EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

DAĞITIK UÇ PROGRAMLAMA

Mohammed Jamal Ahmed ABBASI

Tez Danışmanı: Prof. Dr. Halil ŞENGONCA

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu: 619.02.00
Sunuş Tarihi: 01.07.2011

Bornova-İZMİR

2011

Mohammed Jamal Ahmed ABBASI tarafından YÜKSEK LİSANS tezi olarak sunulan "Dağıtık Uç Programlama" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 01.07.2011 tarihinde yapılan tez savunma sınavında aday oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı: Prof.Dr. Halil ŞENGONCA	**….....**
Raportör Üye: Doç.Dr. Aybars UĞUR	**.....**
Üye: Yrd.Doç. Korhan KARABULUT	**….....**

ÖZET

DAĞITIK UÇ PROGRAMLAMA

ABBASI, Mohammed Jamal Ahmed

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Halil ŞENGONCA
Temmuz 2011, 93 sayfa

Yazılım geliştirmede çevik süreçler, değişikliklerin kaçınılmaz olduğu günümüzün değişken iş dünyasında yazılım geliştirmek için sundukları ortak uygulamalar ve kişi odaklı bakış açıları ile hızla önem kazanmaktadır. Bu süreçlerle geliştirilen projelerin verimliliğinde ekip ortak bir alanda çalışmasının getirdiği iletişim ve işbirliğinin önemi büyüktür. Ama bazı durumlarda ekibin ortak bir alanda çalışması mümkün olmamaktadır; Bu yüzden çevik süreçlerin dağıtık ekiplerde de desteklemesi sağlanmalıdır. Kircher, Jain, Corsaro ve Levine bu noktanın farkına varmış ve hazır yazılımları kullanarak yüz yüze iletişimin getirdiği verimliliğin yazılımsal araç desteği ile sağlanabileceğini göstererek, çevik süreçleri kullanan Dağıtık Uç Programlama (DUP) yöntemini ortaya koymuşlardır.

Bu çalışmada, HP'nin Türkiye'deki en büyük iş ortağı olan 4S Bilgi Teknolojilerinin Irak'ın en büyük GSM operatörü olan Asiacell ile yaptığı ortak projede Dağıtık Uç Programlama kullanarak gerçekleştirilmiştir. Günümüzde iletişim araçlarının yaygınlaşması ve gelişmesinden yararlanarak bir ekibin çalışanları fiziksel olarak ne kendi aralarında nede müşteri ile aynı ortamda bulunmadan bu iletişim araçlarından yararlanarak Dağıtık Uç Programlamayı uygulamak için bir örnek göstermektedir.

Anahtar sözcükler: Çevik Yazılım Geliştirme Süreçleri, Uç Programlama, Dağıtık Uç Programlama.

SUMMARY

DISTRIBUTED EXTREME PROGRAMMING

ABBASI, Mohammed Jamal Ahmed

MSc. in Computer Engineering
Thesis Instructor: Prof. Dr. Halil ŞENGONCA
July 2011, 93 pages

Agile Software Development has gained popularity with their people centric view and their common practices for developing software in today's volatile business world where change on requirements is unavoidable. However; the efficiency of the project depends on the communication and the collaboration of the team, which are supported by the co-location of the team. But in some cases co-location of the team cannot be realized, thus agile processes should also support distributed teams. This point was observed by Kircher, Jain, Corsaro, and Levine and they suggested Distributed eXtreme Programming (DXP) after they prepared a study using off-the shelf software products in order to replace the effect of face-to face communication on the efficiency of the application of agile processes with the aid gathered from tool support.

In this study, we explain the project of 4S Information Technology with the largest GSM operator in Iraq Asiacell, this project realized using Distributed Extreme Programming. Today, if employees of a team and also the customer do not be in the same place physically we can realize Distributed Extreme Programming by taking advantage of widespread and growth of the communication tools.

Keywords: Agile Software Development, eXtreme Programming, Distributed eXtreme Programming

TEŞEKKÜR

Öncelikle tez konumun belirlenmesinden başlayarak sonuçlandırılmasına kadar geçen süre boyunca benden destek ve rehberliğini esirgemeyen danışmanım Prof. Dr. Halil ŞENGONCA' ya değerli zaman ve deneyimlerini, benimle paylaştığı için teşekkürü bir borç bilirim. Ayrıca hayatımın her aşamasında olduğu gibi tez çalışmam süresince da benden maddi – manevi desteklerini esirgemeyen aileme teşekkür ederim. Türkiye'ye geldiğim ilk günden beri gerek eğitim gerekse sosyal yaşama uyum sağlamamda büyük yardımlarını gördüğüm ve bana ikinci bir aile olan, Türkmeneli, Türkiye ve Türk dünyasının diğer coğrafyalarından bütün arkadaşlarıma içtenlikle teşekkür ederim; varlıkları umudumu arttırdı. Son olarak tezimin düzenlemerinde yardımcı olan manevi ablam Arş. Gör. Gaye YAVUZCAN'a teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER(devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER(devam)

Sayfa

İÇİNDEKİLER(devam)

Sayfa

İÇİNDEKİLER(devam)

Sayfa

İÇİNDEKİLER(devam)

Sayfa

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

ÇIZELGE SAYFA

SİMGELER DİZİNİ

Simgeler	Açıklama
Ar-Ge	Araştırma Geliştirme
BT	Bilişim Teknolojileri
BTO	Business Technology Optimization İş Teknolojilerinin Eniyileştirilmesi
CMM	Capability Maturity Model Yetenek Olgunluk Modeli
COACH-IT	Component Oriented Agile Collaborative Handler of Integration and Testing Sınama ve Bütünleşme için Bileşen Tabanlı Çevik Ortak İşleyicisi
CVS	Concurrent Versions System Eşzamanlı Sürümleri Sistemi
CHAOS	Charting the Seas of information system. Bilişim Sistemlerinin Deniz Çizimleri
CAATCS	Canadian Automated Air Traffic Control System Kanada Otomatik Hava Trafiği Denetim Sistemi
CI	Configuration Items – Yapılanış Öğesi
CCPDSR	Command Center Processing and Display System Replacement Komut Merkez Süreci ve Görüntü Sisteminin Değiştirilmesi
DPP	Distributed Pair Programming Dağıtık Çift Programlama
DXP	Distributed eXtreme Programming Dağıtık Uç Programlama
DDM	Discovery and Dependency Mapping Keşif ve Bğımlılık Eşlemi

SİMGELER DİZİNİ (Devam)

Simgeler	Açıklama
DDMI	Discovery and Dependency Mapping Inventory Keşif ve Bağımlılık Eşlemi Stoğu
DB	Database – Veritabanı
DUP	Dağıtık Uç Programlama
FSD	Federal System Division – Birleşik Bölme Sistemi
FDD	Feature Driven Development Özelliğe Dayalı Geliştirme
IT	Information Technology – Bilişim Teknolojileri
ITIL	Information Technology Infrastructure Library Bilişim Teknolojileri Altyapı Kütüphanesi
IID	Interactive Incremental Development Etkileşimli ve Artışlı Geliştirme
İTE	İş Teknolojileri Enayileştirilmesi
LAMP	Light Arborne Multipurpose System Çok Amaçlı Light Arbone Sistemi
MILOS	Minimally Invasive Long-term Organizational Support En Kısa Yayılımlı Uzun Vadeli Örgütsel Destek
MASE	MILOS Agile Software Engineering MILOS Çevik Yazılım Mühendisliği
MS	Microsoft
MEMA	Middle East and Middle Asia Ortadoğu ve Orta Asya
MTTR	Mean Time to Repair Ortalama Onarım Süresi
NNM	HP Network Node Manager HP Ağ Düğümü Yöneticisi

SİMGELER DİZİNİ (Devam)

Simgeler	Açıklama
NFS	Network File System Ağ Dosyası Sistemi
OM	HP Operation Manager HP İşlemler Yönetimi
OS	Operating System İşletim Sistemi
RP	Report Pack – Rapor Paketi
RPP	Remote Pair Programming **–** Uzak Çift Programlama
SVN	Subversion – Alt Sürüm
SNMP	Simple Network Management Protocol Yalın Ağ Yönetimi İletişim Kuralları
SD	Service Desk – Hizmet Masası
SM	HP Service Manager – HP Hizmet Yöneticisi
SPI	Smart Plug-in – Akıllı Uyumlu Ek
TDD	Test Driven Development – Sınamaya Dayalı Geliştirme
UCMDB	HP Universal Configuration Management Database HP Evrensel Yapılanış Onetime Veri Tabanı
UI	User Interface – Kullanıcı Arayüzü
UP	Uç Programlama
VCS	Version Control System – Sürüm Denetim Sistemi
VPN	Virtual Private Network – Sanal Özel Ağ
WebDAV	Web-based Distributed Authoring and Versioning Web'e Dayalı Dağıtık Yazma ve Sürümlendirme
XP	eXtreme Programming – Uç Programlama

1.GİRİŞ

Yazılım geliştirme süreci yazılımın verimliliğini ve niteliğini arttırmak için biçimlenmeye başladı. Ana amaç tutarlı yazılım geliştirme yöntemi geliştirmek ve bu yöntemle beklenen işlevsellik, maliyet ve istenilen sürede yazılım geliştirmektir. Genelde bütün süreçler aynı aşamalardan oluşur: Gereksinim Çözümleme, Tasarım ve Mimari, Kodlama, Sınama ve Bakım. Yazılım geliştirme projeleri, aralarındaki değişiklikler ise bu aşamaların uygulama sırası ve biçimi, örneğin Çağlayan Modelinde (Semih, 2008) olduğu gibi bütün aşamalar bir yaşam döngüsü biçiminde uygulanır ve bu yaşam döngüsü bir kaç kez yinelenir veya bu aşamalar kısa döngüler biçiminde yapılır ve bu döngüler sürekli iyileştirilir. Tıpkı, Artışlı ve Etkileşimli Modellerde olduğu gibi (Semih, 2008). Bütün bu aşamalar ayrıntılı bir biçimde belgelendirilmelidir ve her aşamanın girdileri ve çıktıları yazılıp bu aşamalar sıralı ve düzenli bir kapsamda belgelendirilme işlemi ile beraber gerçekleştirilmelidir. Geliştirme sürecinin en büyük sorunu, süreç üzerinde yapılan değişiklikler ve iyileştirmelerin maliyetin yükselmesine neden olması ve bu yükselmenin projenin onaylanmamasına yol açması olasılığıdır.

Yazılım geliştirme karmaşık ve zor bir süreçtir. Bu karmaşanın bir bölümünü öngöremeyiz, bir bölümünü ise ayrıntılı olarak inceleyemeyiz. Ancak yazılım geliştirme sürecinde bu karmaşayı büsbütün göz ardı etmek olanaksızdır. Yazılım geliştirme sürecinin zorluğu tahmin edilebilirliği azaltmakta, yanlışların yinelenmesine ve çaba yitimine neden olmaktadır. Müşteriler ertelenen takvimler, artan bütçeler ve düşük nitelik nedeniyle hayal kırıklığı yaşamaktadır. Geliştiriciler ise saatlerce çalışıp yine de nitelikli yazılımlar üretmemektedir (Schwaber, 2004).

Yukarıda söz ettiğimiz sorunlar, bir takım korkulara yol açmaktadır. Bu korkular:

- Beklenen dışında bir ürün geliştirmek,
- Beklenenden daha düşük nitelikte bir ürün geliştirmek,
- Projeyi beklenenden daha geç tamamlamak,
- Haftada 80 saat çalışmak zorunda kalmak,
- Yazılım geliştirme sürecini eğlenceli duruma getirememek gibi olası sorunlardan kaynaklanmaktadır (Martin, 2008).

Bu korkular, proje ekibinin etkinliklerini kısıtlamakta ve kesin çıktılar üretilmesini bekleyen süreçlerin geliştirilmesine neden olmaktadır. Bu süreçler geçmişteki deneyimlere göre oluşturulmakta ve yaşanılan sorunları çözeceği düşünülmektedir. Ancak yazılım projeleri alınan bazı önlemlerin yanlışların önüne geçmesini sağlayacak kadar kolay değildir. Bu önlemler sorunların önüne geçememekte, ortaya çıkan yeni yanlışlara bağlı olarak yeni kısıtlamalar belirlenmektedir. Birkaç proje sonunda geliştiricilerin yeteneklerini ve gücünü engelleyen karmaşık ve hantal süreçler oluşmaktadır (Martin, 2008).

Karmaşık süreçler, yanlışları önlemesi beklenirken tersine birçok yanlışa neden olabilmektedir. Bu süreçler beklenen takvimin uzamasına, bütçe sınırlarının aşılmasına yol açabilecek biçimde proje ekibini yavaşlatabilmektedir. Bu zor süreç içerisinde yazılım sektörü, nereye varacağı bile belli olmayan projelerden dolayı çok kan kaybetmiştir. Bu projeler büyük serüvenler durumuna gelmiştir (Acar, 2009).

2001 yılının başlarında, bu sorunlardan dolayı büyüyen yazılım geliştirme süreçleri nedeniyle kuruluşlardaki yazılım geliştirme ekiplerinin bir bataklık içerisinde saplanıp kaldığını gören yazılım endüstrisinde uzman kişilerden oluşan bir ekip, yazılım geliştiricilerin daha hızlı ve değişikliklere yanıt verebilecek bir biçimde yazılım geliştirebilmelerini sağlayacak ilkeleri belirlemek için bir araya gelmiştir. Kendilerini Çevik Birlik (Agile Alliance) olarak adlandıran bu ekip iki günlük bir çalışmanın ardından bir bildiri yayınlamıştır. Ardından yapılan üç aylık bir çalışma ile çeviklik ilkesinin temel ilkeleri belirlenmiştir (Martin, 2008).

Günümüzde kolayca değişebilen iş dünyası sürekli gereksinimlerin değişmesine neden olmakta ve bu kaçınılmaz değişliklerin sorununu Çevik Yazılım Geliştirme Yöntemleri çözmektedir; çünkü bu yöntemler sürekli gereksinim değişikliklerine yanıt verebilmektedir. Çevik Yöntemler süreç odaklı bir yöntemden daha fazla insan odaklı bir yöntemdir.

