerde kabul testleri listesi ve kullanıcı hikayeleri için uygulanması gerekenler ile geliştirme yapılacaktır. Projede, kabul testleri müşteri tarafından hazırlanmalıdır.

Oturum Giriş

- Kullanıcı kimliği en az 6 karakter olmalıdır. Şifre 8 ile 10 karakter arası olmalıdır.

- Sadece geçerli kullanıcılar oturum girişi yapabilir.

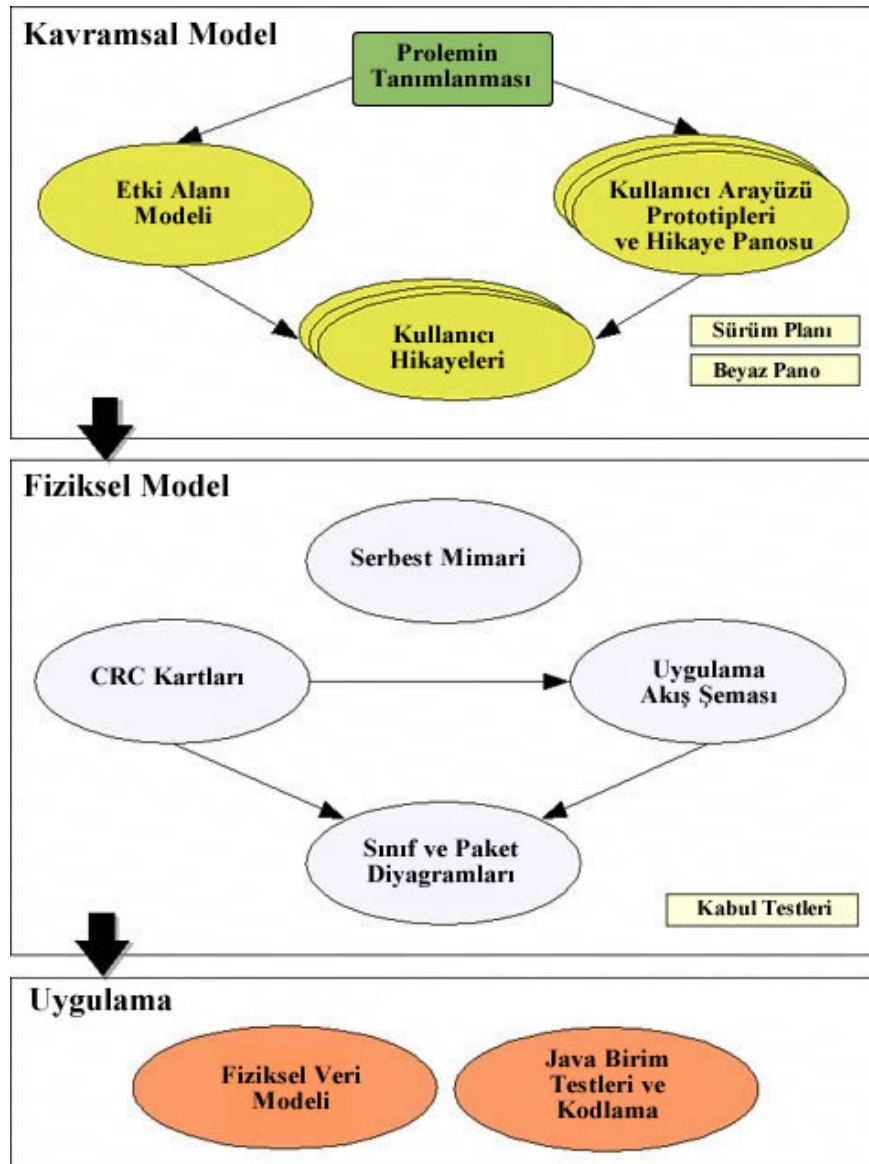
Zaman Çizelgesi Listesi

- Sadece bir kullanıcının kişisel zaman çizelgesine erişilebilir.

Saat Girişleri

- Saat değeri sayısal veriler içermelidir.
- Günlük saat değeri 16 saati geçemez. Haftalık saat değeri 96 saati geçemez.
- Saatlerin hangi departmana ait olduğu girilmelidir.
- Saatler iki ondalıklı basamak olarak girilebilir.
- Kullanıcı, sadece kendi zaman çizelgesini görüntüler ve düzenler.

VII.5.9. Kavramsal, Fiziksel ve Uygulama Katmanları



Şekil VII.17 Zaman Çizelgesi için Kavramsal, Fiziksel ve Uygulama Katmanları

VII.6. ÇALIŞMA ORTAMININ KURULMASI

VII.6.1. Java Platformu İçin Startdart Geliştirme Kitinin Kurulması

Java, Sun Microsystems mühendislerinden James Gosling tarafından geliştirilmeye başlanmış açık kodlu, nesneye yönelik, düzlemde bağımsız, yüksek verimli, çok işlevli, yüksek seviye, adım adım işletilen (interpreted) bir dildir. Java nesneye yönelik bir dil olduğundan tüm yazılım sınıflar ve nesneler üzerinden yürütülmektedir.

Bir Java yazılımı şu şekilde geliştirilebilir.

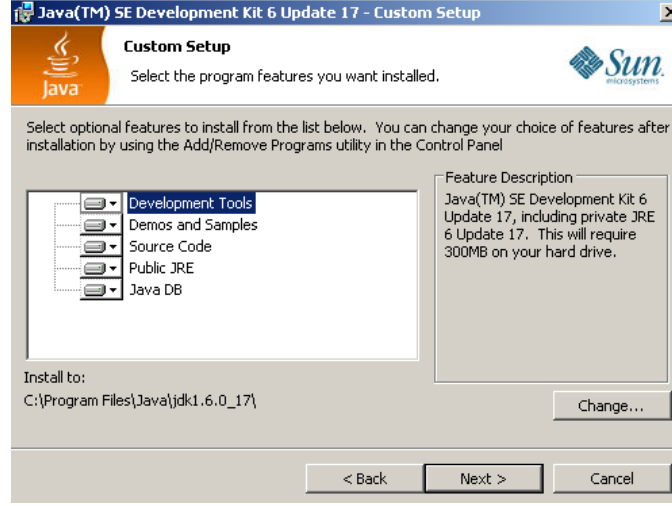
- Yazılımcı java kodunu yazar.
- Java derleyicisi ile java kodları derlendikten sonra 'bytecode' adı verilen bir tür makine kodu ortaya çıkar.
- Düzlem bağımsızlığını sağlayan şey bytecode'dur. Çünkü bir kere bytecode oluştuktan sonra yazılım sanal makine içeren tüm işletim sistemlerinde çalışabilir.
- Bu bytecode Java Virtual Machine (Java Sanal Makinesi) tarafından adım adım işletilir.

Java ile program geliştirebilmek için önce java geliştirme sisteminin (Java Development Kit - JDK) internetten indirilmesi gereklidir. JDK ücretsiz bir yazılımdır. <http://java.sun.com/javase/downloads/index.jsp> adresinden JDK'nın son versiyonu indirilebilir.



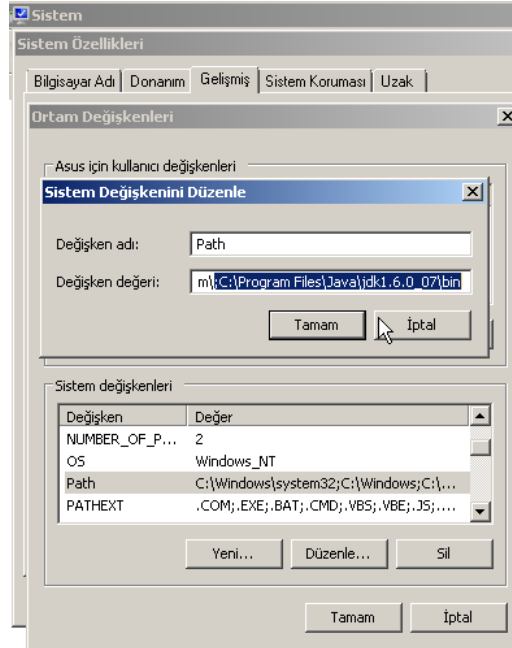
Şekil VII.18 Java Geliştirme Sisteminin İndirilmesi

Dosyanın indirilmesi bittikten sonra kurulum için çalıştırılmalıdır. Ekrandaki yönergeleri izlenerek kurulum işlemini tamamlanır.



Şekil VII.19 Java Geliştirme Sisteminin Kurulumu

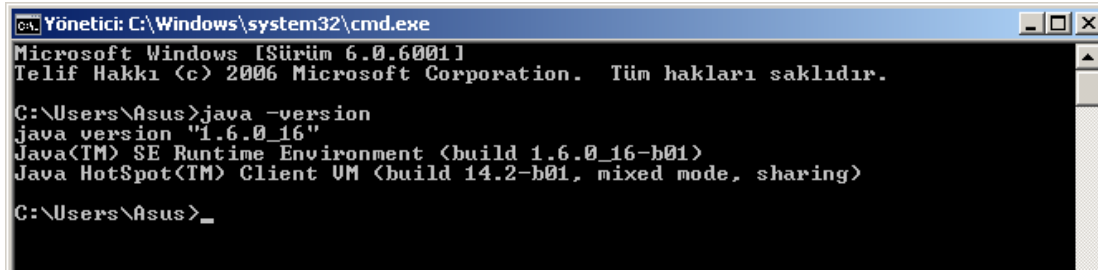
Kurulum tamamlandıktan sonra Denetim Masası (Control Panel)'dan Sistem (System) penceresi açılarak Gelişmiş Sistem Ayarları (Advanced) sekmesi açılmalıdır. Açılan pencerede Çevresel Değişkenler (Environment Variables) düğmesine tıklanmalıdır. Sistem Değişkenleri (System Variables) sekmesindeki Path satırında Düzenle (Edit) seçeneğini seçilerek değişiklik ekranındaki satırın sonuna indirilen sürüm ismine göre `;C:\Program Files\Java\jdk1.6.0_07\bin` ifadesi eklenmelidir. Eğer JDK farklı bir konuma yüklenirse bu yol konuma göre değiştirilmelidir.