Çevik süreçler ortaya çıktıkları günden beri oldukça çok kullanılan duruma gelmiştir. Günümüzde Google, Yahoo, Symantec, Microsoft gibi birçok büyük kuruluş çevik süreçleri kullanmakta ve uygulamaktadır (Shore, 2008). Çeviklik ilkesine bağlı olarak yazılımı hızlı bir biçimde yetiştirmek, müşterinin değişen gereksinimlerini tam olarak karşılayabilmek için birçok yöntem geliştirilmiştir. Tüm bu yöntemleri incelediğimizde bazı temel ilkeleri paylaştıkları görülebilir:

- Müşteri memnuniyeti,
- Değişen gereksinimlere hızlı bir biçimde uyum sağlayabilme,
- Yazılım ürünlerini hızlı bir biçimde yetiştirmek,
- Geliştiricilerin ve kullanıcıların yakın bir işbirliği içerisinde çalışabilmesi (Shore, 2008).

Bu özellikleriyle çevik süreçler hantal ve büyük süreçlerin yol açtığı sorunları ve belirsizlikleri çözebilecek niteliktedir. Ancak çevik süreçlerin yazılım geliştirme sürecini sorunsuz bir duruma getirilmesi beklenemez. Brooks, 1986 yılında yazılım geliştirme sürecindeki tüm sorunları çözebilecek "gümüş bir kurşun" olmadığını belirtmiştir. Çevik süreçler, her ne kadar geleneksel süreçlere göre bazı sorunları giderebilmiş olsa da birer gümüş kurşun değildir. Dolayısıyla çevik süreçlerin sağlayacağı yararlar dikkate alınarak kullanılması ve beklentilerin çok yüksek olmaması gerekmektedir (Shore, 2008).

Bu çalışma 7 bölümden oluşmaktadır. Birinci bölümde yazılım geliştirme ile ilgili genel bir bilgi verilmektedir. İkinci bölümde Çevik Süreçler ayrıntılı bir biçimde incelenmiş ve özetlenmiştir. Üçüncü bölümde çevik süreçlerin en çok kullanılan uygulamalarından biri olan Uç Programlama incelenmiş ve özetlenmiştir. Dördüncü bölümde dağıtık programlamayı destekleyen araçlardan örnekler verilmektedir. Beşinci bölümde çalışmamızın ana konusu olan Dağıtık Uç Programla ayrıntılı bir biçimde yer almaktadır. Altıncı bölüm ise bu çalışmanın 4S Bilgi Teknolojilerinin Irak'ın en büyük GSM operatörü olan Asiacell ile yaptığı ortak projede Dağıtık Uç Programlama kullanarak gerçekleştirilmesini göstermektedir. Çalışmamızda son olarak, kullanılan kaynaklara yer verilmektedir.

2. ÇEVİK SÜREÇLER

2.1 Yazılım Geliştirme

Yazılım geliştirme, yavaş, pahalı ve yanlışa açık bir süreçtir. Geliştirme süreci içerisinde oluşabilecek sorunlar geliştirilen ürünlerde kullanılabilirlik, güvenilirlik ve başarım ile ilgili gerçek sorunlara yol açabilmektedir (Lorman, 2003).

Yazılım geliştirme sürecindeki ana amaç müşteri gereksinimleri sağlayan, belirlenen bütçe ve planlanan süre içerisinde tamamlanabilen, yanlışlardan arındırılmış, müşterinin rekabet yeteneğini arttıracak bir yazılım sistemi geliştirmektir. Yazılım geliştirme süreçleri bu amaçları sağlayabilmek için gerekli yöntemleri ve etkinlikleri içermektedir (Acar, 2009).

Amaçların net bir biçimde tanımlanabilmesine rağmen, müşteri gereksinimlerini tam olarak sağlayan ve belirlenen bütçe ve süre kısıtları içerisinde tamamlanabilen yazılım projeleri az sayıdadır. Bunun en önemli nedenlerinden biri yazılım geliştirmenin değerlemesi kolay yapılan bir süreç olmamasıdır. Ayrıca her projede yeni teknolojilerin ve araçların kullanılabilmesi, sürecin değerlemesi yapılabilir ve öngörülebilir olmasını daha da zorlaştırmaktadır (Lorman, 2003) (Marjanovic, 2009).

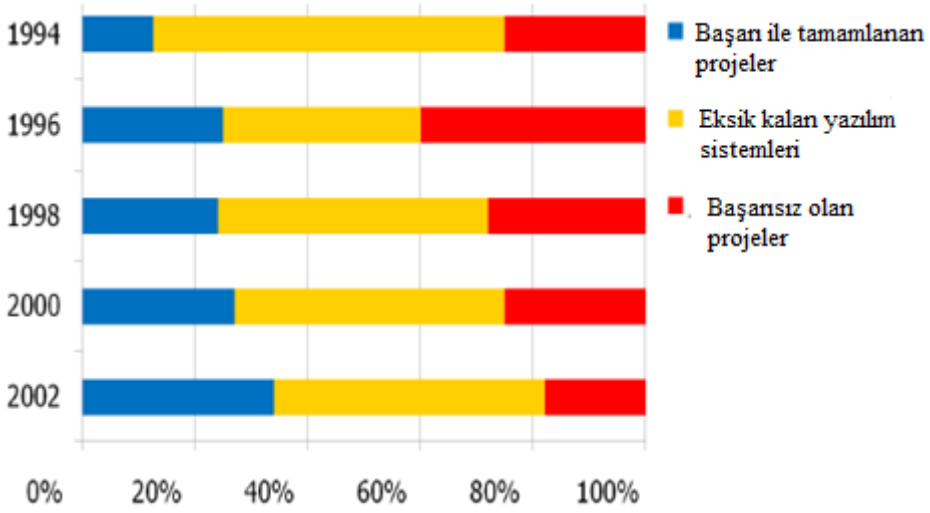
Özellikle geleneksel süreçlerin kullanımında yazılım geliştirme sürecinde oluşabilecek sorunları aşağıdaki gibi özetleyebiliriz:

- Geleneksel yazılım geliştirme yöntemlerinde ilk aşamada müşterinin tüm gereksinimleri çözümleyip belirlemeye çalışılır ve diğer aşamalara geçilir.
- Yazılım geliştirme süreci değişebilecek müşteri gereksinimlerine uyum sağlayabilecek biçimde düzenlemek gerekir. Dolayısıyla geliştirilen yazılımın müşteri gereksinimlerini tam olarak karşılaması çok zordur.
- Müşterinin istekleri doğrultusunda yapılması gereken değişiklikler kolay olmayan yapısal değişiklikler gerektirerek maliyetleri yükseltebilir.
- Proje yöneticilerinin geliştiricilerden beklentileri bu bireylerin çok çalışmalarına, ruhen ve bedenen yıpranmalarına neden olmaktadır.

Özendirmesi (Motivation) düşen bireylerin yaratıcılığı azalmakta ve buna bağlı olarak geliştirilen kodun niteliği düşmektedir

- Projelerde ekip çalışması ve bireyler arası iletişimin güçlendirilmesi desteklenmez. Her geliştirici yazılım içerisinde kendi bölümünden sorumlu olduğundan, projede ayrılabilecek kişiler çekince haline dönüşür
- Proje planı bir süre sonra geçerliliğini yitirebilir. Ancak plana uyulması gerektiği düşüncesi, planın dışına çıkmamak için fazla mesai yapılmasını gerektirir (Acar, 2009).

"Standish Group" tarafından yapılan sayımlama (Bkz. Şekil 2.1), yazılım projelerinin başarıyla tamamlanma oranlarını açık bir biçimde göstermektedir (Acar, 2009). Yapılan bu araştırma başarıyla tamamlanan proje sayısının artış eğilimli bir çizgi izlediğini gösterse de bu oranın %50'nin altında olması başarının yetersiz olduğunu ortaya koymaktadır. Yapılan çalışmada bu yanlışların birçoğunun geleneksel süreçlerden kaynaklandığı belirlenmiştir.



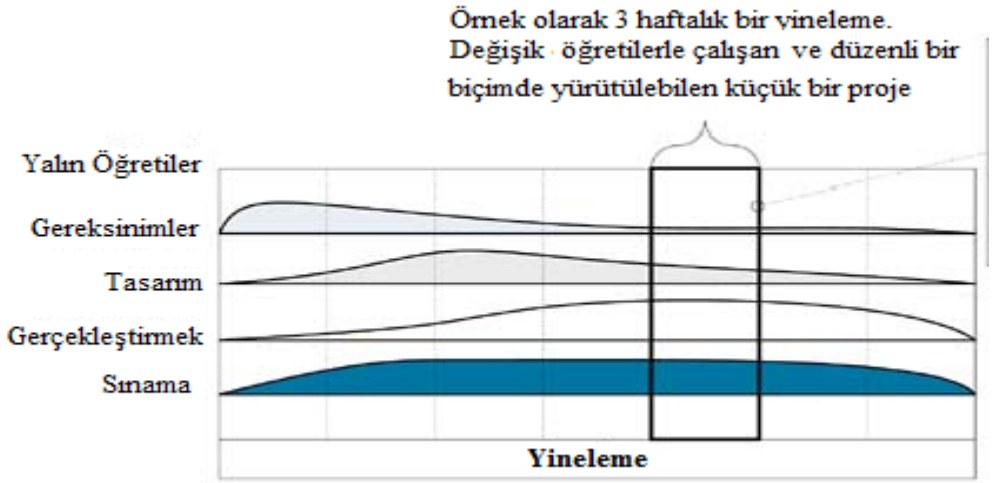
Şekil 2,1 Standish Group Tarafından Gerçekleştirilen Chaos Study sayımlamaları,(Acar, 2009)

Yukarıda belirlenen sorunları ortadan kaldırmak veya azaltmak olanaklıdır. Çevik süreçler bu sorunları azaltarak değişen müşteri gereksinimlerine uygun ve hızlı bir biçimde yetiştirilebilen yazılım ürünlerinin geliştirilebilmesini amaçlamaktadır.

2.2Yinelemeli ve Artışlı Geliştirme

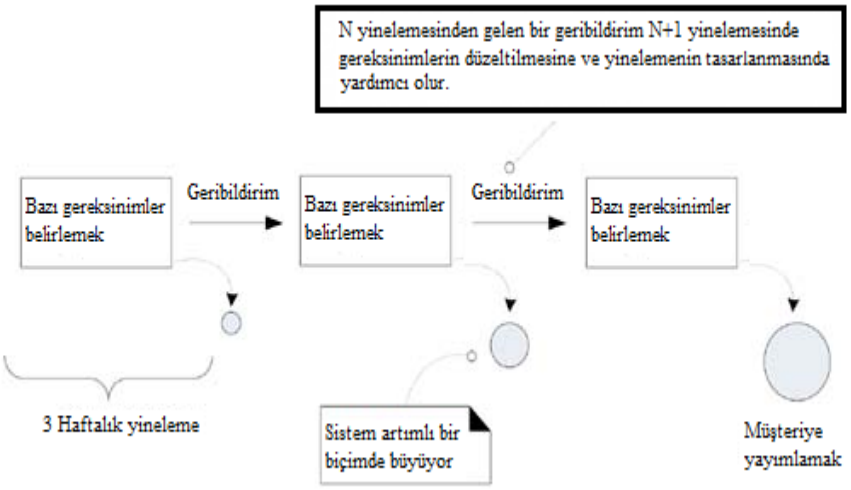
Yazılım projelerinin geliştirilmesi sırasında birçok etkinlik gerçekleştirilir. Sorun anlaşılmalı, çözüme gidebilmek için gereksinimler belirlenmeli, tasarım oluşturulmalı, gerçekleştirim yapılmalı ve sınanmalıdır. Bu etkinlikler olması beklenen genel etkinliklerdir ve çoğunlukla doğrudur. Ancak sorun, gereksinimlerin çözümleme, tasarım, kodlama ve sınama aşamalarının tam olarak ve sıralı bir biçimde gerçekleştirilmeye çalışılmasından kaynaklanmaktadır. Bu yöntem olası çekinceleri arttırmaktadır (Lorman, 2003).

Bu sorunlar bizi yinelemeli ve artışlı geliştirme yaklaşımına yöneltmektedir. Yinelemeli geliştirme, yazılım geliştirme yaşam döngüsünün yinelemelere ayrıldığı bir yaklaşımdır. Her bir yineleme kendi içerisinde gereksinimlerin çözümleme, tasarım, kodlama ve sınama aşamalarından oluşur (Bkz. Şekil 2.2) (Lorman 2003).



Şekil 2.2 Yinelemelerde Gerçekleştirilen Etkinlikler,(Lorman, 2003)

Yinelemeli yöntemlerde amaç her yinelemenin sonunda kararlı, bütünleşmiş ve sınanmış bir sürüm oluşturmaktır. Yinelemelerin sonundaki sürümler çoğunlukla geliştirme ekibinin yararlanması için kullanılır. Bu sürümler müşterilere de sunulabilir. Son yinelemede oluşan sürüm beklenen üründür ve müşterilere verilir. (Bkz. Şekil 2.3) (Lorman, 2003)



Şekil 2.3 Yinelemeli ve Artışlı Geliştirme,(Lorman, 2003)

Sistem her bir yineleme sonunda yeni özelliklerin eklenmesi ile birlikte artışlı olarak büyür. Bu biçimde bir yaklaşım yinelemeli ve artışlı (Iterative and Incremental Development-IID) geliştirme olarak adlandırılır. IID, Scrum ve UP gibi çevik süreçlerin temelini oluşturmaktadır (Lorman, 2003).

IID yaklaşımında, birçok proje an az üç yinelemeden oluşmaktadır. Çağdaş yinelemeli yöntemlerde bir yinelemenin bir ile altı hafta arasında olması önerilmektedir. Her bir yinelemenin sonucunda ortaya çıkan yazılım sadece bir ön örnek değildir. Son olarak oluşması beklenen yazılımın bir alt kümesidir (Lorman, 2003).

2.3 Çeviklik Nedir?

Çeviklik kavramı, "Gerekli kaynakları ve süreçleri bir araya getirerek değişen ortama ve gereksinimlere sürekli olarak uyum sağlayabilme" olarak tanımlanabilir (Marjanovic, 2009).

"Nasıl çevik olunur?" sorusunun yanıtı düşünüldüğünden çok daha zordur. Çeviklik soyut ve göreceli bir kavramdır ve uygulanabilecek bir yazılım yöntemi değildir. Çevik yazılım geliştirme adım adım izleyebileceğimiz bir süreç değil, bir felsefedir. Çevik olabilmek için temel çevik ilkelerin uygulanması gerekmektedir. Dolayısıyla bir çevik süreç gerekir (Acar, 2009).

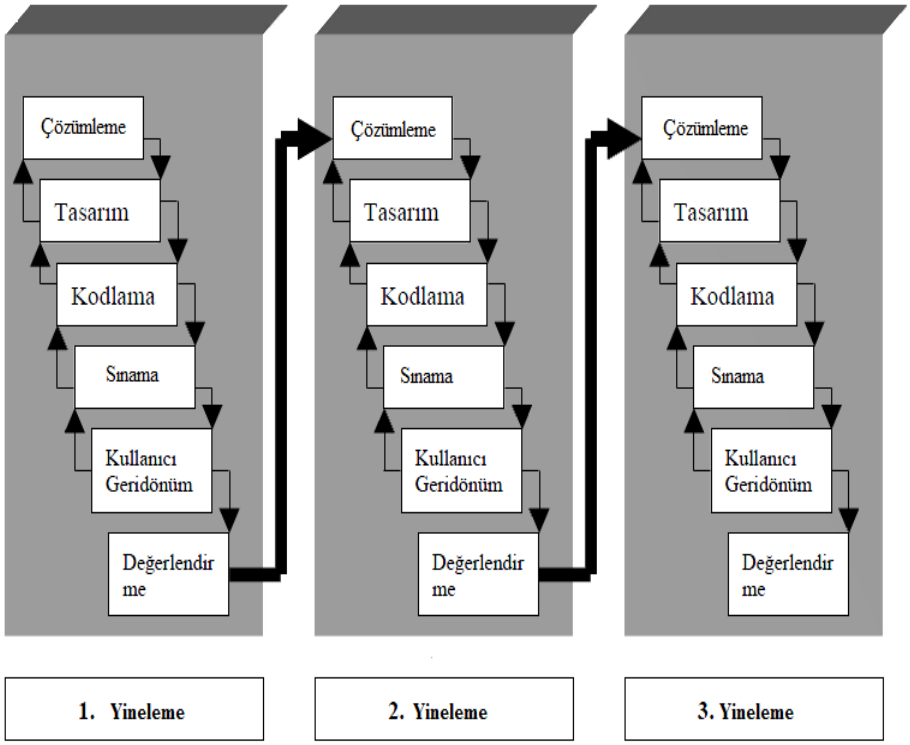
"Neden çevik olmalıyız?" sorusunun en önemli yanıtı değişen müşteri gereksinimleridir. Müşteriler gereksinimleri doğru bir biçimde anlatabilir, yeni

gereksinimler belirtebilir ve gereksinimler değişebilir. Çevik süreçler, yinelemelere ve müşteriden aldığı geribildirimlerle bağlı olarak bu değişimlere uyum sağlayabilir (Acar, 2009).

2.4 Çevik Süreç Nedir?

Çevik olmak bir çevik süreç ile anlam kazanır ve somutlaşır. Çevikliği somut duruma getiren çevikliğin içerdiği yazılım yöntemleridir (Acar, 2009).

Çevik süreçler yinelemeli geliştirmeye dayanmaktadır. Yapılacak olan işleri küçük artışlara ayırır ve en düşük düzeyde planlama yapar. Yinelemeler bir haftadan dört haftaya kadar değişebilen kısa dilimlerdir. Her yineleme kendi içerisinde planlama, çözümleme, tasarım, kodlama ve sınama aşamalarından oluşmaktadır. Dolayısıyla her bir yinelemenn küçük bir çağlayan model içerdiğini söyleyebiliriz (Bkz. Şekil 2.4) (Acar, 2009).



Şekil 2,4 Çevik Süreçlerde Yinelemeler,(Acar, 2009)

Her bir yineleme sonunda ürüne yeni işlevler eklenmeyebilir. Ancak amaç her yinelemede olduğunca az yanlışlı bir sürüm elde etmektir. Çevik yöntemlerde, yazılı belgelerden daha çok yüz yüze iletişimi seçilir. Bu özelliğinden dolayı diğer yöntemlere göre daha az belge oluşturulur. Süreci değerlendirmek için en önemli ölçüt çalışır bir yazılım elde edilmesidir (Lorman, 2003).

Çevik süreçler yazılım geliştirmeyi deneysel bir süreç olarak görür. Mühendislikte süreçler, tanımlı ve deneysel olmak üzere ikiye ayrılır. Tanımlı süreçler, başlangıcından tamamlanmasına dek aynı sonucu üreten süreçlerdir. Bu süreçlere otomobil üretimini örnek gösterebiliriz. Mühendisler arabanın üretimi için gerekli süreci tanımlar, parçaların birleştirilme sırasını ve birleştirme işleminde yer alacak çalışanların, makinelerin ve robotların düzenini belirler. Eğer üretim sırasında bu adımlara uyulursa, istenen nitelikte arabalar üretilebilir (Williams ve Cockburn, 2003).

Ancak yazılım geliştirme böyle bir süreç değildir. Geliştirim sırasında birçok değişiklik olabilir. Yazılım geliştirme süreci gibi deneysel süreçler, "denetle ve uyarla" biçiminde gerçekleştirilen kısa döngüler ve kısa aralıklarla alınan geri dönümlere dayanmalıdır (Williams ve Cockburn, 2003). Çevik süreçler bu yöntemleri uygulayarak değişikliklere uyarlanmaya çalışır. Değişimi onaylar ve bu değişikliklerle yaşamak için yazılım yöntemleri sunar (Acar, 2009).

2.5 Çeviklik Bildirisi

2001 yılının Şubat ayında yinelemeli ve çevik süreçlerle ilgilenen 17 kişi ortak bir zemin bulabilmek için ABD'nin Utah eyaletinde bir araya geldi. Yapılan çalışmaların sonucunda bu ekip kendilerini "Çevik Birlik" (Agile Alliance) olarak adlandırdı ve "Çeviklik Bildirisi" (Agile Manifesto) yayınlandı. Bu bildiri özgün durumu ile aşağıda yer almaktadır.

Çevik Yazılım Geliştirme Bildirisi

Bu bildiride biz yazılım geliştirmede en iyi geliştirme yollarını
ortaya çıkarmaya, uygulamaya ve başkalarına da bu yolları
uygulamaları için yardım etmeye çalıştık.
Bu çalışmada bulgularımız ise:

Kişiler ve etkileşim süreç ve araçlardan önce gelir,

Çalışır durumda olan yazılım ayrıntılı belgelemeden daha önceliklidir

Müşteri ile beraber çalışmak sözleşmelerden ve anlaşmalardan daha önceliklidir

Değişikliklere ayak uydurmak bir planı izlemekten daha önemlidir.

Bu başlıklar içerisinde koyu olarak belirtilen bölümlerin,
diğer bölümlere göre daha önemli olduğu vurgulanmıştır.

Kent Beck	James Grennin	Robert C. Martin
Mike Beedle	Jim Highsmit	Steve Mellor
Arie van Bennekum	Andrew Hur	Ken Schwaber
Alistair Cockburn	Ron Jeffrie	Jeff Sutherland
Ward Cunningham	Jon Ker	Dave Thomas
Martin Fowler	Brian Marick	

Bu bildiride 4 önemli konu üzerinde durulmuştur:

- **Kişiler ve etkileşim** araçlardan önce gelir
- **Çalışır durumda olan yazılım** ayrıntılı belgelemeden daha önceliklidir
- **Müşteri ile beraber çalışmak** sözleşmelerden ve anlaşmalardan daha önceliklidir
- **Değişikliklere ayak uydurmak** bir planı izlemekten daha önemlidir.

Bu başlıklar içerisinde koyu olarak belirtilen bölümlerin, diğer bölümlere göre daha önemli olduğu vurgulanmıştır (Sutherland, 2001).

2.6 Çevik İlkeler

Çeviklik bildirisi ile birlikte çeviklik için uyulması gereken temel ilkeler de belirlenmiştir:

- En önemli öncelik, zamanında ve sürekli teslimlerle müşteri memnuniyetini sağlamaktır. Çevik süreçlerin en önemli özelliği sıklıkla ve erken ürün teslimi yapılmasıdır. Projenin başlarında ürün temel özellikleriyle oluşturulmaya ve teslim edilmeye çalışılır. Sıklıkla yapılan teslimler son üründeki niteliği arttıracaktır (Martin, 2008). Yazılım tamamlanmamasına rağmen teslim edilen ürünler yeterli işlevselliği içeriyorsa, müşteri tarafından kullanılabilir. Ayrıca müşteri, hazırlanmış olan işlevsellikleri inceleyerek yapılmasını istediği değişiklik ve eklemeleri bildirebilir (Martin, 2008).

- Yazılımın ilerleyen aşamalarında bile değişiklik gereksinimleri anlayışla karşılanmalıdır. Çevik süreçler değişiklikleri müşterinin rekabet ortamındaki yararını korumak için kullanır. Çevik süreçlerde değişikliklerden korkulmaz. Tersine değişiklikler uygun biçimde karşılanır. Çünkü geliştirme ekibinin, benzer ürünler için piyasanın nasıl bir beklentisi olduğu konusunda bilgisi artar (Martin, 2008).

Çevik ekipler yazılımı olduğunca esnek tutarlar. Böylelikle gereksinimler değiştiğinde sistem en düşük düzeyde etkilenir. Değişikliklere uyarlanmış olunması, müşterinin isteklerinin gerçekleştirilmesini ve rekabet ortamı içerisindeki yararını korunmasını sağlamaktadır (Martin,2008), (Acar, 2009).

Olduğunca kısa aralıklarla (birkaç haftadan birkaç aya kadar) çalışan ürünler teslim edilmelidir. Amaç olduğunca erken ve kısa aralıklarla çalışan ürünlerin teslim edilmesini sağlamaktır. Çok sayıda belge veya planlama üretilmesi önemli değildir. Bu tür belgelerin teslim edilmesi kritik değildir. Her bir yineleme sonunda geliştirilen ürünler müşteriye teslim edilir. Böylelikle müşteri tamamlanmış bölümleri kullanarak geri dönümler yapabilir (Martin,2008), (Acar, 2009).

Proje süresince, geliştirici ve müşterilerin birlikte çalışması gerekir. Çevik olabilmenin en önemli koşullarından biri geliştiricilerin, müşterinin ve diğer paydaşlarının güçlü bir iletişim kurmalarıdır. Dolayısıyla proje ekibi ve müşteriler birlikte çalışır. Programdan beklentileri en iyi müşteriler bilebileceği için onlardan alınacak geri dönüşler oldukça önemlidir (Martin,2008), (Acar, 2009). Geliştiriciler, müşteri tarafından belirtilen gereksinimlerin olurluğunu araştırırken, her gereksinim için gerçekleştirim süresi değerlenir. Bu doğrultuda müşteri, kendi tanımlamış olduğu gereksinimlere bir öncelik sırası vererek hangi gereksinimlerin öncelikli olarak gerçekleştirilmesi gerektiğini belirler. Geliştiriciler, bu konularda müşteriye yardımcı olarak, gereksinimlerin daha somut duruma getirilmelerini sağlarlar. Bu gereksinimler genellikle birkaç tümceden oluşan kullanıcı öykülerine (user story) dönüştürülür (Lorman, 2003), (Acar, 2009).

- İyi projeler özendirmesi (motivation) yüksek bireyler çevresinde kurulur. Ekip bireylerine gerekli destek verilmeli, istekleri karşılanarak proje ile ilgili ekiplere güvenilmelidir. Çevik projelerde başarı için en önemli etken insandır. Süreç, ortam, yönetim gibi diğer etkenler ikinci plandadır ve insan üzerinde olumsuz bir etki yaratıyorlarsa değişiklik yapılması olasıdır (Martin,2008). Geliştiricilere duyulan güvenin onlara hissettirilmesi özendirmeyi (motivation) arttıracaktır. Sorumluluk almaları sağlanmalı ve öz güvenleri pekiştirilmelidir (Acar, 2009).

- Geliştirme ekibi içerisindeki en verimli ve etkin bilgi akışı yüz yüze iletişimle gerçekleşebilir**.** Çevik projelerdeki en önemli iletişim biçimi yüz yüze iletişimdir. Çünkü bilgi aktarımının en az bozuntuya uğradığı yöntem budur. Yüz yüze konuşmalar güveni arttırır ve yanlış anlamaları ortadan kaldırır. Belgeler oluşturulabilir ancak tüm projeyi belgelemek gibi bir girişim yoktur. Çevik geliştirme ekiplerinden belirtilimler (specification) yazılı duruma getirmesi beklenmez. Ancak, ekipler gerekli gördüğü durumlar için belgeleme yapar (Martin,2008), (Acar, 2009).

- Çalışan yazılım süreç için en önemli ölçüdür. Çevik projeler üretilen yazılımın çalışabilirliğine göre değerlendirilir. Süreç içerisinde bulunulan

aşama, üretilen belgeler ve oluşturulan altyapının ölçümlerde etkisi yoktur. Eğer yerine getirilmesi gereken işlevselliğin %30'u tamamlanmış ise proje %30 oranında tamamlanmıştır (Martin, 2008).