Şekil VII.20 Java Geliştirme Sisteminde Sistem Yolu Tanımlama

JDK yükleme işlemi tamamlandıktan sonra kurulum testi yapılabilir. Bunun için Başlat (Start) - Çalıştır (Run) penceresine `cmd` ifadesi yazılmalıdır. Açılan

konsol ekranında `java -version` komutu yazılarak java version testi yapılmış olmaktadır.



```
Yönetici: C:\Windows\system32\cmd.exe
Microsoft Windows [Sürüm 6.0.6001]
Telif Hakkı (c) 2006 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\Asus>java -version
java version "1.6.0_16"
Java(TM) SE Runtime Environment (build 1.6.0_16-b01)
Java HotSpot(TM) Client VM (build 14.2-b01, mixed mode, sharing)

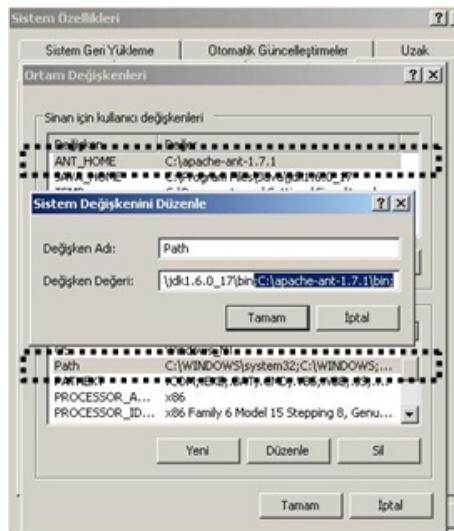
C:\Users\Asus>_
```

Şekil VII.21 JDK Kurulum Testi

VII.6.2. Ant Aracı

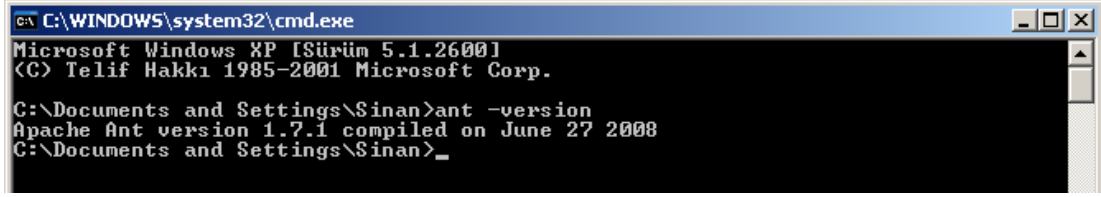
Ant adı İngilizce "Another Neat Tool" (Bir Başka Harika Araç) söz öbeğinin kısaltmasıdır. Apache Ant projesi, bir java projesi yapılırken bir çok işlemi birleştirmeyi, bir sıra halinde yapmayı sağlayan faydalı bir araçtır. Apache Ant, yazılım inşası süreçlerini otomatikleştirmektedir. Örneğin java projesini derleme, JAR veya WAR dosyalarını oluşturma ve bunları dağıtma (deploy) gibi bir çok işlem otomatik olarak yapılabilir. Ant'ın inşa süreci ve bağımlılık tanımları için XML kullanılmaktadır.

Ant paketi, <http://ant.apache.org> adresinden indirilebilmektedir. Apache Ant paketini indirilip bir dizine açıldıktan sonra ortam değişkenlerinin tanımlanması yapılmalıdır. ANT_HOME kullanıcı değişken değerine kurulum dizin yolu verilmelidir. Her dizinde çalışabilmek için Ant kurulum dizinindeki bin dizininin yolunu path sistem değişkenine eklemek gerekmektedir. Örneğin ;C:\apache-ant-1.7.1\bin ifadesi sürüm bilgisine göre Şekil VII.22'de olduğu girilmelidir.



Şekil VII.22 Ant Aracı Sistem Yolu Tanımlama

Kurulumun başarılı olup olmadığını test etmek için konsolda *ant -version* komutu yazılmalıdır.

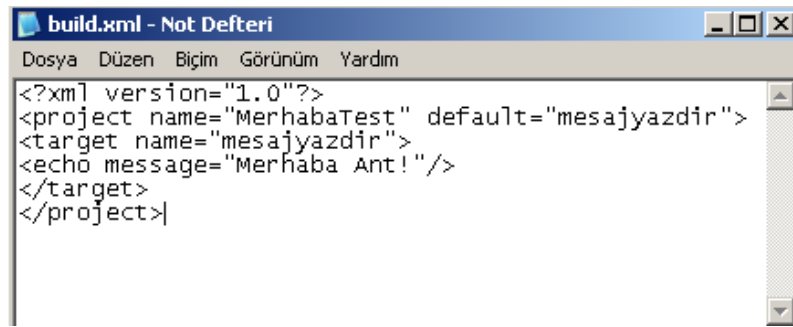


```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Sürüm 5.1.2600]
(C) Telif Hakkı 1985-2001 Microsoft Corp.

C:\Documents and Settings\Sinan>ant -version
Apache Ant version 1.7.1 compiled on June 27 2008
C:\Documents and Settings\Sinan>_
```

Şekil VII.23 Ant Aracı Kurulum Testi

Ant aracıyla yapılacak işlemler bir XML dosyasında tanımlanmalıdır. Bu dosyaya genelde build.xml ismi verilmektedir. Şekil VII.24’de basit bir ant dosya yapısı verilmektedir.

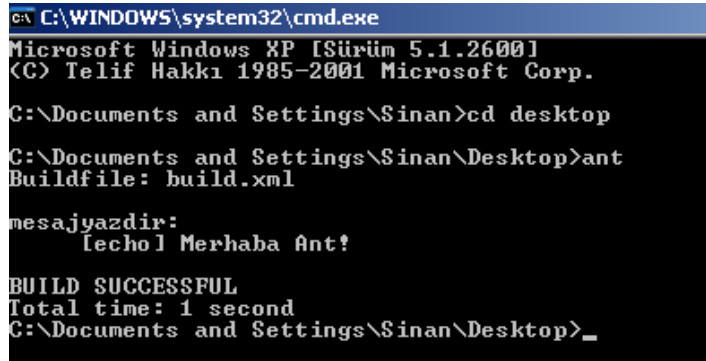


```
build.xml - Not Defteri
Dosya Düzen Biçim Görünüm Yardım

<?xml version="1.0"?>
<project name="MerhabaTest" default="mesajyazdir">
<target name="mesajyazdir">
<echo message="Merhaba Ant!"/>
</target>
</project>
```

Şekil VII.24 Basit Bir Ant Dosya Yapısı

Ant aracının build.xml’de belirtilen bilgilere göre gerekli işlemleri yapması için konsol ekranında dosya dizini içinde *ant* veya *ant build.xml* komut satırı yazılmalıdır.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Sürüm 5.1.2600]
(C) Telif Hakkı 1985-2001 Microsoft Corp.

C:\Documents and Settings\Sinan>cd desktop
C:\Documents and Settings\Sinan\Desktop>ant
Buildfile: build.xml

mesajyazdir:
[echo] Merhaba Ant!

BUILD SUCCESSFUL
Total time: 1 second
C:\Documents and Settings\Sinan\Desktop>_
```

Şekil VII.25 Ant Aracı İle İşlem Yapma

Ant aracında çeşitli görevlere sahip bir çok element bulunmaktadır.

<project>: Project elementi en üst seviyede yer alır. Project elementinde üç özellik bulunur.

name : Proje ismi belirtilir.

default : XML dosyasında target elementine yönlendirme yapılır.

basedir : Kaynak olan dizin adı belirtilir.

<target> : Görevleri birbirine bağlayan bir elementtir. <Target> elementleri *depends* özelliği ile birbirine bağlı duruma getirilerek çalıştırılabilir.

<javac> : Bir dizinde yer alan sınıfları derlemek için kullanılabilir.

<java> : Bir sınıfı çalıştırabilir.

<mkdir> : Bir dizin yaratmak için kullanılabilir.

<delete> : Dizin veya dosyaları silebilir.

<copy> : Bir dosyayı veya dizini bir yerden başka yere kopyalayabilir.

<move> : Bir dosyayı veya dizini bir yerden başka yere taşıyabilir.

<jar> : jar,war,ear dosyaları hazırlanabilir.

<ftp> : Bir dosyayı başka bilgisayara aktarmak için kullanılabilir.

<echo> : Kullanıcıya bilgi vermek için kullanılır.

Zaman çizelgesi projesinde, build.xml ant dosya yapısında ilk XML elementi <project> elementidir. Sonraki satırlarda iç değişkenlerin tanımlanması <properties> elementi ile yapılmaktadır. İç değişken tanımlamalarının çoğu kaynak ve hedef dizinlerle ilgilidir. <path> elementi ile JAR dosyalarının yer aldığı classpath oluşturulur. Derlenen sınıf dosyaları “*build/zcproje/WEB-INF/classes/*” dizini altında yer almaktadır.

Tablo VII.9 Ant Build.xml Dosyası

```
<project name="zcproje" basedir="." default="build">
  <property name="appname" value="zcproje" />
  <property name="lib.dir" value="lib" />
  <property name="war.dir" value="build/zcproje" />
  <property name="war.file" value="{dist.dir}/{appname}.war" />
  <property name="webinf.dir" value="{war.dir}/WEB-INF" />
  <path id="master-classpath" description="CLASSPATH olusturma">
    <fileset dir="{lib.dir}">
      <include name="*.jar" />
    </fileset>
    <pathelement location="build/zcproje/WEB-INF/classes/" />
  </path>
  <target name="init" description="build komutu icin dizin olusturulmasi">
    <mkdir dir="{class.dir}" />
    <mkdir dir="{libs.dir}" />
    <mkdir dir="{jsp.dir}" />
  </target>
```

“*build/zcproje/WEB-INF/classes/*” altına tüm java dosyaları derlenerek WAR dosyası için ayarlamalar yapılmalıdır.

Tablo VII.10 Ant Build.xml Dosyası