- Çevik süreçler sürdürülebilir geliştirmeyi destekler. Destekleyiciler, geliştiriciler ve kullanıcılar belirli bir çalışma biçimi izlemelidir. Geliştirme ekibi tam hızla geliştirmeye başlayıp süreç boyunca bunu iyileştirmeye çalışmaz. Tersine hızlı ancak arttırılabilir bir çalışma biçimi ile geliştirmeye başlanır (Martin, 2008). Çok hızlı davranmak sorunlara yol açabilir. Çevik ekipler kendi hızlarını kendileri ayarlar. Çok yorulmalarına izin vermezler. Bir sonraki günün gücünü bugün biraz daha iş yapabilmek için harcamazlar (Martin, 2008). Güçlü teknik alt yapı ve tasarım çevikliği arttırır. Yüksek nitelik, hızlı ve etkili yazılım üretilmesi için anahtar etkendir. Hızlı ilerleyebilmek için yazılım net ve güçlü olmalıdır. Çevik ekipler nitelikli yazılım geliştirmeye çalışırlar. Gereksiz kodlar üretip daha sonra bunları temizlemek zorunda kalmazlar (Martin, 2008).

- Kolaylık temeldir. Çevik ekipler amaçlarına ulaşmak için en kolay yolu seçerler. Yarın oluşabilecek sorunlar için bugün çalışma yapmazlar. Bugün yapabilecekleri işi kolay ve nitelikli bir biçimde gerçekleştirip, ilerleyen günlerde sorun oluşması durumunda en az etkilenecek biçimde esnek tutarlar (Martin, 2008).

- En iyi mimariler, gereksinimler ve tasarımlar kendi kendine düzenlenebilen ekiplerden çıkar. Çevik ekipler kendi kendine düzenlenen ekiplerdir. Ekip içerisindeki bireylerin sorumlulukları ekip dışından belirlenmez. Sorumluluklar ekip içerisinde görüşülür ve bunları yerine getirebilmek için en iyi yol seçilir (Martin, 2008). Çevik ekipler tüm projede birlikte çalışırlar. "Gereksinimlerden, tasarımdan ve gerçekleştirimden sorumlu bireyler" gibi bir ayrım yapılmaz (Martin, 2008).

- Düzenli aralıklarla ekipler kendi yöntemlerini gözden geçirerek verimliliği arttırmak için gerekli iyileştirmeleri yaparlar. Çevik ekiplerde bilgi alış verişi ve sürekli öğrenme çok önemlidir. Belirli zaman aralıklarıyla bir araya gelerek düşünce alış verişinde bulunurlar ve çalışma yöntemlerini sorgularlar. Edinilen deneyimler doğrultusunda düzenlemeyi, kuralları ve

ilişkileri değiştirebilir. Çünkü çevik ekipler ortamın sürekli olarak geliştiğini bilir ve çevik kalabilmek için bu değişimlere uyum sağlar (Martin,2008) (Acar, 2009).

2.7 Tarihçe

Çevik süreçler temel olarak yinelemeli ve artışlı geliştirme yöntemlerine dayanmaktadır. Birçoğumuz yinelemeli ve artışlı geliştirme yöntemlerini çağdaş teknikler olarak düşünsek de, bu yöntemler 1950'li yıllara dek uzanmaktadır. Birçok kişinin bunu bilmesine rağmen, IID yöntemlerinin geçmişini tam olarak bilmeyen kuruluşlar da vardır (Shore, 2008).

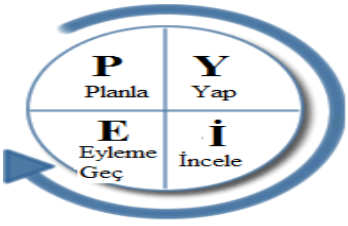
1970'ler ve 1980'lerde IID çalışmaları ile ilgili birçok örnek bulunmaktadır. Bu dönemler, IID yöntemlerinin en az bilinen ancak en hareketli dönemleridir.

2.7.1 1970 öncesi

IID, Bell Laboratuarları'nda çalışan nitelik uzmanı Walter Shewhart'ın 1930'lu yıllardaki çalışmalarına dek dayanmaktadır. Walter Shewhart, niteliğinin iyileştirilmesi için "planla-yap-incele-eyleme geç" (PYİE) ("plan-do-study-act"(PDSA)) döngüsünü ileri sürmüştür. PYİE herhangi bir sürecin iyileştirilmesi için kullanılabilir ve 4 sürekli adımdan oluşmaktadır:

- Planla: Sürecin niteliğini iyileştirmek için planlama yapılır
- Gerçekleştir: Planlananlar gerçekleştirilir. Öncelikle dar bir kapsamla sınırlandırılmalıdır.
- Çalış: Planın başarısını ölçmek için geri dönümler ve sonuçlar değerlendirilir.
- Davran: Süreci iyileştirmek için gerekli değişikliklere karar verilir.

Bu döngü "PYİE döngüsü, Deming wheel ve Deming Çarkı" olarak da adlandırılmaktadır (Bkz. Şekil 2.5) (Larman, 2003).



Şekil 2.5 PYİE Döngüsü

1940'lı yılların başlarında, W.Edward Deming, PDSA yöntemini daha güçlü bir hale getirebilmek için geliştirmeler yapmıştır (Larman, 2003).

"X-15 hypersonic jet" projesi 1950'li yıllarda IID yönteminin uygulanması için bir kilometre taşı olmuştur. X-15 bir yazılım projesi olmamasına rağmen dikkate değer, önemli bir projedir. Çünkü bu projedeki bireylerin bazıları 1960 yılların başlarında NASA tarafından geliştirilen ve IID yöntemlerinin kullanıldığı "Mercury" adlı yazılım projesine katkıda bulunmuştur (Larman, 2003).

Mercury, yarım günlük kısa yinelemelerle gerçekleştirilmiş bir projedir. Yinelemelerde tüm değişikliklerle ilgili teknik gözden geçirmeler yapılmış ve UP yönteminin deneyimlerden biri olan "Önce-Sınayarak Geliştirmek" (test-first development) yaklaşımı uygulanmıştır. Bu yaklaşıma göre her bir artış için sınamalar planlanmış ve yazılmış, daha sonra bu sınamaları geçecek kodlar geliştirilmiştir (Larman, 2003).

2.7.2 1970'ler

IID ile ilgili ilk dönemlerdeki önemli çalışmalar, uzay ve savunma sistemleriyle ilgili çalışmalar yürüten IBM FSD (Federal Systems Division) içerisindeki görevleri süresince, Mike Dyer, Bob McHenry ve Don O'Neill liderliğinde gerçekleştirilmiştir (Larman, 2003).

FSD içerisinde ilk önemli IID projesi 1972 yılında geliştirilmiştir. Bu proje ilk Amerikan Trident denizaltısı için geliştirilmiş bir komuta ve kontrol sistemidir. Dyer, McHenry ve O'Neill proje yöneticiliğini gerçekleştirmiştir. O'Neill proje içerisinde IID kullanımını tasarlamış ve planlamıştır. Bu çalışmadan dolayı O'Neill "IBM Outstanding Contribution – IBM Üstün Katkı" ödülünü kazanmıştır (Larman, 2003).

Geliştirilmekte olan bu denizaltı projesi oldukça önemliydi ve bir gün bile gecikmemesi gerekiyordu. FSD, geciken her gün için $100,000 ceza ile karşı karşıya kalacaktı. Proje, her biri altı ay uzunluğunda dört yinelemeli süreç biçiminde düzenlenmiştir (Bu yineleme süreçlerinin süreleri günümüzde önerilen sürelere göre daha uzundur). Projedeki gereksinimlerin geri dönüşlere bağlı olarak

gelişip değişmesine rağmen IID, geliştirme sürecindeki karmaşıklık ve çekinceleri yönetmek için bir yol olarak görülmüştür (Larman, 2003).

1972 yılında FSD'nin rakibi olan TRW tarafından IID yöntemlerinin uygulandığı başka önemli bir proje geliştirilmeye başlanmıştır. Bu proje ile balistik füze savunma sistemleri için gerekli yazılım geliştirilmiştir. 1972 yılının Şubat ayında başlayan proje beş yinelemeye ayrılmıştır ve birkaç yıl içerisinde tamamlanmıştır (Larman, 2003).

IID yöntemlerinin uygulanmasındaki başarılardan bir diğeri yine FSD tarafından Amerikan donanması için geliştirilen "Light Airborne Multipurpose System - Çokamaçlı Hafif Havanakli Sistemi" (LAMP) projesidir. Bu proje helikopter gemileri için geliştirilmiştir. Birer aylık 45 yinelemeli süreç sonucunda tamamlanmıştır (Larman, 2003).

FSD, 1977 ve 1980 yılları arasında geliştirdiği NASA uzay mekiği yazılımı "the Primary Avionics Software System - Temel Havacılık Elektroniği Yazılım Sistemi" ile IID yöntemlerinin kullanıldığı başarılı bir çalışma daha gerçekleştirmiştir. Proje, her biri ortalama 8 hafta süren 17 yinelemeli süreç ile 31 ayda tamamlanmıştır (Larman, 2003).

1975 yılında Vic Basili ve Joe Turner yayınladıkları makale ile derleyici geliştiriminde IID yöntemlerinin uygulamasını anlatmıştır. Tom Gilb, 1976 yılında yayınladığı "Software Metrics – Yazılım Ölçümleri" adlı kitabında IID ile birlikte "Değerlendirme ve Yazılım Ölçütleri " kavramlarını anlatmıştı (Larman, 2003).

2.7.31980'ler

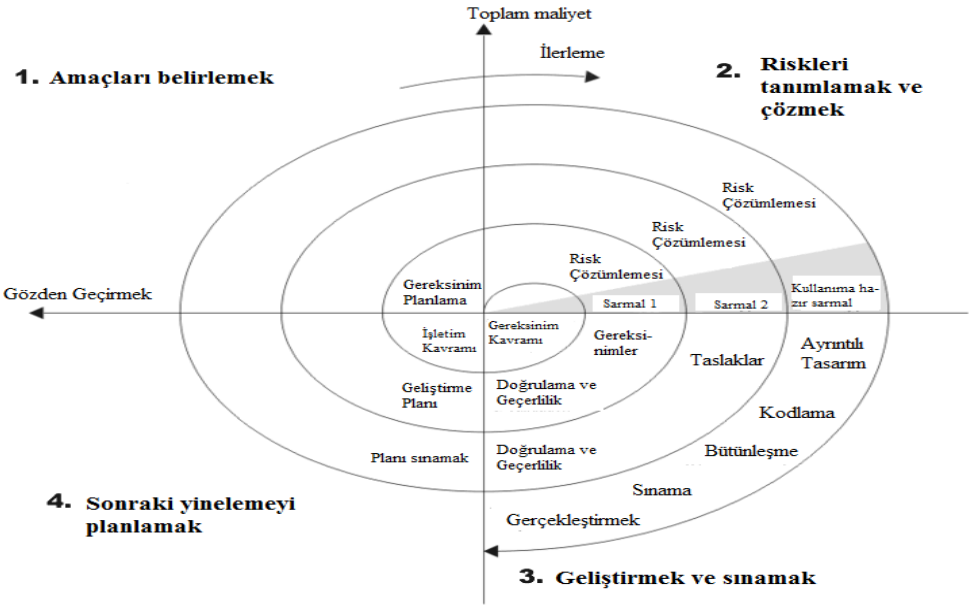
1980 yılında Weinberg, "Adaptive Programming: The New Religion - Uyarlanabilen Programlama: Yeni İnanç" adlı bir makale yazmış ve bu makale Avustralya'nın "Computer World - Bilgisayar Dünyası" adlı dergisinde yayınlanmıştır. Makalede özet olarak, geliştirmenin küçük artışlar durumunda ve kullanıcıdan gelen dönüşlere bağlı olarak yinelemeli olması düşüncesi yer alıyordu. Bu makaleden bir yıl sonra, Tom Gilb, "Evolutionary Development - Evrimsel Geliştirme" ile ilgili daha ayrıntılı bir makale yazmıştır (Larman, 2003).

1981 yılında, Daniel McCracken ve Michael Jackson, William Cotterman'ın yazısında yinelemeli ve artışlı geliştirmeyi destekleyen ve çağlayan modeli ile karşılaştıran bir bölüm yazmışlardır. "A Minority Dissenting Position - Azınlık Karşıt Durumu" adlı bu bölümde IID yaklaşımının önemini vurgulamışlardır (Larman, 2003).

Bu yıllarda IID kullanılarak geliştirilen büyük uygulamalardan biri, 1982 yılında geliştirilmiş olan komuta ve denetim projesidir. 1983 yılında Grady Booch, yayınladığı "Software Engineering with ADA – ADA ile Yazılım Mühendisliği" adlı yazısında, nesneye dayalı sistemler için yinelemeli bir süreci anlatmıştır (Larman, 2003).

1984 ve 1988 yılları arasında, "Magnavox Electronic Systems – Magnavox Elektronik Sistemler" tarafından Amerikan ordusu için bir komuta ve denetim sistemi geliştirilmiştir. IID kullanılarak ADA ile geliştirmiş olan bu proje yaklaşık 1.3 milyon satır kod içermektedir. Proje toplam beş yinelemeli süreç tamamlanmıştır. 1980'li yıllarda IID yaklaşımı kullanılarak geliştirilmiş önemli ve büyük projelerden biri de Amerikan ve Kanada hava trafik denetleme sistemidir (Larman, 2003).

1985 yılında Bary Boehm tarafından yazılan "A Spiral Model of Software Development and Enhancement – Yazılım Geliştirmek ve Genişletmek için Bir Sarmal Model" adlı makale, IID alanındaki yazılar için bir dönüm noktası olmuştur. Bu makale ile anlatılan Sarmal Model, geliştirme döngülerinin çekincelere göre yönlendirildiği ilk yaklaşım değildir. IBM FSD içerisinde benzer düşünceler daha önce uygulanmıştır. Ancak Sarmal Modelin önemi, çekince yönelimli yineleme kavramından ve her bir yinelemede çekince değerlendirme aşamasının olmasından kaynaklanmaktadır (Bkz. Şekil 2.6) (Larman, 2003).



Şekil 2.6 Sarmal Geliştirme Modeli

1986 yılında Frederick Brooks, IID yöntemlerinin yararlarından anlattığı "No Silver Bullets – Gümüş Kurşun Yoktur" makalesini yayınlamıştır.

Yine 1986 yılında David Parnas ve Paul Clements "A Rational Design Process: How and Why to Fake It – Akla Yatkın Bir Tasarım Süreci: Nasıl ve Neden Sahte Olur " adlı makaleyi yayınlamıştır. Bu yazıda, çağlayan modelinin kusursuz bir model olduğuna inandıklarından ancak bu modelin uygulamada kullanışsız olduğunu anlatmıştır (Larman, 2003).

1987 yılında TRW, "Command Center Processing and Display System Replacement (CCPDS-R) – Kumanda Merkezi İşleme ve Sunma Yedeği" adlı sistemin geliştirilmesi için 4 yıllık bir projeye başlamıştır. Bu projede IID yöntemleri uygulanmış ve süreç her biri ortalama altı ay süren altı yinelemeli sürece ayrılmıştır. Daha sonra Walker Royce bu projeyi 60 sayfalık bir yazıyla anlatmıştır (Larman, 2003).

1988 yılında Glib tarafından IID ile ilgili ilk kitap yazılmıştır. "Principles of Software Engineering Management - Yazılım Mühendisliği Yönetimi İlkeleri" adlı bu kitap, IID ile ilgili önemli gelişmelerden biridir (Larman, 2003).

2.7.41990'lardan Günümüze

Özellikle 1990'lı yıllarına ikinci yarısında yazılım geliştirmede IID yaklaşımının önemi daha çok anlaşılmaya başladı. IID ile ilgili birçok kitap ve makale yazıldı (Larman, 2003).

1990'lı yılların başlarında bir birlik tarafından "Canadian Automated Air Traffic Control System(CAATS) – Kanada Otomatik Hava Trafiği Denetim Sistemi" adlı bir proje geliştirilmeye başlanmıştır. Proje geliştirilirken çekince yönelimli bir IID yöntemi kullanılmış ve altı aylık yinelemeli süreçler uygulanmıştır (Larman, 2003).

1990'lı yılların ortalarında Rational Corp. İçerisindeki katılımcı ve müşteriler, "Rational Unified Process – Akla Yatkın Bileştirme İşlemi" adlı yöntemi geliştirmiştir. 1997 yılında, büyük bir lojistik sistemi geliştirmek için sürdürülen proje, Çağlayan Modelinin uygulanmasından dolayı başarısızlıklar ile karşı karşıya kalmak üzereydi. Peter Coad ve Jeff De Luca'nın işbirliğiyle, geliştirme ekibi projeyi başarısızlıktan kurtardı ve bir IID projesi olarak geliştirdi. Daha sonra De Luca, Feature-Driven Development (FDD) süreç modelini geliştirdi. Bu süreçte Peter Coad'ın verdiği düşüncelerde bulunmaktadır (Larman, 2003).

1998 yılında Standish Ekip tarafından "CHAOS: Charting the Seas of Information Technology – Bilişim Sistemlerinin Deniz Çizimleri" adlı bir bildiri yayınlanmıştır. Bu bildiride, projelerdeki temel başarısızlık nedenlerini bulabilmek için 23,000 proje çözümlenmiştir. Bildiriye göre en önemli başarısızlıklar Çağlayan Modelinin uygulanmasından kaynaklanıyordu (Larman, 2003).

2001 yılında, DSDM, UP, Scrum, FDD vb. modern IID süreçlerinde uzman 17 kişi ortak bir zemin oluşturabilmek için Utah'ta bir araya geldi. Bu görüşmenin sonucunda "Agile Alliance – Çevik Birlik" ortaya çıktı ve çevik süreçlerin temel İlkeleri oluşturuldu (Larman, 2003).

2.8 Çevik Süreçlerde Proje Yönetimi

Yazılım geliştirme projelerinin yönetilmesi çekinceli bir iştir. Geliştirilen yazılımlar başarısız olmakta, müşteriye zamanında verilememekte veya bütçeler aşılmaktadır. Bu başarısızlıklar için birçok neden olmasına rağmen en önemli neden değişen gereksinimlerin yönetilememesidir (Augustine, 2005).

Geleneksel yazılım geliştirme yöntemleri doğrusal ve sıralıdır. Dolayısıyla bu yöntemlerin uygulandığı projelerdeki yönetim etkinlikleri, iyi bilinen ve değişmeyen gereksinimleri olan sistemler için daha etkili olacaktır. Ancak geliştirilen yazılımların birçoğunda bu özellikler olmadığı için. Çoğu zaman gereksinimler değişebilmekte, küçük değişiklikler büyük ve karmaşık etkilere yol açabilmektedir. Dolayısıyla bu çeşit değişken sistemler için geleneksel yöntemlere dayanan yönetim etkinlikleri uygulanması uygun değildir (Augustine, 2005).

Çevik süreçlerde, geleneksel yöntemlerin tersine proje yöntemine gerek olmadığı ve çevik süreçlerin kendi kendini yönetebileceği izlenimi oluşmaktadır. Çevik süreçlerin özellikleri(kendi içerisinde düzenlenmiş olmuş ekipler, hızlı ilerleyiş, daha az planlama vb) bu izlenime yol açmaktadır. Ancak çevik süreçler için de proje yönetimi gerekir (Cohn ve Schwaburn, 2003).

Birçok süreçte olduğu gibi çevik süreçlerin de başarısı için doğru ve etkin bir proje yönetimi sağlanmalıdır. Eğer yönetim olmazsa geliştirme ekipleri projenin geliştiriminde yanlış bir yol izleyebilir. Ekiplerin içerisinde yer alan bireyler yetenek ve kişisel özelliklerine göre düzenlenmelidir, ürün müşteriye yeterli aralıklara verilemeyebilir. Bu sorunlarla benzer birçok sorun oluşabilir (Cohn ve Schwaburn, 2003).

Çevik süreçlerde proje yönetimi, geleneksel yöntemlerin tersine projenin başarısızlığını en pahalı durum olarak görür. Bu başarısızlıklar takvimin ertelenmesi, istenen işlevselliklerin sağlanamaması ve nitelik yetersizlikleri gibi birçok nedene bağlı olabilir. Çevik proje yönetimi, değişiklikleri önlenmesi gereken değil yönetilmesi gereken durumlar olarak değerlendirir. Planlama, tasarım ve belgelemeyi zaman kayıplarını önlemek için en düşük düzeyde tutar. Daha çok çalışan özellikleri olan ürünlerin müşteriye hızlı bir biçimde verilmesi amaçlamaktadır (Karlesky, 2008).

Çevik proje yönetiminde anlatılan temel özelliklerin sağlanabilmesi için uygulanması gereken bazı ilkeler olacaktır. Çevik Birliğin kurucularından ve Uyarlanabilen Yazılım Geliştirme (Adaptive Software Development) yönteminin yaratıcısı olan Jim Highsmith, bu ilkelerle ilgili olarak 9 başlık tanımlamıştır:

- Kullanıcıya işe yarar bir ürün verilmeli ve ürünü nasıl değerlendirdikleri denetlenmeli,
- Proje ile ilgili paydaşlara önem verilmelidir,
- Liderlik ve işbirliğine dayalı bir yöntem kullanılmalıdır,
- Yeterli ve işbirliği ile çalışan ekipler oluşturulmalıdır,
- Ekiplerin karar verebilmesine olanak sağlanmalıdır,
- Değişmeyen uzunlukta yinelemeli süreçler kullanılmalıdır,
- Uyum sağlayabilme yeteneği desteklenmelidir,
- Teknik konularda üstünlük desteklenmelidir,
- Sürece uyum sağlayacak etkinlikler yerine ürün verilmesini sağlayacak etkinlikler uygulanmalıdır (Larman, 2003).

Jim Highsmith'in yanı sıra, UP yöntemine dayalı projelerde deneyimli yöneticiler olan Augustine ve Woodcock, çevik proje yönetimi ile ilgili 6 uygulama tanımlamıştır:

- Yol Gösterici Öz görüş**:** Proje için yol gösterici bir Öz görüş oluşturulmalı ve sürekli olarak geliştirilmelidir.
- Ekip Çalışması ve İşbirliği: İşbirliği ve ekip çalışması kolaylaştırılmalıdır.
- Kolay Kurallar: Scrum ve UP gibi ekibe yol gösterici uygulamalar oluşturulmalı ve geliştirilmelidir.
- Açık Bilgi: Proje ve diğer konulardaki bilgilere herkesin erişebilmesi sağlanmalıdır.
- Daha Az Karışma: Kendi kendini yöneten bir ekip içerisinde birden ortaya çıkabilecek durumları yönetmeye yetecek biçimde denetim sağlanmalıdır.
- Çevik Dikkat: Öz görüş geliştirilmeli, kurallar izlenmeli veya kurallara uyum sağlanmalı, kişiler dinlenmelidir (Larman, 2003).

Çevik süreçlerde proje yöneticisi, yönetimden çok ekip içerisindeki denetim ve planlamayı sağlamaktan sorumludur. İş bölümü, zamanlama ve değerleme ile

ilgili çalışma yapmaz. Bu çalışmalar ekip içinde gerçekleştirilir. Proje yöneticisi kişilere ne yapması gerektikleri konusunda yönlendirme yapmaz, ekip içerisindeki rolleri ve sorumlulukları ayrıntılı bir biçimde belirlemez (Larman, 2003).

Proje yöneticisi, geliştirme ekibine koçluk yapar, gerekli kaynakları sağlar, engelleri ortadan kaldırır ve çevik İlkelere uyulmasını sağlar (Larman, 2003).

2.9 Çevik Süreçlerde Geliştirme Ekipleri

Çevik ekiplerin en önemli özellikleri kendi kendilerini düzenleyebilmeleri ve güçlü bir işbirliği olmasıdır. Sürecin ilerleyişine göre kendilerini düzenler.

Çevik süreçlerin özelliklerinden dolayı geliştirme ekibinin hiç planlama yapmadığı gibi yanlış bir izlenim oluşmaktadır. Ancak bu izlenim çevik ve artışlı süreçlerdeki planlama yaklaşımını tam olarak bilmemekten kaynaklanmaktadır. Çevik ekiplerde en az diğer süreçlerdeki gibi planlama yapmaktadır. Çevik ekiplerin farkı planlamayı projenin başında değil, yinelemelere yayarak tüm proje içerisinde gerçekleştirmeleridir (Cohn ve Schwaburn, 2003).

Çok değişik biçimde çevik yazılım geliştirme yöntemleri olsa da çevik ekipler genel olarak benzer özellikler içerir. Çevik ekiplerin diğer geliştirme ekiplerinden farkını aşağıdaki gibi özetleyebiliriz:

- Kısa Aralıklarla Ürün Çıkarırlar: Ekipler iki veya dört haftada bir, müşteriye çalışan bir işlevsellik verir. Ürünün verilme zamanı değiştirilemez ancak zamanında vermek için, verilecek işlevsellik değeri değiştirilebilir. Bu yüzden planlanan işlevselliğin hangi bölümlerini verilen süre içinde üretilebildiğini belgelemek önemlidir.
- Çoğunluklar Müşterileriyle Birlikte Çalışırlar**:** Olasılığı olan her yerde müşteri yazılım geliştirme sürecinin içinde olur. Müşteri uygulama ile ilgili her konu üstünde karar verme yetkisine ve sorumluluğu olur. Böyle bir durumda çevik ekipler, müşteriden çok daha sık geri besleme ve düşünce alabiliyor demektir.
- Erken ve Sık Yapılan Sınamaların Önemini Bilirler: Çevik ekipler sürekli olarak kodlarını birleştirir ve her durumda sınamaya çalışırlar. Sınamaya yönelik uygulama geliştirme daha çok yanlışın üretim ortamına geçmesini engeller.

- Müşteri İsteklerinin Değişebileceğini Kabul Ederler: Geliştirme ekibi her yinelemeli süreç başında müşteri isteklerini önem sırasına göre düzenler ve çalışılan yinelemenin kapsamını belirler. Ekip, müşteri istek dizinini zamanla değişebileceğini kabul eder ve bütün isteklerin önceden tanımlanamayacağını anlar. Önem sırası belirleme işlemi, isteğin üreteceği değere göre yapılır (Cockburn ve Highsmith, 2001).

Çevik ekiplerin yukarıda belirtilmiş olan ölçütleri yerine getirebilmeleri için ekip içerisindeki her bireyin yeterli özeliklerde olması gerekmektedir. Çevik ekiplerde bulunması gereken bireysel özellikler aşağıdaki gibi özetlenebilir:

- Ekip bireyleri ortalama bir çözümsel ve teknik bilgi düzeyinde olmalıdır,
- Ekip içerisinde birbirinden değişik görevleri olan bireylerin hepsi aynı amaca odaklanmış olmalıdır,
- Ekibin bireyleri, birbirleriyle, müşteriyle ve alan uzmanlarıyla işbirliği içinde olmalıdır,
- Ekip, koşulları göz önünde bulundurarak karar verebilme yeteneği olmalıdır,
- Ekip, belirsizlik durumunda da sorun çözebilme yeteneği de olmalıdır,
- Ekip bireyleri, karşılıklı güven ve saygıya dayalı iletişime önem vermelidir,
- Zaman ve kaynak kısıtlarını düzenleyebilmelidir (Altın, 2007).

2.10 Çevik Yöntemlerin Yararları

- Çevik projelerde müşteri gereksinimleri ve bu gereksinimlerin karşılığı olan program birlikte gelişir. Müşteri projenin durumuna her zaman karışabilir. Bu durum özellikle uzun süreli projelerde önem taşımaktadır. Çünkü çoğu zaman proje başlangıcında müşteri kendi gereksinimlerini tam olarak bilemeyebilir ve zaman içinde oluşan ilk ön ürünler müşterinin nasıl bir sistem istediğini daha iyi anlamasını sağlar. Projedeki geri dönüşlerle, müşteri gereksinimleri her zaman değişikliğe uğrayabilir (Altın, 2007).