```
<target name="compile" description="WAR dizini için .java dosyalarının
derlenmesi">
  <javac srcdir="${src.dir}" destdir="${class.dir}" debug="true"
    failonerror="true" classpathref="master-classpath" />
</target>
<war destfile="${war.file}" webxml="${src.dir}/conf/web.xml">
  <fileset dir="${war.dir}">
    <include name="**/*.*" />
    <exclude name="**/web.xml" />
    <exclude name="**/test/*.class" />
    <exclude name="**/*mock*.jar" />
  </fileset>
</war>
```

VII.6.3. JUnit Aracı

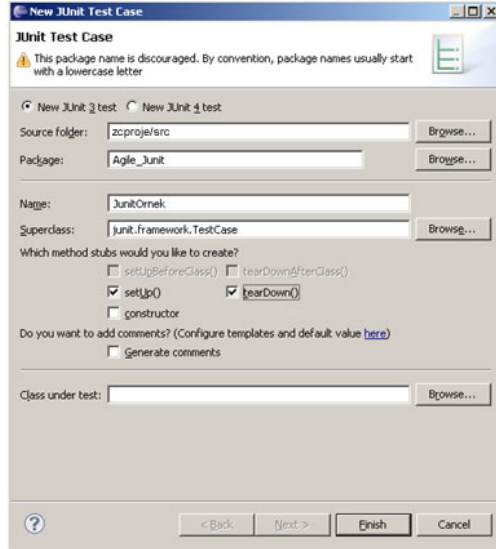
Java ile geliştirilen uygulamalarda bulunan sınıfları test etmek amacıyla oluşturulmuş açık kaynaklı bir araçtır. Test sonuçlarının görüntülenmesini sağlayan bir arayüze sahiptir. Test işlemlerini en iyi şekilde yapmanın yolu ise, testi uygulamanın en küçük parçaları için yapmaktır. Birim testi, uygulamadaki belirli bir sınıfın istediğimiz metotlarının doğru çalışıp çalışmadığını test etmek için kullanılmaktadır. Çalışmasını test etmek istediğimiz sınıf için bir test sınıfı yazılmalıdır. Bu sınıfın içinde çeşitli *Assert* metotları ile test etmek istediğimiz sınıfın değişkenlerini, fonksiyonlarını ve geri dönüş değerlerini test ederiz.

Kullanabileceğimiz bazı karşılaştırma metotları:

- assertEquals(A,B) : nesneler eşit mi?
- assertFalse(boolean şart) : verilen şart yanlış mı?
- assertTrue(boolean şart) : verilen şart doğru mu?
- assertNotNull() : nesne null değil mi?
- assertNull() : nesne null mu?
- assertEquals(A,B) : ikisi aynı nesneyi gösteriyor mu ?
- fail() : bir testin mesaj belirtmeden başarısız olmasını sağlar.

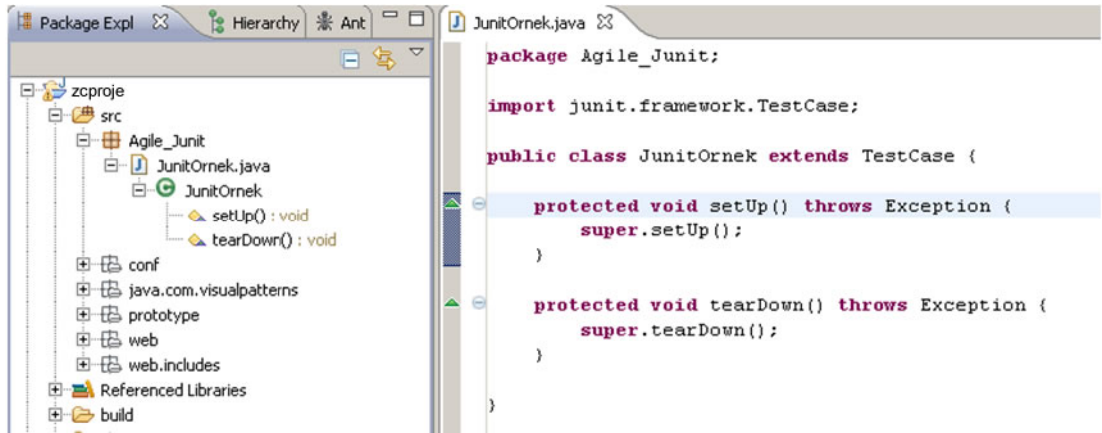
Açık kaynak yazılımı olan JUnit, <http://junit.org/> internet adresinden ücretsiz olarak indirilebilir ve kullanılabilir. Eclipse programı yardımıyla Junit test kontrolü yapılabilmektedir. Junit aracı, standart yaratma ve temizleme metotları içermektedir (setUp ve tearDown). Sonuçları kontrol etmeyi kolaylaştıran metotlara sahiptir. Ant aracıyla ve Eclipse gibi bütünleşmiş geliştirme ortamları (Integrated Development

Environment - IDE) ile kullanılabilir. Eclipse programında File-New-Junit Test Case seçenekleri ile açılan pencere yapısı Şekil VII.26’da gösterilmektedir. Finish düğmesine basılarak java projesine junit testleri eklenir.



Şekil VII.26 Eclipse ile Junit Testi

Her test metoduna girmeden önce setUp(), test bittikten sonra tearDown() metodu çalışmaktadır. setUp() metodu içinde test için gerekli ortam oluşturulmaktadır. tearDown() metodunda test için oluşturulmuş kaynaklar yok edilerek test sonrası temizlik yapılmaktadır. Bu iki metot arasına tüm testler yazılmalıdır.



Şekil VII.27 Eclipse ile Junit Testi

Örnek bir test uygulaması için Şekil VII.28’de yer alan kodları yeniden düzenlenmelidir. “testikivebesincarpimionmu” adlı metodu eklenerek test için gerekli olan “nesnem” adlı sınıf oluşturulmalıdır. Test sonucunda kırmızı ve yeşil renklerle durum belirtilmektedir.

```
JUnitOrnek.java  JUnitTest.java

package Agile_Junit;

import junit.framework.TestCase;

public class JUnitOrnek extends TestCase {

    JUnitTest nesnem;

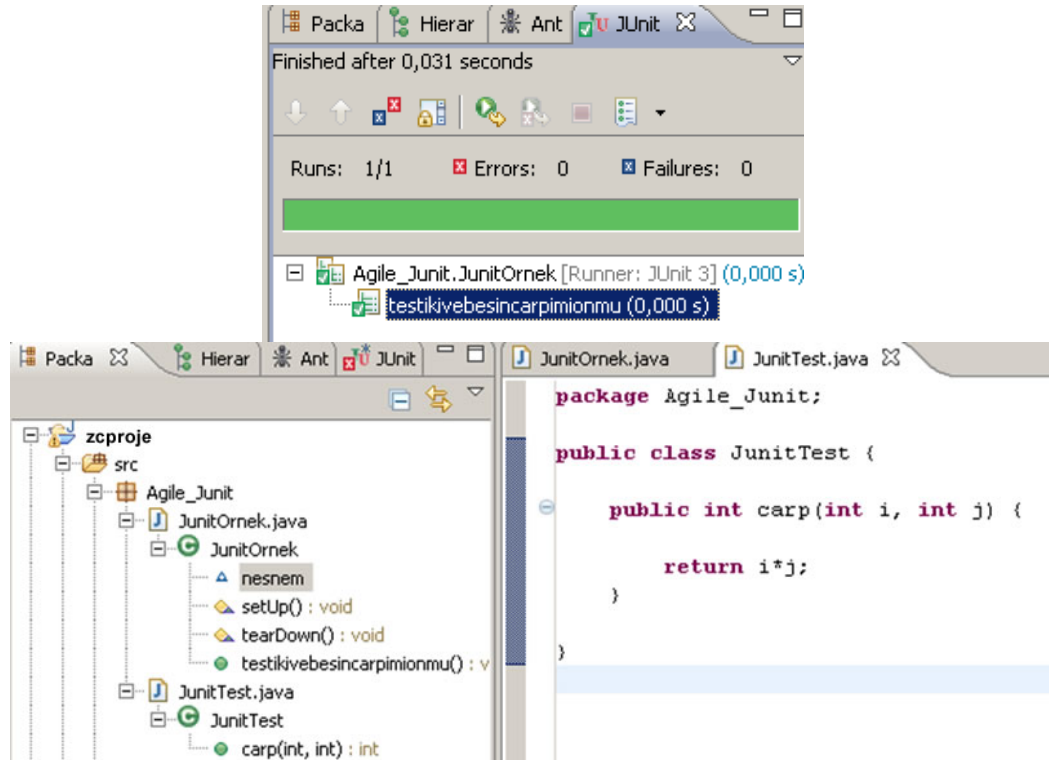
    protected void setUp() throws Exception {
        nesnem=new JUnitTest();
        super.setUp();
    }

    public void testikivebesincarpimionmu() {
        assertEquals("2*5=10 mu?",nesnem.carp(2,5),20);
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