- Bu modelde geri dönüşler önemli bir rol oynamaktadır. Çeşitli aşamalarda sağlanan geri dönüşler ile projenin hangi durumda olduğu saptanabilir. Programcılar hazırladıkları sınamalar aracılığıyla geri besleme sağlayarak,

gerçekleştirdikleri bileşenlerin hangi durumda olduklarını bildirir. Sürekli bütünleştirme yapılarak programın hangi durumda olduğu geri besleme olarak elde edilir. Müşteriye, kısa aralıklarla programın yeni sürümü sunularak geri dönüşler sağlanır (Altın, 2007).

• Proje başlangıcında ayrıntılı belgeleme ve tasarım oluşturulmaz. Sınama Güdümlü Geliştirme (Test Driven Development - TDD) olarak çalışır ve oluşan tasarımı her yeni sınama ile gözden geçirirler. Eğer tasarım yetersiz kalırsa, gerekli tasarım değişikliklerini, ileri adımlardaki sınamaların çıkmaza girmesini engellemek için uygularlar (Altın, 2007).

• Her yineleme başlangıcında müşteri tarafından belirtilen gereksinimler çözümlenir ve gerçekleştirilecek olanlar belirlenir. Müşteri her gereksinim için bir öncelik sırası belirler. Öncelik sırası yüksek olan gereksinimler daha önce gerçekleştirilir. Her yineleme sonunda gerekli değerlendirmeler yapılarak oluşan sorunlar tartışılır ve yeniden oluşmalarını engellemek için gerekli önlemler alınır (Altın, 2007).

• Sınama güdümlü çalışıldığı için kod niteliği çok yüksek olur. Gün boyunca programcılar kendilerine değişik bir programcıyı ekip arkadaşı olarak seçerek (Pair Programming- Eş Programlama), gerçekleştirim yaparlar. Kısa bir süre sonra programcılar arasındaki teknik bilgi aynı düzeye gelir ve her programcı programın herhangi bir bölümünde çalışacak düzeyde bilgisi olur (Altın, 2007).

• Programcılar proje planlamasında etkin olarak yer alırlar. Gereksinimlerin belirlenmesinde müşteriye yardımcı olurlar ve zaman değerleme (valuation) bulunarak proje planlaması için gerekli verilerin oluşturulmasını sağlarlar. Bu durum programcılara belirli bir sorumluluk yükler. Kendisine güvenildiğini bilen ve sorumluluk taşıyabilen bir programcının öz güveni ve özendirme (motivation) artar (Altın, 2007).

• Çevik projelerde iyi bir çalışma ortamının ve hızını oluşturulması fazla çalışmaya gidilmesini engeller. Gerektiğinde fazla çalışma yapılabilir ancak bu durum bir kural durumuna gelmemelidir. Tersine tüm ekibin özendirmesini (motivation) olumsuz etkilenir. Yanlışlar genelde isteksizce

yer alınan fazla çalışma sürelerinde ortaya çıkar. Proje çalışanları ek çalışma süresi olmayan iş günlerinde daha verimli olurlar (Altın, 2007).

- Müşteriye kısa aralıklarla, çalışabileceği bir sürüm sunulur. Program tamamlanmamış olsa bile müşteri hazır bölümleri kullanabilir. Böylece müşterinin yaptığı yatırımın hızlı bir biçimde geri dönüşü sağlanır (Altın, 2007).
- Geliştiriciler ve müşteri arasında sürekli iletişim vardır. Geliştiriciler soruları ve sorunları müşteri ile paylaşarak kısa sürede çözüm üretebilirler. (Altın, 2007).
- Müşterinin piyasadaki değişikliklere ve bununla birlikte rekabet ortamına ayak uydurabilmesi önemlidir. Çevik süreç hızlı tepki göstererek, bu değişikliklere ayak uydurulmasını sağlar (Altın, 2007).

Version One tarafından yıllık olarak gerçekleştirilen Çevik Geliştirme Durumu "State of Agile Development" araştırması da çevik süreçlerin sağladığı yararları göstermektedir.

2008 yılında yapılan üçüncü araştırmaya göre katılımcıların

- %89'unda üretkenlikte artış,
- %84'ünde yazılım yanlışlarında azalma,
- %82'sinde ürünlerin piyasaya verilme süresinde azalma,
- %66'sında maliyetlerde azalma görülmüştür (Altın, 2007).

2.11 Çevik Süreçler Ne Zaman Kullanılmalıdır?

Eğer kurum içerisinde Çağlayan Modeli veya başka bir model uygulanıyor ve başarı sağlanıyorsa bu süreçlerden vazgeçip çevik süreçlere yönelmek yanlış bir davranış olabilir. Çevik bir sürece geçiş yapılmak isteniyorsa şu soruya yanıt aranmalıdır:

"Çevik süreçler bize ne kadar yararlı olacak?"

Eğer bu soruya tam olarak yanıt verilebiliyorsak çevik süreçlere geçiş doğru bir karar olacaktır (Williams ve Cockburn, 2003).

Çevik süreçlere geçilmeden önce geliştirilen projenin özelliklerinin de değerlendirilmesi gerekecektir. Çevik süreçlerin uygun olduğu projelerinin özelliklerini aşağıdaki gibi özetleyebiliriz:

- Kritik amacı olmayan projeler,
- Geliştiricilerin deneyimli olduğu projeler,
- Gereksinimlerin sık sık değiştiği projeler,
- Geliştirme ekiplerinin çok büyük olmadığı projeler (20 kişiden küçük ise) (Holler, 2006).

2.12 Geleneksel Süreçler ile Çevik Süreçlerin Karşılaştırılması:

2.12.1 Geleneksel Süreçler

Geleneksel yazılım yöntemleri proje başlangıcında müşterinin tüm gereksinimlerini belirtir veya belirttiğini düşünür ve sonrasında gerçekleştirir. Yazılım süreci zaman içinde değişikliğe uğrayan müşteri gereksinimlerine ayak uyduracak biçimde düzenlenmiştir. Müşteri tarafından istenen değişiklikler hoş karşılanmaz. Bu nedenden dolayı geliştirilen yazılım sisteminin müşteri gereksinimlerini %100 karşılama olanağı yok gibidir (Acar, 2009).

Müşteri istekleri doğrultusunda yapılmak zorunda kalınan değişiklikler proje maliyetlerini yükseltir, çünkü bu beraberinde kolay olmayan yapısal değişiklikler getirebilir (Acar, 2009).

Proje yöneticilerinin programcılardan insanüstü beklentileri, projenin büyük bir bölümünün sorumluluk ve riskini omuzlarında taşıyan bu bireylerin fazla mesai yapmalarına ve yorulmalarına neden olur. Özendirmesi düşük olan programcıların yaratıcı yönleri azalmakta ve kodun niteliği buna orantılı olarak düşmektedir. Bu programlardaki yanlış oranının artması anlamına gelmektedir(Acar, 2009).

Projelerde ekip çalışması ve bireyler arası iletişimin güçlendirmesini desteklenmez. Her programcı yazılımını yaptığı program parçasından sorumlu olduğu için, projeden ayrılan programcılar proje için bir riske dönüşür (Acar, 2009).

Proje başlangıcında müşteri gereksinimleri en son ayrıntısına kadar belirtilir. Ayrıca yazılım sistemi için teknik mimari ve uygulanacak tasarım belirlenir.

Bunlar proje başlangıcında çok süre kaybedilmesine neden olur. Ayrıca bir süre sonra tasarımın, gerçekleştirilen gereksinimleri karşılayamaması ve yapılan her değişiklik sonucunda tasarım çıkmaza girdiği görülür (Acar, 2009).

Oluşturulan ve uygulanan proje planı bir süre sonra geçerliliğini yitirir. Plana kesinlikle uyulması gerektiği düşüncesi, plan dışına çıkılmasını önlemek için fazla mesai yapılmasını zorunlu kılar (Acar, 2009).

Programcı ekip ve müşteri arasına görünmez bir duvar örülür. Projenin ilerleyen aşamalarında müşteri tarafından istenilen değişiklikler olumlu karşılanmaz, çünkü bu proje başlangıcında yapılan anlaşmalara ters düşmektedir. Başlangıçta ne yapılmasına karar verildiyse, programcılar rahatsız edilmeden var olan gereksinimlerin gerçekleştirimine odaklanabilmelidirler. Doğal olarak böyle bir sistem gölgesinde oluşan programın, müşteri gereksinimlerini ne ölçüde karşılayabileceğini düşünebilirsiniz (Acar, 2009).

Geleneksel modelde gereksinimlerin tam ve açık anlaşılması çok önemlidir. Bu yüzden gereksinimlerin tam olarak anlaşılamadığı projelerde (yapay zeka ve yenilik içeren projeler gibi) uygulanması zordur.

Geleneksel biçimde yapılan projelerde gerekli değişikliklerin yapılması maliyeti zamanla yükselir;

- Planlamaya dayalıdır.
- Öngörülü (Predictive)
- Yüksek görünürlük: Proje üyeleri , bütün geliştirme süreci boyunca hangi özellik ve görevlerin planlandığını söyleyebilirler.
- Düşük uyarlanabilirlik
- Ağır (Heavy-weight)
- Düşük teknoloji (Low-tech) ve statik projelerde daha uygundur.
- Her şey planlı ve sıkı denetimli olduğu için hayati önem taşıyan (life-critical) veya çok sayıda kişi içeren büyük projelere, çevik süreçlere göre daha uygundur.
- Geleneksel modelde, kullanıcı katılımı azdır. Kullanıcı sistemin çalışan bir parçasını görmediği için, son ürünün tam olarak neye benzeyeceğini bilemez (Larman, 2003).

2.12.2 Çevik Süreçler

Proje ekibinin proje başlangıcından ve sürekli olarak çalışır durumda olan programlar oluşturarak, müşteriye sunması ve görüşlerini alması gerekmektedir. Bundan dolayı müşteri inceleyebileceği ve çalışır durumda olan bir program aracılığıyla gereksinimlerinin ne oranda gerçekleştirimle örtüştüğünü denetleyebilir. Gereksinimlerin değişmesi ya da müşterinin istekleri dışında bir yazılım yapılması durumunda müşteri yazılım sürecine düşüncelerini söyleyebilir ve gerekli değişiklikleri isteyebilir. Bu durum proje sonunda müşteri gereksinimleri ile yüksek oranda örtüşen bir programın oluşmasını sağlar (Acar, 2009).

Müşteri ile geliştirme sırasında haftalık ve aylık toplantılar yoluyla etkileşimde bulunulduğu için, gereksinimlerin değişmesi ve artması maliyeti geleneksel süreçlerde olduğu gibi artırmaz. Artışlı ve yinelemeli süreçler olduğundan değişikliklere kolaylıkla uyum sağlayan bir yapıdır.

Çevik projelerde değişmeye bir çalışma hızının oluşturulması büyük önem taşımaktadır. Programcılar arasında görev eşit bir biçimde paylaştırılır. Fazla mesai yapılması hoş karşılanmaz. Bu ayrıca çalışanların özendirmesini olumlu etkiler. Proje ilerledikçe iş azalır. Sonradan programcıları beklenmedik durumlara karşılaşmaz, çünkü ortada iyi sınanmış ve çalışır durumda olan bir program vardır (Acar, 2009).

Projelerde ekip çalışması ve bireyler arasında iletişim desteklenir. Ekip arasında alt üst ilişkisi yoktur. Bireyler bildiklerini ekip arkadaşlarına aktarırlar ve kısa sürede herkesin bilgi düzeyi ve proje ile ilgili bilgisi birbirine eşitlenir.

Müşteri gereksiniminin başlangıçta anlaşılması için, geleneksel süreçlerde olduğu gibi uzun bir süre ayrılmaz. Anlaşılan gereksinimlere uygun olarak sarmal model geliştirilir. Kullanıcıya sunulur. Kullanıcının anlaşılmayan gereksinimleri anlaşılır ve değişen gereksinimlerine uygun olarak geleneksel süreçteki adımlar yinelenir.

Her aşamada müşterinin ve geliştiricilerin görüşleri alındığından geçerliliğini yitirmiş plan ya da başta bir gereksinim olarak tanımlanmasına karşın gereksinim olmaktan çıkmış durumlar yeniden değerlendirilir ve plan yinelenir.

Özellikle programcılar, müşteri tarafından dile getirilen gereksinimlerin yapılabilirliğini araştırırken, her gereksinim için gerçekleştirme zamanını değerlendirir. Bu doğrultuda müşteri, kendi tanımlamış olduğu gereksinimlere bir öncelik sırası vererek, hangi gereksinimlerin öncelikli olarak gerçekleştirilmesi gerektiğini belirler. Programcılar bu konularda müşteriye yardımcı olarak, gereksinimlerin daha somut duruma getirilmelerini sağlarlar. Bunlar genellikle birkaç tümceden oluşan kullanıcı öyküleri (user story) dönüştürülür. Sürekli müşteri ve programcılar arasındaki karşılıklı konuşmalar ile yanlış anlaşılmaları ortadan kaldırır. Bu yüzden müşterinin her gün programcıların erişebileceği bir uzaklıkta olması gerekmektedir.

- Değişen koşullarda kolay uyum sağlar. Uyarlanabilirdir.
- Düşük görünürlük
- Hafif (Light-weight)
- İletişim ve işbirliği gerçekleme, doğrulama çalışanlarının yeterli olmaması nedeniyle, çevik süreçler genel olarak hayati önem taşıyan (life-critical) sistemlerin geliştirilmesi için daha az uygundur.
- Kullanıcı katılımı yüksektir.

  Kullanıcıyla haftalık ve aylık toplantılar düzenlenir. Kullanıcı da geliştirme sürecinin bir parçasıdır. Böylece kullanıcı memnuniyeti sağlanır (Larman, 2003).