}
```

Şekil VII.28 Eclipse ile Junit Testi



Şekil VII.29 Eclipse ile Junit Testi

VII.6.4. Eclipse IDE

Eclipse, açık kaynak kodlu bir tümleşik geliştirme ortamıdır (IDE). Ana odak noktası Java ve Java ile ilişkili teknolojiler olsa da esnek yapısı sayesinde C ve Python gibi farklı diller için de kullanılmaktadır. Hızlı arayüzü, şık görünümü ve çok kuvvetli özellikleriyle kısa zamanda Java geliştiricileri arasında en popüler geliştirme ortamı olmuştur. Ayrıca sunulan eklentiler ile işlevleri geliştirilerek birçok alanda kullanılabilir.

Java kodları, not defteri gibi basit bir metin düzenleyici programla yazılabilmektedir. Ama bu yaklaşım zaman kaybına neden olup büyük projelerin yönetilebilmesini zorlaştırmaktadır. Bu nedenle Eclipse gibi tümleşik geliştirme ortamları (Integrated Development Environment, IDE) kullanılmaktadır.

IDE'lerin en önemli avantajları, kodları organize bir şekilde tutabilmeyi sağlaması, intelli-sense özelliği sayesinde kodları hatırlatması, classpath ayarlamasını basitçe yapabilmesi, kodları renklendirmesi, kodların girinti-çıkıntı ayarlarını düzenlemesi olarak sayılabilmektedir. Program ücretsiz olup <http://www.eclipse.org/> adresinden indirilebilmektedir. Şekil VII.30'da eclipse download sayfası gösterilmektedir.

Eclipse Downloads

Eclipse Packages

Projects

Development Builds

Galileo Packages (based on Eclipse 3.5 SR1) - [Compare Packages](#) Log in

**Eclipse IDE for Java EE Developers (189 MB)**
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn and others. [More...](#)
Downloads: 1,990,935

Windows 32bit
Mac Carbon 32bit
Mac Cocoa 32bit 64bit
Linux 32bit 64bit

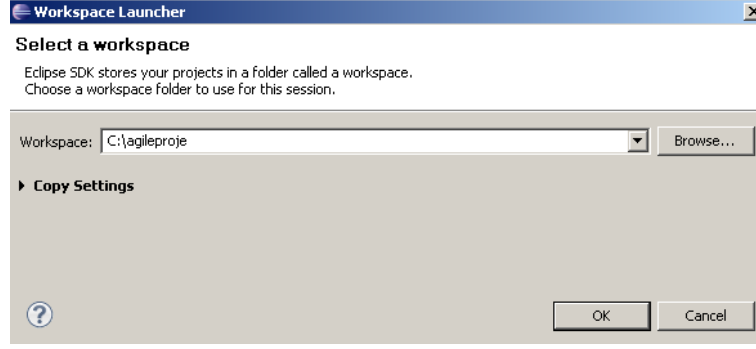
**Eclipse IDE for Java Developers (92 MB)**
The essential tools for any Java developer, including a Java IDE, a CVS client, XML Editor and Mylyn. [More...](#)
Downloads: 983,737

Windows 32bit
Mac Carbon 32bit
Mac Cocoa 32bit 64bit
Linux 32bit 64bit

You will need Eclipse (Java) provided under the Eclipse Foundation specified.

Şekil VII.30 Eclipse IDE Programını İndirme

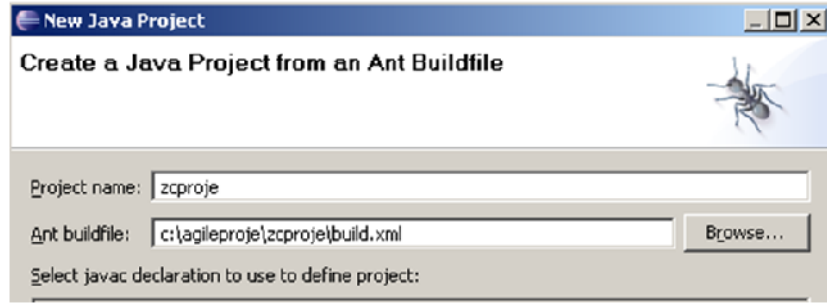
Eclipse dosyası internetten indirildikten sonra kurulumun yapıldığı dizine gidilerek eclipse.exe dosyası çalıştırılmalıdır. Karşımıza gelen ekran görüntüsünde Şekil VII.31'de gösterilen diyalog penceresi açılacaktır. Bu pencerede çalışma alanı adresi girilmelidir. Örneğin C:\agileproje gibi. Proje dosyaları bu dizin altında saklanacaktır.



Şekil VII.31 Eclipse IDE Çalışma Alanı

Zaman Çizelgesi Ayarları

Eclipse IDE programının genel yapısından sonra daha ileri aşamada zaman çizelgesi ayarlarının yapılması gereklidir. Mevcut olan build.xml dosyası kullanılarak Eclipse projesi oluşturulabilir. *File* menüsünden *New-Other* seçenekleri açılan pencerede “Java Project from an Existing Ant Buildfile” ifade seçilmelidir. Şekil VII.32’de gösterilen biçimde doldurularak sonraki aşamalara geçilmelidir. İşlem tamamlandığında zcproje adlı projemizin dizin yapısı otomatik olarak oluşacaktır.

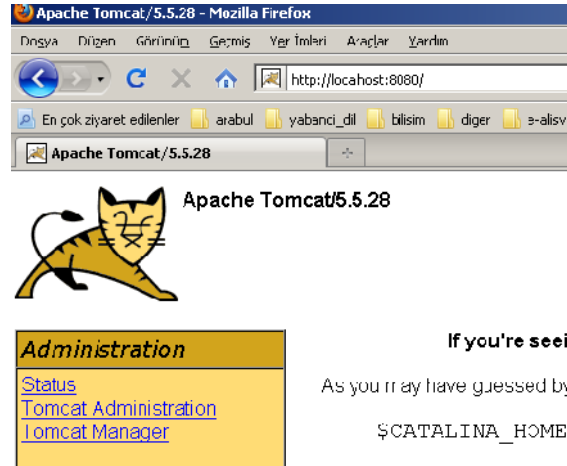


Şekil VII.32 Eclipse IDE Ant Build.xml Dosyası

VII.6.5. Apache Tomcat

Tomcat, Java Servlet ve Java Server Pages teknolojileri için resmi kodlama referansı olarak kullanılan bir servlet barındırıcısıdır. Java Servlet ve Java Server Pages belirtimleri SUN tarafından Java Community Process altında geliştirilmiştir. <http://tomcat.apache.org/> adresinde indirilebilmektedir. İndirilen dosya bir dizine indirilip işletim sistemine kurulur.

Tomcat uygulama sunucusunu test etmek için açılan herhangi bir web tarayıcıda adres satırına <http://localhost:8080> yazarak yapabilirsiniz. Tomcat’in açılış ekranını Şekil VII.33’de gösterilmektedir.



Şekil VII.33 Tomcat Açılış Ekranı

Tomcat dizin yapısı Tablo VII.11’de verilmektedir. Tomcat uygulama sunucusunda, localhost üzerinde bulunan anasayfada status ve manager alanlarına, tomcat-users.xml dosyasında yetki vereceğiniz kullanıcılar (users) girebilmektedir.

Tablo VII.11 Tomcat Uygulama Sunucusu Dizin Yapısı

Dizin Adı	Görevi
bin	Uygulama ve script dosyaları bulunur. Tomcat buradan çalıştırılır.
conf	Çalışma ayar dosyaları bulunur.
lib	Java kütüphaneleri burada bulunur.
logs	Log dosyaları bulunur.
temp	Geçici dosyalar burada bulunur.
webapps	Oluşturacağımız uygulamaları standart olarak bu klasör altına bulunur.
work	JSP kodlarının Servlet kodları haline getirilip burada bulundurulur.

VII.6.6. Spring MVC

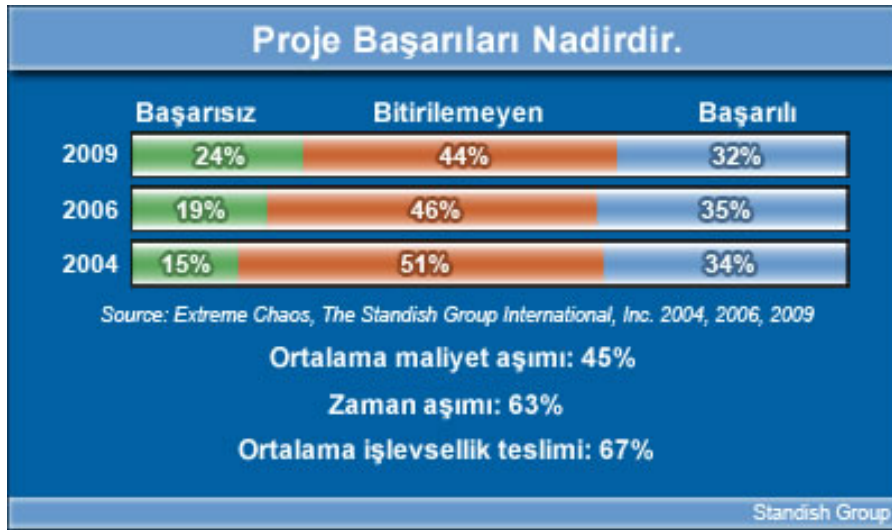
Spring, J2EE (Java 2 Enterprise Edition) uygulamalarını geliştirme süreçlerini kolaylaştırmayı amaçlayan bir uygulama geliştirme çatısıdır. Spring, oldukça kullanışlı birçok özelliği bünyesinde barındırmaktadır. Bu özellikler, çatıyı oluşturan yedi farklı birim tarafından sunulmaktadır. Bu çatılardan birisi olan Spring MVC (Model-View-Control/Model-Görünüm-Denetim) birimi, web uygulamaları için Model-Görünüm-Denetim gerçekleştirmesini sağlamaktadır.

Proje için, <http://www.springsource.org/> adresinden indirilen *spring.jar* dosyası zcrproje/lib dizini içine kopyalanmalıdır.

BÖLÜM VIII

SONUÇLAR VE TARTIŞMA

Yazılım projelerinin zamanında, uygun bütçe ile gereksinimleri tam olarak karşılayacak şekilde teslim edilememesi yazılım endüstrisini süreç iyileştirme çalışmalarına yöneltmiştir. Standish Grup tarafından hazırlanan yazılım projeleri ile ilgili araştırma sonuçları günümüzde uygulanan birçok projenin başarıyla tamamlanamadığı göstermektedir.



Şekil VIII.1 Standish Grup - Yazılım Projeleri Başarı Tarihçesi [77]

Yapılan tez çalışmasında öncelikle geleneksel süreçler incelenerek proje süreçlerine etkileri belirlenmiştir. Geleneksel süreçlerin değişime karşı yeterli olmadığı ve müşteri gereksinimlerini karşılayamadığı anlaşılmıştır. Bu süreçlerde bulunan doğrusal yapıda (gereksinim, tasarım, programlama, entegrasyon, teslimat, bakım) başlangıçta yapılan değerlendirilmelerin proje süresince yapılmaması proje başarısını olumsuz yönde etkilemektedir.

Çevik yazılım geliştirme bölümünde, çevik değer ve ilkeleri incelenerek proje başarısına etkileri belirlenmiştir. Çevik yazılım geliştirme süreçlerinin, geleneksel süreçlerin aksine tekrarlanan yazılım geliştirme metodu temel alınarak geliştirilmiş, sık aralıklarla, parça parça yazılım teslimatını ve değişikliği teşvik eden bir yapıya sahip olduğu anlaşılmıştır.

Ekstrem programlama bölümünde, en çok kullanılan çevik süreçlerden olan ekstrem programlamanın değer, ilke ve uygulamaları incelenerek sürece olan etkileri belirlenmiştir. Ekstrem programlamada müşteri ve gereksinimlerin ayrı bir önemi olduğu dikkat çekmiştir. Projelerde dokümantasyon işlemlerinin az seviyeye çekildiği ve yazılım metotlarına yoğunlaşıldığı anlaşılmıştır.

Çevik model tabanlı geliştirmede, çevik süreçlerle model tabanlı geliştirme yapısı incelenmiştir. Bu yaklaşım, yazılım geliştirmede kapsamlı modellerin önceden kaynak kodu yazılarak oluşturulmasıdır. Zaman içinde projelerde, artan değişikliğe yanıt imkanı verilmesini sağlamaktadır. Verim sağlamak için geliştirme araçlarınızda çok yönlü modellere ihtiyaç olabilmektedir. Modelleme yaparken sadeliğin önemi ve çalışırken taleplerin zaman içinde değişiminden dolayı değişime bağlı kalmak gerekmektedir. Değişime yanıt vermede sürece yönelik bir modelleme yapılabilmektedir.

Tez çalışmasında örnek proje olarak ekstrem programlama ve çevik model tabanlı geliştirme ile çevrimiçi zaman çizelgesi uygulaması yapılmıştır. Bu projeye ekstrem programlama ve ÇMTG değer, ilke ve uygulamaları aktararak yazılım geliştirme sürecinde ilk sürüm oluşturma evreleri incelemiştir. Proje takımının sürüm oluşturma sırasında yapabileceği etkinlikler sıralı olarak anlatılmıştır.

BÖLÜM IX

SON DEĞERLENDİRMELER VE ÖNERİLER

Yazılımda başarıyı yakalamanın temel yolu, dünyada belirlenmiş yazılım süreçlerini uygulamak, doğru teknolojileri verimli kullanmak ve yeni teknolojileri olabildiğince hızlı öğrenip olumlu ve olumsuz noktalarıyla değerlendirmekten geçmektedir.

Geleneksel yazılım süreçleri nedeniyle projelerde gereksinimler yeterince anlaşılmadan üretime geçilmektedir. Bu durum sonradan meydana gelen maliyet, iletişim ve diğer kayıplarla hem müşteri hem de yazılım firmalarına zarar vermektedir. Yazılım endüstrisinde proje başarısızlıklarının devam etmesi yeni çözümlerinde ortaya çıkmasına neden olmuştur. Proje başarısızlıklarına bir çözüm olması amacıyla çevik yazılım geliştirme ortaya çıkmıştır. Çevik süreçinde, yazılım yöntemleri kullanılarak müşteriye değer veren, yazılım ihtiyaçlarını netleştiren ve küçük adımlarla projeyi teslim etme yaklaşımı bulunmaktadır.

Çevik yazılım geliştirme, projelere başarı garanti veren kurtarıcı bir çözüm değildir. Mutlaka kullanıldıkları proje yapılarına göre düzenlenmeleri gereklidir. Bu süreçlerle, yazılım geliştirme dışında üretim akışında iyileştirilmektedir. Çevik yazılım geliştirmede temel amaç proje süreçlerini daha iyi yapabilmektir.

KAYNAKLAR

- [1] Leonhardt, D.: “*John Tukey, 85, Statistician; Coined the Word 'Software'*”, The New York Times, **(2000)**
<http://www.nytimes.com/2000/07/28/us/john-tukey-85-statistician-coined-the-word-software.html> (01.09.2009).
- [2] Hally, M.: “*Electronic brains/Stories from the dawn of the computer age*”, British Broadcasting Corporation and Granta Books, **(2005)** 79
- [3] <http://tr.wikipedia.org/wiki/Yazılım> (01.09.2009).
- [4] http://en.wikipedia.org/wiki/History_of_software_engineering (01.09.2009).
- [5] Brooks, F.P.: “*No Silver Bullet - Essence and Accident in Software Engineering*”, Proceedings of the IFIP Tenth World Computing Conference, **(1986)** 1069-1076
- [6] Brooks, F.P.: “*No Silver Bullet - Essence and Accidents of Software Engineering*”, IEEE Computer Magazine, **(1987)** 10-19
- [7] Brooks, F.P.: “*No Silver Bullet - Essence and Accident*”, Anniversary Edition with four new chapters edition, Addison-Wesley, **(1995)**
- [8] Abran, A.; Moore, J.W.; Bourque, P.; Dupuis, R.: “*Guide to the Software Engineering Body of Knowledge - 2004 Version*”, IEEE Computer Society, <http://www.swebok.org>, **(2004)**
- [9] Sommerville, I.: “*Software Engineering*”, Addison-Wesley Publishers Limited, 8th edition, Chapter 1, **(2006)**
- [10] Kan, S.H.: “*Metrics and Models in Software Quality Engineering*”, Addison-Wesley Publishers Limited, Second Edition, **(2002)**
- [11] Jones, C.: “*Critical Problems in Software Measurement*”, Software Productivity Research, **(1992)**
- [12] <http://xprogramming.com/xpmag/whatisxp> (20.11.2009)
- [13] <http://www.selfishprogramming.com/wp-content/uploads/2009/08/pm-release-planning.jpg> (25.12.2009)
- [14] Borysowich, C.: “*Software Quality Factors Checklist*”, <http://it.toolbox.com/blogs/enterprise-solutions/software-quality-factors-checklist-19562> (15.09.2009).

- [15] DRM Associates, "New Product Development Glossary",
<http://www.npd-solutions.com/glossary.html#s> (17.09.2009)
- [16] <http://agile.outformations.com/EAT/AgileTrainingStoryCardExample.png>
- [17] Giray, G.; Ünalır, M.O.: "Yazılım Mühendisliğinde Ontolojilerin Kullanımı",
Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi
http://www.emo.org.tr/ekler/0b53bf73332d20a_ek.pdf (25.09.2009)
- [18] Riehle, D.: "A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn from Each Other", SKYVA International, (2000)
<http://dirkriehle.com/computer-science/research/2000/xp-2000.pdf>
- [19] Wiegers, K.E.: "Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle", Microsoft Press, (2003)
- [20] Kaner, C.: "Exploratory Testing", Quality Assurance Institute Worldwide Annual Software Testing Conference, Florida Institute of Technology, (2006)
<http://www.kaner.com/pdfs/ETatQAI.pdf> (05.10.2009)
- [21] <http://www.testroot.com/2009/01/05/konfigurasyon-yonetimi-nedir/>
(08.10.2009)
- [22] Balci, O.; Gilley, W.S.; Adams, R.J.; Tunar, E.; Barnette N.D.: "Animations to Assist Learning Some Key Computer Science Topics", Department of Computer Science, Virginia Tech
<http://courses.cs.vt.edu/~csonline/SE/Lessons/LifeCycle/> (12.10.2009)
- [23] Schach, S.R.: "Software Engineering", McGraw-Hill (1999) 11
- [24] Britton, C.; Doake, J.: "Software system development: a gentle introduction", McGraw-Hill, (1993)
- [25] Elliott, G.: "Global Business Information Technology: An Integrated Systems Approach", Pearson Education, (2004) 87
- [26] "Selecting A Development Approach",
<http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf> (17.10.2009)
- [27] Yazılım geliştirme metodolojilerinde kullanılan üç temel model
http://en.wikipedia.org/wiki/File:Three_software_development_patterns_mashed_together.svg (25.10.2009)

- [28] Madachy, R.J.: “*Software Process Dynamics*”, Wiley-Interscience, by IEEE Press **(2008)** 29-40
- [29] Balci, O.; Gilley, W.S.; Adams, R.J.; Tunar, E.; Barnette N.D.: “*Animations to Assist Learning Some Key Computer Science Topics*”, Department of Computer Science, Virginia Tech
<http://courses.cs.vt.edu/~csonline/SE/Lessons/Waterfall/index.html>
 (25.10.2009)
- [30] http://qastation.files.wordpress.com/2008/04/incremental_model1.jpg
 (27.10.2009)
- [31] Cockburn, A.: “*Agile Software Development*”, Agile Software Development Series, Addison-Wesley Publishers Limited, **(2001)**
- [32] <http://qastation.wordpress.com/2008/04/28/software-development-lifecycles-part-6/> (09.11.2009)
- [33] <http://www.robabdul.com/Data-Management-System-Software-Development-Cycle.asp> (09.11.2009)
- [34] <http://qastation.wordpress.com/2008/04/25/software-development-life-cycles-part-3/> (09.11.2009)
- [35] <http://qastation.wordpress.com/2008/04/26/software-development-life-cycles-part-4/> (09.11.2009)
- [36] <http://www.aztecsoft.com/downloads/agilemethology.pdf> (15.11.2009)
- [37] <http://www.robabdul.com/images//Exploratory%20Development%20Model.gif>
 (18.11.2009)
- [38] http://en.wikipedia.org/wiki/File:Iterative_development_model_V2.jpg
 (18.11.2009)
- [39] <http://en.wikipedia.org/wiki/File:Development-iterative.gif> (18.11.2009)
- [40] http://en.wikipedia.org/wiki/File:Spiral_model_%28Boehm,_1988%29.png
 (20.11.2009)
- [41] http://qastation.files.wordpress.com/2008/04/prototype_model.jpg
 (20.11.2009)
- [42] Kossiakoff, A.; Sweet, W.N.: “*Systems engineering: principles and practices*”, John Wiley and Sons Inc., **(2003)**
- [43] <http://www.thefreedictionary.com/agile> (25.11.2009)
- [44] <http://www.sozluk.net/index.php?word=agile> (25.11.2009)

- [45] Cockburn, A.: “*Agile Software Development: The Cooperative Game*”, Addison-Wesley Publishers Limited, (2006) 222
- [46] <http://www.versionone.com/Resources/AgileDevelopment.asp> (01.12.2009)
- [47] Highsmith, J.: “*Agile Software Development Ecosystems*”, , Addison-Wesley Publishers Limited, (2002)
- [48] Palmer, S.R.; Felsing, J.M.: “*A Practical Guide to Feature-Driven Development*”, Prentice-Hall Inc., (2002)
- [49] Smith, G.; Sidky, A.: “*Becoming Agile: ...in an imperfect world*”, Manning Publications, (2009) 6
- [50] Ambler, S.W.: “*Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*”, John Wiley and Sons Inc., (2002)
- [51] http://en.wikipedia.org/wiki/Agile_Manifesto (05.12.2009)
- [52] Shore, J.; Warden, S.: “*The Art of Agile Development*”, Oreilly, (2008) 10
- [53] <http://www.versionone.com/Resources/AgileBenefits.asp> (06.12.2009)
- [54] http://www.prime.com.tr/agile/agile_primeteknoloji.pdf (06.12.2009)
- [55] Leffingwell, D.: “*Scaling software agility, Best Practices for large enterprises*”, Pearson Education, (2007)
- [56] <http://www.inveon.com.tr/Metodoloji/Cevik-Metotlar> (09.12.2009)
- [57] Acar, Ö.: “*Extreme Programming*”, Pusula Yayıncılık (2009)
- [58] Larman, C.: “*Agile and Iterative Development: A Manager’s Guide*”, Pearson Education, (2003)
- [59] Beck, K.: “*Extreme Programming Explained: Embrace Change*”, Addison-Wesley Publishers Limited, (2004)
- [60] Cocburn, A.; Highsmith, J.; Johansen, K.; Jones, M.: “*Crystal Methodologies*”, (2004)
- [61] Leffingwell, D.; Muirhead, D.: “*Tactical Management of Agile Development: Achieving Competitive Advantage*”, Rally Software Development Corporation Whitepaper, (2004)
www.rallydev.com/documents/rally_whitepaper.pdf (13.12.2009)
- [62] Everette, R.K.: “*A software development process, A different approach to software design*”, (2002)
<http://www.agilealliance.com/system/article/file/1099/file.pdf> (20.12.2009)
- [63] <http://www.versionone.com/Resources/AgileMethodologies.asp> (13.12.2009)

- [64] “Scrum It’s about common sense”, Control Chaos
<http://www.controlchaos.com/about/how.php> (15.12.2009)
- [65] Highsmith, J.: “*What is agile Software Development?*”, The journal of Defense Software Engineering, Ekim (2002)
- [66] Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J.: “*Agile Software Development Methods, Review and Analysis*”, VTT Publications, (2002)
- [67] Cohn, M.: “*Selecting an agile process, comparing the leading alternatives*”, Mountain Goat Software, (2002)
- [68] Fowler, M.: “*The new methodology*”, (2005)
<http://www.martinfowler.com/articles/newMethodology.html#N101F8>
(18.12.2009)
- [69] <http://www.bilgininadresi.net/Madde/9270/Agile-Modelleme-De%C4%9Ferleri> (20.12.2009)
- [70] http://www.tekool.net/blogfiles/printable-agile-planning-poker/agile_planning_poker_front.png (25.12.2009)
- [71] <http://weblogs.asp.net/blogs/arturtrosin/AgilePractical/1.JPG> (25.12.2009)
- [72] http://www.mountaingoatsoftware.com/system/hidden_asset/file/29
(25.12.2009)
- [73] Ambler, S.W.: “*Agile Scaling Factors*” (2009)
<https://www.ibm.com/developerworks/mydeveloperworks/ScottAmbler>
(25.12.2009)
- [74] Ambler, S.W.: “*Evidence that Agile Software Development Scales*” (2009)
<http://www.enterpriseunifiedprocess.com/essays/agileScales.html> (25.12.2009)
- [75] Ambler, S.W.: “*Effective Practices for Modeling and Documentation*”
<http://www.agilemodeling.com> (30.12.2009)
- [76] <http://www.acm-software.com/Makaleler> (30.12.2009)
- [77] <http://www.talentontarget.com/index.php/2009/08/the-2009-chaos-report-is-project-success-really-that-rare/> (15.01.2010)

EKLER

controller/zclistedenetleyici.java dosya kodlamaları

```
package com.agileproje.zcproje.controller;

import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import com.agileproje.zcproje.model.Kullanici;
import com.agileproje.zcproje.model.ZamancizelgesiYoneticisi;
import com.agileproje.zcproje.util.UygulamaGuvenlikYoneticisi;
import com.agileproje.zcproje.util.ZcprojeJmxBean;

/* ACIKLAMA - Zaman cizelgesi liste ekranı için denetleyiciler
*/
public class zclistedenetleyici implements Controller
{
    private ZamancizelgesiYoneticisi zamancizelgesiYoneticisi;
    private UygulamaGuvenlikYoneticisi uygulamaGuvenlikYoneticisi;
    public static final String MAP_KEY = "zamancizelgeleri";
    private String successView;
    private ZcprojeJmxBean zcprojeJmxBean;

    /** ACIKLAMA
     * ModelAndView de veritabanı nesneleri ile zaman cizelgesi listesinin çağırılması
     * com.agileproje.zcproje.model.Zamancizelgesi ne bakabilirsiniz.
     */
    public ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception
    {
        Kullanici kullanici = (Kullanici) uygulamaGuvenlikYoneticisi
            .getKullanici(request);
        List zamancizelgeleri = zamancizelgesiYoneticisi.getZamancizelgeleri(kullanici
            .getKullaniciId());

        zcprojeJmxBean.setZamancizelgeleriFetched(zcprojeJmxBean.getZamancizelgeleriF
            etched()
            + zamancizelgeleri.size());
    }
}
```