2.12.3 Geleneksel ve Çevik Süreçler için Uygun Olan Özellikler

Çevik Süreçler için Uygun Olan Özellikler:	Geleneksel Süreçler için Uygun Olan Özellikler:
•Düşük kritiklik derecesi •Deneyimli geliştiriciler •Değişken gereksinimler •Az sayıda kişi •Değişmeyen çalışma hızı •Müşteri veya yönetici, tüm gereksinimleri başlangıçta belirlemez. •Müşteri ile proje ekibi fiziksel olarak beraber çalışır ve müşterinin ani değişkliklerini ve yeni gereksinimlerini karşılamak.	•Yüksek kritiklik derecesi •Deneyimsiz geliştiriciler •Değişmeyen gereksinimler •Çok sayıda kişi •Değişken çalışma hızı •Tüm gereksinimler, proje başlangıcında belirlenmelidir. •Müşteri ile beraber çalışılmaz.

3.UÇ PROGRAMLAMA

3.1UP'nin ortaya çıkışı

1990 lı yıllarda Chrysler firmasında yapılan bir proje bünyesinde Kent Beck, Ward Cunningham ve Ron Jeffries öncülüğünde oluşmuş, içerdiği kolay ancak etkili yöntemlerle çevik süreçler içerisinde en yaygın kullanılan yazılım geliştirme yaklaşımı durumuna gelmiştir (Acar, 2009). Başlangıçta, yazılım geliştirme sürecini etkin olarak kullanmak için kolaylaştırılmış bir yaklaşım olarak öne çıksa da, başarılı denemelerin ardından 1996 yılında olgunlaşarak yerine oturtulmuştur (Abrahamss vd., 2002). İçerisinde geçen "Uç" kelimesinin anlamı, izlediği ilkeleri uç noktalarda uygulamasından gelmektedir (Sommerville, 2006). Kolaylık, haberleşme, geribildirim ve atılganlık temelleri üzerine kurulmuş bir yazılım yöntemidir.

3.2 Süreç İşleyişi

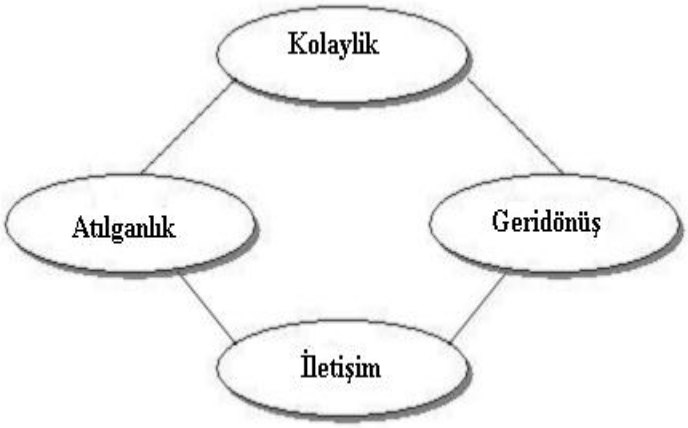
UP projelerinde projenin ilerlemesi genel olarak şu biçimdedir:

- Müşteri, gereksinimleri doğrultusunda kullanıcı öyküleri oluşturur. Bunlara öncelik sırası atar. Programcılar her kullanıcı öyküsü için gerçekleştirim zamanını değerlendirir.
- Müşteri, programcılarla beraber yineleme süreci (1-2 hafta) ve sürüm (1-2 ay) planını hazırlar.
- Müşteri ilk yineleme sürecini (1 hafta) için gerekli kullanıcı öykülerini seçer.
- Programcılar yineleme süreci için seçilen kullanıcı öykülerini uygular.
- Yineleme süreci sonunda programcılar müşteriye çalışır bir sistem sunarlar. Müşteri sistemi değerlendirerek programcılara geribildirim sağlar.
- Edinilen deneyimler ışığında bir sonraki yineleme süreci planlanır. Eğer müşteri yeni gereksinimlerin gerçekleştirilmesini isterse, bunlar için bir daha kullanıcı öyküleri oluşturulur ve değerlemeler yapılır. Eğer yeni kullanıcı öyküleri yoksa var olan kullanıcı öykü listesinden en yüksek öncelik sırası olanlar seçilir ve bir sonraki yineleme sürecinde gerçekleştirilir.

- İlk sürüm sonunda oluşan yazılım sistemi müşteri tarafından kullanıma alınır. Başka sürümler planlandıysa bir sonraki yineleme sürecinde devam edilir.

3.3 UP'nin temel taşları

UP; İletişim, Kolaylık, Geri dönüş ve Atılganlıktan oluşan dört temel üzerine oturmuştur (Şekil 3.1). UP ile verimli bir çevik süreç oluşturabilmek için bu değerlerin hepsinin kabul görmesi ve uygulanması gerekmektedir. Buna ek olarak günümüzde saygı da önem kazanmıştır.



Şekil 3,1 UP'nin temel değerleri

•<u>İletişim</u>**:** Tüm proje çalışanlarının sürekli olarak aralarında iletişim kurmaları gerekmektedir. Bireyler arası yüz yüze görüşmeler büyük önem taşır. Sadece bu sayede sağlıklı bilgi aktarımı gerçekleşebilir. Böylece yanlış anlaşılmalar ve bilinmeyenler ortadan kaldırılır. Eğer ekip içinde iletişim güçlü ise, belge oluşturma ve kullanma gereksiz duruma gelebilir. Bu da zaman yitirmesini önler. Projenin başarısı için belgeleme öncelikli rol oynamamaktadır. Programcılar arasındaki iletişim eş programlama, görev değerleme, günlük toplantılar ile sağlanır.

•<u>Kolaylık:</u> UP kolay yöntemler aracılığıyla sonuca ulaşmak ister, Çünkü sadece bu biçimde hızlı ve düşük maliyetli projeler gerçekleştirilebilir. Temel amaç "çalışan en kolay şeyi seç" tir. Bunun için en kolay tasarım,

teknik, algoritma, teknoloji seçilir. Kolay çözümlerle oluşturulan programın bakımı ve geliştirilmesi de kolaydır. Kolay çözümler kolay anlatılır ve eklenebilir. Bu da zaman kazanılması anlamına gelmektedir.

- Geri dönüş: UP temellerindeki en önemli kavramdır. UP projelerinde nitelik denetimi geribildirim (feedback) üzerinden sağlanır. Programcılar yazdıkları sınamalardan geribildirim alarak niteliği sağlarlar. Kısa süreli yineleme süreçleri ile sık sürüm oluşturarak, sürekli iyileştirme yapmak bu aşamada sağlanır. Kısa zamanlarla yeni sürüm oluşturularak müşteri ve kullanıcılardan geribildirim aracılığıyla programın gereksinimleri karşılayıp karşılamadığı denetler. UP'nin uygulanabilmesi için değişik katmanlarda geribildirim düzeneklerinin oluşturulması gerekmektedir.

- Atılganlık: Kolay çözümler, geribildirim ve iletişim için atılganlık gereklidir. Bu değerler, bireyler arası etkileşimi artıracağı için, bireylerin kendi iç dünyalarını terk edip, ekibin bir parçası olmalarını kolaylaştırırlar. Bu kişisel gelişmeyi sağlar ve temelinde kişisel atılganlık yatar. Tasarımı düzeltme, kötü tasarlanmış kodun atılması, net planlama yapma bu sayede gerçekleşir.

3.4 UP İlkeleri

UP değerlerinden yola çıkarak on beş UP ilkesi oluşturulmuştur (Acar, 2009). Bunlar:

- Hızlı Geri dönüş (Rapid Feedback) : Sık ve hızlı geribildirim edinmek, projenin gidişatını olumlu etkiler. Geribildirim sayesinde yanlış anlaşılmalar ve yanlışlar ortadan kaldırılır. Derleyici geribildirimi (saniyeler), eşli programlama geri bildirimi (saniyeler), birim sınaması geri bildirimi (dakikalar), müşteri geribildirimi (günlük) gibi biçimler de olabilir.

- Kolaylığı Seçmek (Assume Simplicity): UP, programcılardan o sıradaki gereksinimi karşılamak için kolay çözümü bekler. Kolay çözümler kolay gerçekleştirilir ve kısa zamanda oluşturulur. Bu sayede geribildirim de hızlı bir biçimde gerçekleşmiş olur. Kolay çözümlerin kavranması ve anlatılması daha kolaydır. Programcı gelecekte oluşabilecek eklemeleri ve değişiklikleri düşünmemeli, sadece ve sadece kendisinden o an için bekleneni en kolay

durumu ile gerçekleştirmelidir. Bu süre içinde niteliği de ön planda tutarak, daha sonra karmaşıklığın eklenmesine olanak sağlamalıdır.

• Artışlı Değişiklik (Incremental Change): Yazılım sistemleri zaman içinde karmaşık bir yapıya dönüşebilir. Yapılan en ufak bir değişiklik, sistemin bir bölümü üzerinde yanlışlık oluşmasına neden olabilir. Oluşabilecek bu yanlışlar denetim altında tutabilmek için değişikliklerin ufak çapta olması gerekmektedir. Büyük değişiklikler beraberinde büyük sorunları getirebilir. Bu nedenden dolayı değişikliklerin ufak çapta ve sıklıkla yapılması gerekmektedir.

• Değişimi İstemek (Embracing Change) : İlerleyebilmek için yeniliklere açık olmak gerekir. Yeniliklere açık olmak da atılganlık gerektirir. Bilinmeyenle uğraşmak, rahatsız edici olabilir ancak başarıyı elde edebilmek için değişimi istemek gerekir.

•Nitelikli İş **(**Quality Work**):** Her programcı yanlışsız program yazmak ister. Çalışma ortamının da etkisiyle yüksek nitelik de yazılım yapmak hem programcının özgüvenini ve çalışma isteğini artırır, hem de müşterinin isteklerini karşılayabilecek ürünlerin ortaya konmasını sağlar.

•Öğrenmeyi Öğret (Teach Learning) : UP programcı ekiplerinde deneyimli olma ayrıcalığı yoktur. Deneyimli programcılar, bilgilerini daha az deneyimli programcılarla paylaşarak, hem bilginin çoğalmasını sağlarlar, hem de ekip arkadaşları ile teknik olarak aynı düzeye gelirler. Programcılara komutlar vererek iş yaptırmak yerine, kendiliğinden bazı şeyleri öğrenerek, görevlerini yerine getirmeleri sağlanmalıdır.

•Az Başlangıç Yatırımı (Small Initial Investment) : UP'de başlangıç giderleri ne kadar düşük tutulabilirse, projenin kaldırma durumunda kayıplar o oranda az olacaktır. Örneğin proje başlangıcında son model araçlar satın alınarak, tüm ekibe nasıl kullanacaklarına ilişkin seminer verilebilir. Bu çok pahalı ve bir o kadar da gereksizdir. Projeye açık kaynaklı ve ücretsiz araçlarla başlanırsa programcı ekip ne bir hafta tanımadıkları bir ürün için harcamış olur, ne de büyük harcamalar yapılarak şu an için gerek olmayan bir altyapı bileşeni satın alınmış olur. Gerekli araçlar zamanı geldiğinde edinilmelidir.

•Kazanmak İçin Oyna (Play to win) : UP ekiplerinin kafalarında istenilen sonuç vardır; programı tamamlamak ve müşteriye vermek. UP, programcı ekibe tünelin sonundaki ışığı görmek için gerekli tüm olanakları sunar.

•Somut Denemeler (Concrete EUPeriments) : Alınan kararların sonuçlarını doğrulamak için denemeler yapılır. Bunun için bir denetleme düzeneği gerekmektedir. Somut denemeler yazılım sistemlerinde sınamak, oluşturulan mimari ve tasarımı denetlemede yarar sağlar.

•Açık ve Dürüst İletişim (Open, honest Communication) : Çalışanlar arasında açık ve dürüst türde bir iletişim ile proje başarısı olanaklıdır. Birçok projede bu böyle değildir. Çoğu zaman bireylerin korkuları, deneyimsiz olmaları veya kendilerini çok beğenmeleri ve diğerlerini kendilerinden alt aşamada görmeleri, açık ve dürüst bir iletişim ortamının oluşmasını engeller.

•Ekibin İçgüdülerini Kullan, Onlara Karşı Koyma (Work with people's instincs, not against them): Bireysel içgüdünün yanı sıra, ekiplerin da içgüdüsü vardır. Eğer ekip, bir şeylerin doğru gitmediği duygusunu duyarsa ve bunu dile getiriyorsa, o zaman yolunda gitmeyen bazı şeyler vardır demektir. Ekibin içgüdüsüne kulak verilmelidir. Bunun göz ardı edilmesi, proje için olumsuz sonuçlar doğurabilir.

•Sorumluluk Üstlenmek (Accepted Responsibility): Sorumluluk birilerine verilmemeli, bireyler kendileri sorumluluk üstlenmeliler. Bireyler / ekipler kendi sorumluluklarını kendileri seçerlerse, hem yaptıkları işte kendilerini iyi hissederler, hem de yüksek özenme ile üstlendikleri işi başarıyla tamamlarlar. Eğer bir bireye / ekibe yapılması zor bir projenin sorumluluğu yüklenirse, bu birey / ekip için güdünün düşmesini ve kaybetme korkusunun pekişmesini hızlandırır.

•Sürecin Ortam Koşullarına Uyarlanması (Local Adaptations): Her ekip, UP'yi Kent Beck'in anlattığı biçimindeki uygulaması olanaksızdır. Amaç kısa bir zamanda projeyi başarılı bir sonuca ulaştırmaktır. Eğer proje UP'de yapılacak değişikliklere başarıya ulaşacaksa, o zaman süreç üzerinde bu değişiklikler yapılmalı ve uygulanmalıdır.

• 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.

• Dürüst Ölçüm (Honest Measurement): Proje gidişatını denetlemek için değişik türde ölçümlerin yapılması gerekmektedir. Ölçülmeyen bir şey değerlendirilemez. Yapılan ölçümler doğru ve dürüstçe yapıldığı biçimde projenin gidişatını olumlu yönde etkilenir. Örneğin hazırlanan birim sınamaları ile sınıfların işlevleri denetlenir.