```

        return new ModelAndView(getSuccessView(), MAP_KEY, zamancizelgeleri);
    }

    public ZamancizelgesiYoneticisi getZamancizelgesiYoneticisi()
    { return zamancizelgesiYoneticisi; }

    public void setZamancizelgesiYoneticisi(ZamancizelgesiYoneticisi
zamancizelgesiYoneticisi)
    { this.zamancizelgesiYoneticisi = zamancizelgesiYoneticisi; }

    public UygulamaGuvenlikYoneticisi getUygulamaGuvenlikYoneticisi()
    { return uygulamaGuvenlikYoneticisi; }

    public void setUygulamaGuvenlikYoneticisi(
        UygulamaGuvenlikYoneticisi uygulamaGuvenlikYoneticisi)
    { this.uygulamaGuvenlikYoneticisi = uygulamaGuvenlikYoneticisi; }

    public String getSuccessView()
    { return successView; }

    public void setSuccessView(String successView)
    { this.successView = successView; }

    public ZcprojeJmxBean getzcprojeJmxBean()
    { return zcprojeJmxBean; }

    public void setZcprojeJmxBean(ZcprojeJmxBean zcprojeJmxBean)
    { this.zcprojeJmxBean = zcprojeJmxBean; }
}

```

model/Zamancizelgesi.java dosya kodlamaları

```
package com.agileproje.zcproje.model;
```

```
import java.util.Date;
```

```

public class Zamancizelgesi
{
    private int zamancizelgesiId;
    private int kullaniciId;
    private int dakikaPzt = 0;
    private int dakikaSa = 0;
    private int dakikaCrs = 0;

```

```

private int dakikaPrs = 0;
private int dakikaCm = 0;
private int dakikaCmt = 0;
private int dakikaPa = 0;
private String mevkiKodu;
private String departmanKodu;
private Date bitisSuresiTarihi;
private Departman departman;

public int getZamancizelgesiId()
{ return zamancizelgesiId; }

/** ACIKLAMA
 * zamancizelgesiId ayarlari
 */
public void setZamancizelgesiId(int zamancizelgesiId)
{ this.zamancizelgesiId = zamancizelgesiId; }

public int getKullaniciId()
{ return kullaniciId; }

public void setKullaniciId(int kullaniciId)
{ this.kullaniciId = kullaniciId; }

public int getDakikaPzt()
{ return dakikaPzt; }

public void setDakikaPzt(int dakikaPzt)
{ this.dakikaPzt = dakikaPzt; }

public int getDakikaSa()
{ return dakikaSa; }

public void setDakikaSa(int dakikaSa)
{ this.dakikaSa = dakikaSa; }

public int getDakikaCrs()
{ return dakikaCrs; }

public void setDakikaCrs(int dakikaCrs)
{ this.dakikaCrs = dakikaCrs; }

public int getDakikaPrs()
{ return dakikaPrs; }

```

```

public void setDakikaPrs(int dakikaPrs)
{   this.dakikaPrs = dakikaPrs; }

public int getDakikaCm()
{   return dakikaCm; }

public void setDakikaCm(int dakikaCm)
{   this.dakikaCm = dakikaCm; }

public int getDakikaCmt()
{   return dakikaCmt; }

public void setDakikaCmt(int dakikaCmt)
{   this.dakikaCmt = dakikaCmt; }

public int getDakikaPa()
{   return dakikaPa; }

public void setDakikaPa(int dakikaPa)
{   this.dakikaPa = dakikaPa; }

public String getMevkiKodu()
{   return mevkiKodu; }

public void setMevkiKodu(String mevkiKodu)
{   this.mevkiKodu = mevkiKodu; }

public String getDepartmanKodu()
{   return departmanKodu; }

public void setDepartmanKodu(String departmanCode)
{   this.departmanKodu= departmanKodu; }

public Date getBitisSuresiTarihi()
{   return bitisSuresiTarihi; }

public void setBitisSuresiTarihi(Date bitisSuresiTarihi)
{   this.bitisSuresiTarihi = bitisSuresiTarihi; }

public Departman getDepartman()
{   return departman; }

public void setDepartman(Departman departman)

```

```

    { this.departman = departman; }

    public int getTotalMinutes()
    { return (dakikaPzt + dakikaSa + dakikaCrs + dakikaPrs + dakikaCm +
        dakikaCmt + dakikaPa); }
}

```

model/ZamancizelgesiYoneticisi.java dosya kodlamaları

```

package com.agileproje.zcproje.model;

import java.util.Date;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.LockMode;
import org.hibernate.Session;
import com.agileproje.zcproje.util.HibernateUtil;

/* ACIKLAMA
 * zamancizelgesi tablosu icin veritabani islemlerinin yonetilmesi
 */
public class ZamancizelgesiYoneticisi
{
    /*
    * ACIKLAMA
    * kullanıcıId ye gore veritabani kayitlarinda
    * zaman cizelgelerinin eslesme sonucu bulunan liste cagirilir
    */
    public List getZamancizelgeleri(int kullanıcıId)
    {
        List zamancizelgesiList = null;
        Session session = HibernateUtil.getSessionFactory()
            .getCurrentSession();
        session.beginTransaction();
        try
        {
            zamancizelgesiList = session.createQuery(
                "from Zamancizelgesi" + " where kullanıcıId = ?").setInteger(0,
                kullanıcıId).list();
            session.getTransaction().commit();
        }
        catch (HibernateException e)
        { session.getTransaction().rollback();
            throw e;
        }
    }
}

```

```

    }
    return zamancizelgesiList;
}
/*
* ACIKLAMA
* kullanıcıId ve bitisSuresiTarihi degerlerine gore veritabani kayitlarinda
* eslesme sonucu bulunan liste cagirilir.
*/
public Zamancizelgesi getZamancizelgesi(int kullanıcıId, Date bitisSuresiTarihi)
{
    Zamancizelgesi zamancizelgesi = null;

    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    try
    {
        zamancizelgesi = (Zamancizelgesi) session.createQuery(
            "from Zamancizelgesi" + " where kullanıcıId = ?"
            + " and bitisSuresiTarihi = ?").setInteger(0,
            kullanıcıId).setDate(1, bitisSuresiTarihi).uniqueResult();
        session.getTransaction().commit();
    }
    catch (HibernateException e)
    {
        session.getTransaction().rollback();
        throw e;
    }
    return zamancizelgesi;
}
/**
* Zamancizelgesi nesnesinin kaydedilmesi
*/
public void saveZamancizelgesi(Zamancizelgesi zamancizelgesi)
{
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    try
    {
        session.saveOrUpdate(zamancizelgesi);
        session.getTransaction().commit();
    }
    catch (HibernateException e)
    {
        session.getTransaction().rollback();
        throw e;
    }
}

```

```

    }
}
/* ACIKLAMA
*zamancizelgesiId ye gore sirali kayitlardan zaman cizelgesi cagirilir.
*/
public List getZamancizelgeleri()
{
    List zamancizelgesiList = null;
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    try
    {
        zamancizelgesiList = session.createQuery(
            "from Zamancizelgesi ORDER BY ZamancizelgesiId").list();
        session.getTransaction().commit();
    }
    catch (HibernateException e)
    {
        session.getTransaction().rollback();
        throw e;
    }

    return zamancizelgesiList;
}

/**
 * zamancizelgesiId ile eslesen kayitlar silinir.    */

public void deleteZamancizelgesi(int zamancizelgesiId)
{
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    try
    {
        session.delete(session.load(Zamancizelgesi.class, new Integer(
            zamancizelgesiId)));
        session.flush();
        session.getTransaction().commit();
    }
    catch (HibernateException e)
    {
        session.getTransaction().rollback();
        throw e;
    }
}

```

```

public Zamancizelgesi getZamancizelgesi(int zamancizelgesiId, boolean doLock)
{
    Session session = HibernateUtil.getSessionFactory()
        .getCurrentSession();
    session.beginTransaction();
    Zamancizelgesi zamancizelgesi = null;
    try
    {
        if (doLock)
            zamancizelgesi = (Zamancizelgesi) session.get(Zamancizelgesi.class,
                new Integer(zamancizelgesiId), LockMode.UPGRADE);
        else
            zamancizelgesi = (Zamancizelgesi) session.get(Zamancizelgesi.class,
                new Integer(zamancizelgesiId));
        session.getTransaction().commit();
    }
    catch (HibernateException e)
    {
        session.getTransaction().rollback();
        throw e;
    }
    return zamancizelgesi;
}
}

```

test/zclisteDenetleyiciTest.java dosya kodlamaları

```

package com.agileproje.zcproje.test;

import com.agileproje.zcproje.controller.zclisteDenetleyici;
import com.agileproje.zcproje.model.Kullanici;
import com.agileproje.zcproje.model.Zamancizelgesi;
import com.agileproje.zcproje.model.ZamancizelgesiYoneticisi;
import com.agileproje.zcproje.util.UygulamaGuvencilikYoneticisi;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import org.springframework.mock.web.MockHttpServletRequest;
import org.springframework.web.servlet.ModelAndView;

```

```

/** ACIKLAMA
 * zclistedenetleyiciTest icin sinif testi
 */
public class zclistedenetleyiciTest extends TestCase
{
    private MockHttpServletRequest mockHttpServletRequest = null;
    private zclistedenetleyici zclistedenetleyici = null;
    private ZamancizelgesiYoneticisi zamancizelgesiYoneticisi = new
ZamancizelgesiYoneticisi();
    private UygulamaGuvenlikYoneticisi UygulamaGuvenlikYoneticisi = new
UygulamaGuvenlikYoneticisi();
    private final int Kullanici_ID = 2;
    private static int zamancizelgesiId1 = 0;
    private static int zamancizelgesiId2 = 0;

    public static void main(String args[])
    {
        junit.textui.TestRunner.run(suite());
    }

    public static Test suite()
    {
        return new TestSuite(zclistedenetleyiciTest.class);
    }

    /** Bir kullanıcı için zaman çizelgesi listesini çağırma
     */
    public void testShowForm() throws Exception
    {
        mockHttpServletRequest = new MockHttpServletRequest("GET",
            "/" + "zc_liste.htm");

        Kullanici Kullanici = new Kullanici();
        Kullanici.setKullaniciId(Kullanici_ID);
        UygulamaGuvenlikYoneticisi.setKullanici(mockHttpServletRequest,
            Kullanici);

        zclistedenetleyici = new zclistedenetleyici();
        zclistedenetleyici.setZamancizelgesiYoneticisi(zamancizelgesiYoneticisi);
        zclistedenetleyici
            .setUygulamaGuvenlikYoneticisi(UygulamaGuvenlikYoneticisi);

        ModelAndView modelAndView = zclistedenetleyici.handleRequest(
            mockHttpServletRequest, null);
    }
}

```

```

assertNotNull(modelAndView);
assertNotNull(modelAndView.getModel());

List zamancizelgeleri = (List) modelAndView.getModel().get(
    zclisteDenetleyici.MAP_KEY);
assertNotNull(zamancizelgeleri);

Zamancizelgesi zamancizelgesi;
for (int i = 0; i < zamancizelgeleri.size(); i++)
{
    zamancizelgesi = (Zamancizelgesi) zamancizelgeleri.get(i);
    assertEquals(Kullanici_ID, zamancizelgesi.getKullaniciId());
    System.out.println(zamancizelgesi.getZamancizelgesiId() + " passed!");
}
}

/**Veritabanında Zamancizelgesi nesnesi testini oluşturma
*/
protected void setUp() throws Exception
{
    Zamancizelgesi zamancizelgesi = null;

    zamancizelgesi = new Zamancizelgesi();
    zamancizelgesi.setKullaniciId(Kullanici_ID);
    zamancizelgesi.setBitisSuresiTarihi(new GregorianCalendar(2006,
        Calendar.MARCH, 04).getTime());
    zamancizelgesi.setMevkiKodu("P");
    zamancizelgesi.setDepartmanKodu("IT");
    zamancizelgesi.setDakikaPzt(480);
    zamancizelgesi.setDakikaSa(480);
    zamancizelgesi.setDakikaCrs(480);
    zamancizelgesi.setDakikaPrs(480);
    zamancizelgesi.setDakikaCm(480);
    zamancizelgesi.setDakikaCmt(0);
    zamancizelgesi.setDakikaPa(0);
    zamancizelgesi.Yoneticisi.saveZamancizelgesi(zamancizelgesi);
    zamancizelgesiId1 = zamancizelgesi.getZamancizelgesiId();

    zamancizelgesi = new Zamancizelgesi();
    zamancizelgesi.setKullaniciId(Kullanici_ID);
    zamancizelgesi.setBitisSuresiTarihi(new GregorianCalendar(2006,
        Calendar.MARCH, 11).getTime());
    zamancizelgesi.setMevkiKodu("A");

```

```

        zamancizelgesi.setDepartmanKodu("IT");
        zamancizelgesi.setDakikaPzt(480);
        zamancizelgesi.setDakikaSa(480);
        zamancizelgesi.setDakikaCrs(480);
        zamancizelgesi.setDakikaPrs(480);
        zamancizelgesi.setDakikaCm(480);
        zamancizelgesi.setDakikaCmt(0);
        zamancizelgesi.setDakikaPa(0);
        ZamancizelgesiYoneticisi.saveZamancizelgesi(zamancizelgesi);
        zamancizelgesiId2 = zamancizelgesi.getZamancizelgesiId();
    }
    /* ACIKLAMA
    * Veritabanında Zamancizelgesi nesnesi testini silme
    */
    protected void tearDown() throws Exception
    {
        ZamancizelgesiYoneticisi.deleteZamancizelgesi(zamancizelgesiId1);
        ZamancizelgesiYoneticisi.deleteZamancizelgesi(zamancizelgesiId2);
    }
}

```

test/ZamancizelgesiYoneticisiTest.java dosya kodlamaları

```

package com.agileproje.zcproje.test;

import com.agileproje.zcproje.model.Zamancizelgesi;
import com.agileproje.zcproje.model.ZamancizelgesiYoneticisi;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.List;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class ZamancizelgesiYoneticisiTest extends TestCase
{
    ZamancizelgesiYoneticisi zamancizelgesiYoneticisi = new
    ZamancizelgesiYoneticisi();

    public static void main(String args[])
    {
        junit.textui.TestRunner.run(suite());
    }

    public static Test suite()

```

```

{
    return new TestSuite(ZamancizelgesiYoneticisiTest.class);
}

public void testGetByKullaniciId()
{
    List zamancizelgesiList = zamancizelgesiYoneticisi.getZamancizelgeleri();
    assertNotNull(zamancizelgesiList);
    assertTrue(zamancizelgesiList.size() > 0);

    int kullaniciId = ((Zamancizelgesi) zamancizelgesiList.get(0)).getKullaniciId();
    zamancizelgesiList =
zamancizelgesiYoneticisi.getZamancizelgeleri(kullaniciId);
    assertNotNull(zamancizelgesiList);

    Zamancizelgesi zamancizelgesi;
    for (int i = 0; i < zamancizelgesiList.size(); i++)
    {
        zamancizelgesi = (Zamancizelgesi) zamancizelgesiList.get(i);
        assertEquals(kullaniciId, zamancizelgesi.getKullaniciId());
        System.out.println(">>>> Departman isim = "
            + zamancizelgesi.getDepartman().getName());
    }
}

public void testGetByEmployeeAndPeriod()
{
    List zamancizelgesiList = zamancizelgesiYoneticisi.getZamancizelgeleri();
    if (zamancizelgesiList == null || zamancizelgesiList.size() < 1) return;

    Zamancizelgesi zamancizelgesi = (Zamancizelgesi) zamancizelgesiList.get(0);
    final int kullaniciId = zamancizelgesi.getKullaniciId();
    Date bitisSuresiTarihi = zamancizelgesi.getBitisSuresiTarihi();
    zamancizelgesi = zamancizelgesiYoneticisi.getZamancizelgesi(kullaniciId,
        bitisSuresiTarihi);
    assertNotNull(zamancizelgesi);

    assertEquals(kullaniciId, zamancizelgesi.getKullaniciId());
    assertTrue(!bitisSuresiTarihi.before(zamancizelgesi.getBitisSuresiTarihi()));
    assertTrue(!bitisSuresiTarihi.after(zamancizelgesi.getBitisSuresiTarihi()));
    System.out.println(">>>> Departman Adi = "
        + zamancizelgesi.getDepartman().getName());
}

```

```

public void testSaveSingle()
{
    final int KULLANICI_ID = 1111;
    GregorianCalendar gc = new GregorianCalendar(2006,
        Calendar.SEPTEMBER, 15);
    Date bitisSuresiTarihi = gc.getTime();
    Zamancizelgesi zamancizelgesi =
ZamancizelgesiYoneticisi.getZamancizelgesi(KULLANICI_ID,
        bitisSuresiTarihi);
    boolean newRecord=false;
    if (zamancizelgesi == null)
    {
        zamancizelgesi = new Zamancizelgesi();
        zamancizelgesi.setKullaniciId(KULLANICI_ID);
        zamancizelgesi.setBitisSuresiTarihi(bitisSuresiTarihi);
        newRecord = true;
    }
    else
        System.out.println("zamancizelgesi record found: "
            + zamancizelgesi.getZamancizelgesiId());

    zamancizelgesi.setMevkiKodu("P");
    zamancizelgesi.setDepartmanKodu("IT");
    zamancizelgesi.setDakikaPzt(480);
    zamancizelgesi.setDakikaSa(480);
    zamancizelgesi.setDakikaCrs(480);
    zamancizelgesi.setDakikaPrs(480);
    zamancizelgesi.setDakikaCm(480);
    zamancizelgesi.setDakikaCmt(0);
    zamancizelgesi.setDakikaPa(0);
    ZamancizelgesiYoneticisi.saveZamancizelgesi(zamancizelgesi);

    Zamancizelgesi timesheetInDb =
ZamancizelgesiYoneticisi.getZamancizelgesi(KULLANICI_ID,
        bitisSuresiTarihi);
    assertEquals(timesheetInDb.getKullaniciId(), zamancizelgesi.getKullaniciId());
    assertEquals(timesheetInDb.getMevkiKodu(), "P");
    assertEquals(timesheetInDb.getDepartmanKodu(), "IT");
    assertEquals(timesheetInDb.getDakikaPzt(), zamancizelgesi.getDakikaPzt());
    assertEquals(timesheetInDb.getDakikaSa(), zamancizelgesi.getDakikaSa());
    assertEquals(timesheetInDb.getDakikaCrs(), zamancizelgesi.getDakikaCrs());
    assertEquals(timesheetInDb.getDakikaPrs(), zamancizelgesi.getDakikaPrs());
    assertEquals(timesheetInDb.getDakikaCm(), zamancizelgesi.getDakikaCm());
    assertEquals(timesheetInDb.getDakikaCmt(), zamancizelgesi.getDakikaCmt());
}

```

```

        assertEquals(timesheetInDb.getDakikaPa(), zamancizelgesi.getDakikaPa());

ZamancizelgesiYoneticisi.deleteZamancizelgesi(zamancizelgesi.getZamancizelgesiId());
    }

    public void testDeleteByKullaniciId()
    {
        final int KULLANICI_ID = 9999;
        Date bitisSuresiTarihi = new Date();
        Zamancizelgesi zamancizelgesi =
ZamancizelgesiYoneticisi.getZamancizelgesi(KULLANICI_ID,
        bitisSuresiTarihi);

        if (zamancizelgesi == null)
        {
            zamancizelgesi = new Zamancizelgesi();
            zamancizelgesi.setKullaniciId(KULLANICI_ID);
            zamancizelgesi.setBitisSuresiTarihi(bitisSuresiTarihi);
        }
        else

ZamancizelgesiYoneticisi.deleteZamancizelgesi(zamancizelgesi.getZamancizelgesiId());

//ZamancizelgesiYoneticisi.deleteTimesheetUsingCreateQuery(zamancizelgesi.getTimesheetId());

        zamancizelgesi.setKullaniciId(KULLANICI_ID);
        zamancizelgesi.setBitisSuresiTarihi(bitisSuresiTarihi);
        zamancizelgesi.setMevkiKodu("P");
        zamancizelgesi.setDepartmanKodu("IT");
        zamancizelgesi.setDakikaPzt(480);
        zamancizelgesi.setDakikaSa(480);
        zamancizelgesi.setDakikaCrs(480);
        zamancizelgesi.setDakikaPrs(480);
        zamancizelgesi.setDakikaCm(480);
        zamancizelgesi.setDakikaCmt(0);
        zamancizelgesi.setDakikaPa(0);
        ZamancizelgesiYoneticisi.saveZamancizelgesi(zamancizelgesi);

        assertTrue(zamancizelgesi.getZamancizelgesiId() > 0);

```

```
ZamancizelgesiYoneticisi.deleteZamancizelgesi(zamancizelgesi.getZamancizelgesiId());
```

```
        zamancizelgesi =  
ZamancizelgesiYoneticisi.getZamancizelgesi(KULLANICI_ID,  
        bitisSuresiTarihi);  
        assertNull(zamancizelgesi);  
    }  
  
    public void testGetByZamancizelgesiId()  
    {  
        List zamancizelgesiList = ZamancizelgesiYoneticisi.getZamancizelgeleri();  
        assertNotNull(zamancizelgesiList);  
        assertTrue(zamancizelgesiList.size() > 0);  
  
        Zamancizelgesi timesheetInList = (Zamancizelgesi)zamancizelgesiList.get(0);  
        int zamancizelgesiId = timesheetInList.getZamancizelgesiId();  
        Zamancizelgesi timesheetInDb =  
ZamancizelgesiYoneticisi.getZamancizelgesi(zamancizelgesiId, false);  
        assertNotNull(timesheetInDb);  
        assertEquals(timesheetInDb.getKullaniciId(), timesheetInList.getKullaniciId());  
        assertEquals(timesheetInDb.getMevkiKodu(), "P");  
        assertEquals(timesheetInDb.getDepartmanKodu(), "IT");  
        assertEquals(timesheetInDb.getDakikaPzt(), timesheetInList.getDakikaPzt());  
        assertEquals(timesheetInDb.getDakikaSa(), timesheetInList.getDakikaSa());  
        assertEquals(timesheetInDb.getDakikaCrs(), timesheetInList.getDakikaCrs());  
        assertEquals(timesheetInDb.getDakikaPrs(), timesheetInList.getDakikaPrs());  
        assertEquals(timesheetInDb.getDakikaCm(), timesheetInList.getDakikaCm());  
        assertEquals(timesheetInDb.getDakikaCmt(), timesheetInList.getDakikaCmt());  
        assertEquals(timesheetInDb.getDakikaPa(), timesheetInList.getDakikaPa());  
    }  
}
```

view/zamancizelgesiListesi.jsp dosya kodlamaları

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-9">
```

```

<title>Zaman Çizelgesi Listesi </title>
<link href="includes/zcproje.css" rel="stylesheet" type="text/css">
</head>
<body>
<table width="85%" border="1" align="center" cellpadding="0"
    cellspacing="0">
<tr>
    <td>
        <table width="100%" border="0" align="center" cellpadding="4"
            cellspacing="0" bordercolor="#CCCCCC">
        <tr bgcolor="#C2DCEB" valign="middle">
            <td width="90%" height="60" valign="middle"
                bgcolor="#C2DCEB">
                <h1>Zamancizelgesi List</h1>
                </td>
            <td align="right" nowrap="nowrap"><a href=" oturumcikis.htm">Sign
out</a></td>
        </tr>
        <tr>
            <td colspan="2">
                <p align="center"><!-- --> <c:if
                test="{not empty message}">
                <font color="green"><c:out value="{message}" /></font>
                <c:set var="message" value="" scope="session" />
                </c:if></p>
                <p align="center"> yeni zaman çizelgesi eklemek veya aşağıdaki
                listeden eklemek için <a href=" zc_saatgiris.htm">tıklayınız</a></p>
                <table border="1" align="center" cellpadding="8" cellspacing="0">
                <tr>
                    <th> Bitiş Süresi </th>
                    <th> Saat</th>
                    <th> Durum</th>
                    <th> Zaman Çizelgesi Id</th>
                </tr>
                <c:forEach items="{zamancizelgeleri}" var="zamancizelgesi">
                <tr>
                    <td align="center"><a
                        href='zc_saatgiris.htm?tid=<c:out
                            value="{zamancizelgesi.zamancizelgesiId}" />'><fmt:formatDate
                            value="{zamancizelgesi.bitisSuresiTarihi}"
                                type="date"
                                    pattern="MM/dd/yyyy" /></a>
                    </td></tr>
                </c:forEach>
            </td>
        </tr>
    </td>
</tr>

```

```

</td>
<td align="center"><fmt:formatNumber
    value="\${zamancizelgesi.totalMinutes / 60}"
    pattern="0.00" /></td>
<td align="center"><c:out
    value="\${zamancizelgesi.departman.isim}" />
<div></div>
</td>
<td align="center">
<c:out value="\${zamancizelgesi.zamancizelgesiId}" />
    <div></div>
</td>
</tr>
</c:forEach>
</table>
</td>
</tr>
</table>
</td>
</tr>
</table>
</body>
</html>

```

build.xml ANT kurulum dosya kodlamaları

```

<?xml version="1.0"?>

<!-- Agile Yazilim Gelistirme -->
<project name="zcproje" basedir="." default="build">
    <property name="appname" value="\${ant.project.name}" />
    <property name="env" value="local" />

    <!-- dosya ozelliklerinin yuklenmesi -->

    <property file="local.properties" />
    <property name="lib.dir" value="../lib" />
    <property name="dist.dir" value="dist" />
    <property name="src.dir" value="src" />
    <property name="tmp.dir" value="tmp" />
    <property name="img.dir" value="images" />
    <property name="data.dir" value="data" />
    <property name="build.dir" value="build" />
    <property name="conf.dir" value="\${src.dir}/conf" />

```

```

<property name="web.dir" value="${src.dir}/web" />
<property name="war.dir" value="${build.dir}/${appname}" />
<property name="war.file" value="${dist.dir}/${appname}.war" />
<property name="webinf.dir" value="${war.dir}/WEB-INF" />
<property name="class.dir" value="${war.dir}/WEB-INF/classes" />
<property name="warlib.dir" value="${war.dir}/WEB-INF/lib" />
<property name="jsp.dir" value="${war.dir}/WEB-INF/jsp" />
<property name="views.dir"
    value="${src.dir}/java/com/agileproje/zcproje/view" />
<property name="model.dir"
    value="${src.dir}/java/com/agileproje/zcproje/model" />
<property name="controller.dir"
    value="${src.dir}/java/com/agileproje/zcproje/controller" />
<property name="util.dir"
    value="${src.dir}/java/com/agileproje/zcproje/util" />
<property name="envprops" value="${env}.properties" />
<property name="zcprojeprops" value="${class.dir}/zcproje-servlet.properties"/>

    <path id="master-classpath" description="kodlamalar icin ana CLASSPATH">
    <fileset dir="${lib.dir}">
        <include name="*.jar" />
    </fileset>
    <pathelement location="build/zcproje/WEB-INF/classes/" />
</path>

    <target name="init" description="Kodların olusturulması için ayarlar">
    <mkdir dir="${conf.dir}" />
    <mkdir dir="${web.dir}" />
    <mkdir dir="${data.dir}" />
    <mkdir dir="${class.dir}" />
    <mkdir dir="${lib.dir}" />
    <mkdir dir="${warlib.dir}" />
    <mkdir dir="${jsp.dir}" />
    <mkdir dir="${views.dir}" />
    <mkdir dir="${model.dir}" />
    <mkdir dir="${controller.dir}" />
    <mkdir dir="${util.dir}" />
</target>
<target name="updateweb" depends="init"
    description="WAR dizinine ilgili web dosyalarının
kopyalanması">
    <copy todir="${war.dir}">
        <fileset dir="${web.dir}">
            <include name="**/*" />

```

```

        </fileset>
    </copy>
    <copy todir="${jsp.dir}">
        <fileset dir="${views.dir}" />
    </copy>
</target>

<target name="updatelib" depends="init"
        description="WAR dizinine JAR dosyarinin kopyalanmasi">
    <copy todir="${warlib.dir}">
        <fileset dir="${lib.dir}">
            <exclude name="*servlet*.jar" />
            <exclude name="*jetty*.jar" />
            <exclude name="*junit*.jar" />
            <exclude name="*spring-mock*.jar" />
        </fileset>
    </copy>
</target>

<target name="deleteconfig" depends="init"
        description="WAR dizininden konfigurasyon dosyalarinin
silinmesi">
    <delete failonerror="false">
        <fileset dir="${war.dir}">
            <include name="**/*.properties" />
            <include name="**/*.xml" />
        </fileset>
    </delete>
</target>

<target name="updateconfig" depends="init"
        description="WAR dizinine konfigurasyon dosyalarinin
kopyalanmasi">
    <echo level="info">Environment config file: ${envprops}</echo>
    <copy file="${envprops}" tofile="${zcprojeprops}" />
    <copy todir="${class.dir}">
        <fileset dir="${src.dir}/java/com/agileproje/zcproje/model/">
            <include name="**/*.xml" />
        </fileset>
        <fileset dir="${src.dir}/conf">
            <include name="**/*.properties" />
            <exclude name="dev.properties" />
            <exclude name="test.properties" />
            <exclude name="acpt.properties" />
        </fileset>
    </copy>
</target>

```

```

        <exclude name="prod.properties" />
    </fileset>
</copy>
<copy todir="${webinf.dir}" preservelastmodified="true">
    <fileset dir="${src.dir}/conf">
        <include name="**/*.xml" />
        <include name="**/*.tld" />
    </fileset>
</copy>
</target>

<target name="compile" depends="init"
        description="WAR dizinine .java dosyalarının derlenmesi">
    <javac srcdir="${src.dir}" destdir="${class.dir}" debug="true"
        failonerror="true" classpathref="master-classpath"
        depend="true"/>
</target>

<target name="build" depends="compile,updateweb,updateconfig,updatelib" />

<target name="rebuild"
        depends="clean,compile,updateweb,updateconfig,updatelib" />

    <target name="dist" depends="rebuild" description="WAR dosyalarının
yerleştirilmesi">
    <copy todir="${webinf.dir}/${src.dir}">
        <fileset dir="${src.dir}">
            <include name="**/*.java" />
        </fileset>
    </copy>
    <copy todir="${webinf.dir}/${data.dir}">
        <fileset dir="${data.dir}">
            <include name="**/*" />
        </fileset>
    </copy>
    <mkdir dir="${dist.dir}" />
    <war destfile="${war.file}" webxml="${src.dir}/conf/web.xml">
        <fileset dir="${war.dir}">
            <include name="**/*.*" />
            <exclude name="**/web.xml" />
            <exclude name="**/test/*.class" />
            <exclude name="**/*mock*.jar" />
        </fileset>
    </war>

```

```

    <delete dir="${webinf.dir}/${src.dir}" failonerror="false" />
</target>

<target name="deploy" depends="dist"
        description="deploy.dir WAR dosyalarinin dagitilmasi">
    <delete failonerror="true" dir="${deploy.dir}/${appname}" />
    <copy file="${war.file}" todir="${deploy.dir}" />
</target>

    <target name="clean" description="WAR dizininden dosyalarin silinmesi">
<delete failonerror="false">
    <fileset dir="${war.dir}">
        <include name="**/*.*" />
    </fileset>
</delete>
<delete failonerror="false">
    <fileset dir="${dist.dir}">
        <include name="**/*.*" />
    </fileset>
</delete>
</target>

<target name="junitswing">
    <java fork="true" classpathref="master-classpath"
        classname="junit.swingui.TestRunner" />
</target>

<target name="test" depends="compile">
    <junit printsummary="true" showoutput="yes" filtertrace="false">
        <classpath refid="master-classpath" />
        <batchtest fork="yes">
            <formatter type="plain" />
            <fileset dir="${class.dir}">
                <include name="**/*Test.class" />
                <exclude name="**/HibernateTest.class" />
                <exclude name="**/SpringTest.class" />
                <exclude name="**/AllTests.class" />
            </fileset>
        </batchtest>
    </junit>
</target>

<target name="hibernatetest" depends="build">
    <java fork="true" classpathref="master-classpath"

```

```
        classname="com.agileproje.zcproje.test.HibernateTest" />
    </target>

    <target name="springtest" depends="build">
        <java fork="true" classpathref="master-classpath"
            classname="com.agileproje.zcproje.test.SpringTest" />
    </target>
    <target name="zcListeKontrolutest">
        <java fork="true" classpathref="master-classpath"
            classname="com.agileproje.zcproje.test.zcListeKontroluTest"/>
    </target>
</project>
```

ÖZGEÇMİŞ

1978 yılında İstanbul’da doğdu. İlk ve orta öğrenimini İstanbul’da tamamladı. 1996 yılında Kartal Teknik ve Endüstri Meslek Lisesinden mezun oldu. 1997-2001 yılları arasında Marmara Üniversitesi Teknik Eğitim Fakültesi Bilgisayar ve Kontrol Öğretmenliği Lisans öğrenimini tamamladı. 2001 yılında Batman ilinde MEB kurumuna bağlı bilgisayar öğretmeni olarak işe başladı. 2003 yılında aynı kuruma bağlı Maltepe ili Handan-Hayrettin Yelkikanat Teknik ve Endüstri Meslek Lisesinde görevine başladı. Aynı kurumda halen görevine devam etmektedir.