3.5 UP Uygulamaları

• Programcıya Yakın Müşteri (On-site Customer) : UP projeleri müşteri gereksinimlerine odaklı ilerler. Bu yüzden müşteri ve sistem kullanıcılarının projeye içinde olmaları gerekmektedir. Müşteri gereksinimlerini ekibe bildirir. Yanlış anlaşılmaları ve yanlışları gidermek için programcıların müşteri ve sistem kullanıcıları ile iletişim içinde olması gerekmektedir. Bu nedenle müşteri veya sistem kullanıcılarının programcıların erişebileceği bir uzaklıkta olmaları gerekir. UP projelerinde genellikle müşteri ve programcılar aynı odada beraber çalışırlar. Müşteri ekibin sorularını zaman kaybı olmadan yanıtlar ve projenin ilerlemesine katkıda bulunur.

• Kısa Toplantılar (Standup-Meeting): Proje çalışanları her gün 15 dakikayı aşmayan ve ayakta yapılan toplantılarda bir araya gelirler. Bu toplantının amacı, projenin gidişatı ile bilgi alışverişinde bulunmaktır.

• Planlama Oyunu (Planning Game): Müşteri ve programcı arasındaki yakın etkileşimin ürünüdür. UP projeleri yinelemeli ve artışlı yol alır. Bir sonraki yinelemede yapılması gereken işler, planlama oyununda görüşülür. Müşteri ve kullanıcılar daha önce kullanıcı öyküsünde (user story) dönüştürdükleri isteklerine öncelik sırası verirler. Programcılar her kullanıcı öyküsü için gerekli zamanı belirler. Kullanıcı öykülerini öncelik sırası bu değerleme (valuation) olarak değişebilir. Planlama oyunlarında sürüm ve yineleme planları oluşur.

• Kısa Aralıklarla Yeni Sürümler (Short Releases): UP projelerinde kısa sürede sürümler oluşturularak müşteri ve kullanıcının beğenisine sunulur. Bu sayede hem müşteriler çalışır durumda olan programdan yararlanabilir hem de yeni sürümü inceleyerek, gereksinimleri ile örtüşüp, örtüşmediğini denetler – geribildirim yaparlar. Eğer yeni sürüm müşteri isteklerini karşılayacak durumda değilse, gereksinimler değişikliğe uğrayabilir. Bu değişiklikler bir sonraki yinelemede göz önünde bulundurularak, müşteri istekleri ile yüksek düzeyde örtüşen bir programın oluşturulması sağlanır.

• Geriye Bakış (Retrospective): Proje çalışanları düzenli aralıklarla geriye bakarak, ortaya çıkan sorunları gözden geçirirler. Buradaki amaç gelecekte bu sorunların tekrarını önlemektir. Geriye bakış bir ila altı aylık zaman birimleri için tüm proje çalışanları yâda seçilen bireyler tarafından yapılır. Geriye bakış toplantıları yarım gün ila üç gün arasında sürebilir.

• Benzetme (Metaphor): UP projelerinde, oluşan program için, programın nasıl bir işlevi olacağını ekibin gözünde canlandırmalarını sağlayacak benzetmeler, öğe ya da görüntüler kullanılır. Bunlar proje çalışanlarının ortak bir payda da buluşarak, ne yapılması gerektiği ile ilgili bir düşünceleri olmalarını kolaylaştırır. Örneğin bir alışveriş sistemi yazılımı yapılıyorsa burada benzetmeler olarak alışveriş sepeti kullanılabilir. Alışveriş sepetini duyan her programcının beyninde, bir alışveriş sisteminin programlanması gerektiği düşünce doğar.

• Ortak Sorumluluk (Collective Ownership): UP projelerinde programcılar ortak sorumluluk taşırlar. Bu her kod parçasının herhangi bir programcı tarafından gerekli durumlarla değiştirilebileceği anlamına gelir. Böylece yapılması gereken işler aksamaz, çünkü belli kod bölümlerinden belli programcılar sorumlu değildir. Tersine her programcı programın her bölümü üzerinde çalışabilir. Bir programcının işe gelmemesi durumunda, başka bir programcı kolaylıkla onun görevlerini üstlenebilir.

• Sürekli Bütünleştirme (Continuous Integration): Sistem değişiklikleri ve yeni bileşenler hemen sisteme eklenerek sınanır. Sürekli bütünleşme sayesinde yapılan tüm değişiklikler, her programcının sistem üzerinde yapılan değişiklikleri görmesini sağlar. Ayrıca sistem bütünleşmesi için

gerekli zaman azaltılır, çünkü oluşabilecek yanlışlar erken belirlenerek, ortadan kaldırılır.

• Kodlama Standartları (Coding Standards): Programcılar tarafından aynı nitelik de kod yazılımı yapılabilmesi için, kod yazarken kullanılacak kuralların oluşturulması gerekmektedir. Kodun nasıl biçimleneceği, sınıfların, yöntem adlarının ve değişkenlerin nasıl adlandırılabileceğini kod standartlarında yer alır.

• Kalıcı Çalışma Hızı (Sustainable Pace): UP projelerinde programcılar haftalık belirli çalışma süreleri aşmazlar. Gereğinden fazla çalıştırılan ve yorulan bir programcıdan verimli iş yapması beklenemez. Programcıların güdünün ve çalışma güçlerinin yüksek olması için günde sekiz saatten fazla çalışmalarına izin verilmemelidir. Kalıcı Çalışma Hızı, haftada 40 saatten fazla mesai yapmamak olarak tanımlanabilir.

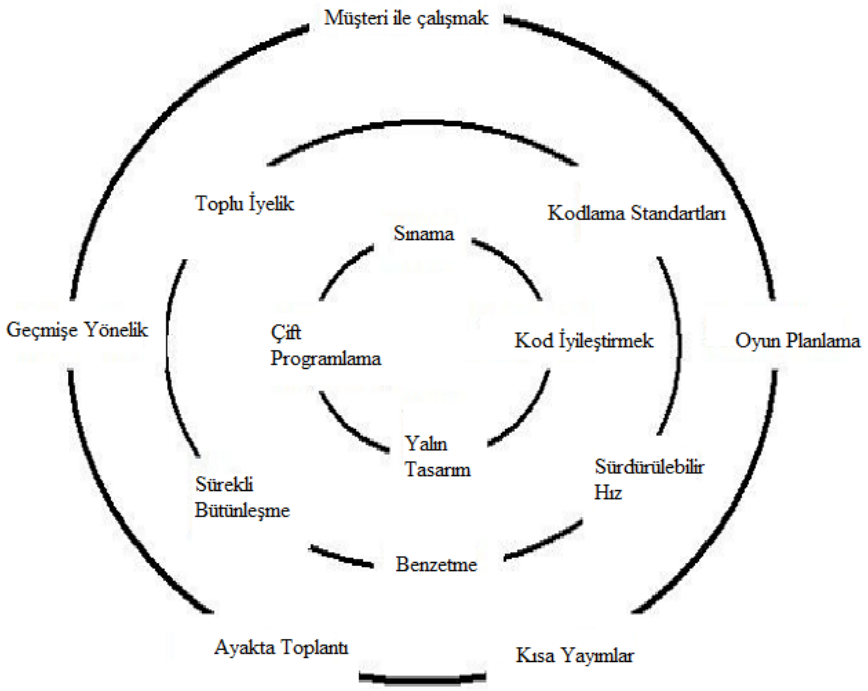
• Sınama (Testing): UP, sınama odaklı bir süreçtir. Oluşturulan programların nitelik denetlemesinden geçmesi gerekmektedir. Bu, yazılım sırasında oluşturulan denetlemelerde yapılır. Programcılar, bileşenler için birim sınamaları hazırlar. Bu sınamalarla bileşenlerin işlevleri denetlenir. Müşteri gereksinimlerini sınamak için onay sınamaları hazırlanır. Bileşenlerin bütünleşmesini sınamak için bütünleşme sınamaları hazırlanır.

• Sade Tasarım (Simple Design): Programcılar üstlendikleri görevleri en kolay durum ile yaparlar. Bu sayede, programın kolay bir yapıda kalmasını ve ilerde değiştirilebilir ve genişletilebilir olması sağlanır. Sade tasarımlar daha kolay ve daha hızlı gerçekleştirilir. Kolay bir gerçekleştirimi anlamak ve anlatmak daha kolaydır.

• Kod İyileştirme (Refactoring): Sistemin davranış ve özelliklerinin değiştirmeden yeniden yapılandırılmasıdır. Hazırlanan birim sınamaları ile kısa sürede yapılan küçük değişikliklerin etkilerini denetler. Tüm sınamalardan geçen yazılımda iyileştirme işlemleri sürer. Gereksiz birimlerin kaldırılması, birimler arası iletişimin artırılması, kolaylaştırma, esneklikle sağlanabilir.

• Pair Programming (Eşli programlama): İki programcının aynı bilgisayarda çalışmasıdır. Eşlerden biri kodlama yaparken, diğeri kod gözden geçirme

yapar. Böylece programcıların kısa bir zaman içinde aynı düzeye gelmesi sağlanır. Ayrıca niteliğinin yükselmesi sağlanır. (Bkz. Şekil 3.2)



Şekil 3.2 UP'nin teknikleri:www.XProgramming.com (Acar, 2009)

Uygulamalar, birbirlerini destekler durumdadır. (Abrahamass vd. 2002)(Sommerville, 2006)(Highsmith, 2000)(Newkrik, 2002)

3.6 UP 'de Roller

Bir çevik projede ekip çalışanlarının sorumluluk alanlarını tanımlamak için roller belirlenir. Her rol beraberinde bazı sorumluluklar ve tanımlanmış yetkiler getirir. Kişilere değişik roller verilebilir ve daha sonra rol değişikliği yapılabilir. Proje gereksinimleri doğrultusunda yeni roller ortaya çıkabilir (Abrahamss vd., 2002)(Acar, 2009).

- Müşteri: Yazılımı isteyen ve gereksinimlerine yanıt verebilecek bir yazılım sistemi için yatırım yapan kişidir. Projenin var olma nedenidir. Yapılması gerekenleri kullanıcı öyküleri (user story) oluşturarak anlatır. Programcılar müşteriye bu süreçte yardımcı olurlar. Her kullanıcı öyküsü yazılım sisteminin bir özelliğini tanımlar. Oluşturulan kullanıcı öykülerinin sınanabilir yapıda olması gerekmektedir. Hangi kullanıcı öykülerinin

gerçekleştirileceğine müşteri belirler. Bu konuda müşteriye herhangi bir sınırlama getirilmez. Gerçekleştirilen kullanıcı öykülerini denetlemek amacıyla müşteri, onay sınamaları tanımlar. Bu sınamalar programcı ya da sınamacı tarafından sınanır. Bu sınamalar, kullanıcı öykülerinin doğru gerçekleştirilip gerçekleştirilmediğini denetleyici bir düzenektir.

• Programcı: Sistem çözümleme, tasarım, gerçekleştirim ve sınama aşamalarından sorumludur. Müşteri tarafından hazırlanan kullanıcı öykülerinin gerçekleştirim süresi programcılar tarafından belirlenir. Yani programcılar, UP projelerinde proje planlama sürecinin içinde olmaktadırlar. Geleneksel projelerde bu değerlemeleri teknik bilgisi olmayan yöneticiler yapmak zorundadır. Bu yüzden bu değerlemeler genelde gerçekleri yansıtmaz. Her programcı sınama güdümlü (TDD - Test Driven Development) ve bir ekip arkadaşıyla beraber çalışır. Eşli Programlama olarak bilinen, iki programcının birlikte yazılım yapması, kısa zamanda kod ile ilişkin bilginin tüm programcılar tarafından paylaşılmasını kolaylaştırır. Ayrıca Eşli Programlama programcılar arasında iletişimi ve ekip içinde çalışabilme özelliğini artırır. Sınama güdümlü çalışmak bakımı ve geliştirilmesi kolay kodun oluşmasını sağlar. UP projelerinde programcılar herhangi bir satır kod yazmadan önce gerekli sınama sınıflarını oluşturarak gerçekleştirmeye başlarlar. Programcılar oluşturdukları sınamalar yardımıyla her gün bir veya birden fazla biçimde birim bütünleşmesi gerçekleştirirler. Sürekli bütünleşme programcılar tarafından oluşturulan birimleri ekleyen bir süreçtir. Her programcı bu süreçten geribildirim sağlayarak, kendi yaptıklarının ne düzeyde sisteme ile bütünleştiğini ölçebilir.

•Proje Yöneticisi (Big Boss):Müşteri ve programcıları bir araya getiren, beraber çalışabilecekleri ortamların oluşmasını sağlayan kişilerdir. UP proje yönetici tek başına proje planlamasından sorumlu değildir. Programcılara görev atamaz, onların kendi başlarına seçim yaparak, sorumluluk almalarını kolaylaştırır. Toplantı ve diğer buluşmaları düzenler, ekibin karşılaştığı sorunları ortadan kaldırmak için gerekenleri yapar.

• Koç: Çevik süreci tanıyan ve nasıl uygulanması gerektiğini bilen uzmandır. Koçun görevi proje başlangıcında çevik ekibi oluşturmak ya da bir araya getirmek ve onlara belirli bir süre rehberlik yapmaktır.

• Sınamacı: Müşteri tarafından onaylanan sınamaların oluşturulmasına yardım eden ve bu sınamaları gerçekleştiren programcıdır.

• Danışman: Teknik bilgiye gerek duyulduğunda başvurulan dışsal üyedir.

3.7 UP Proje Aşamalar

UP projesinin her aşamasında, kendine özgü etkinlikler bulunur. Görüntüde bir UP projesinde olması gereken aşamalar yer almaktadır (Abrahamss, 2002).

UP projesi şu aşamalardan oluşur:

• Keşif (EXPloration Phase): Projenin başlangıcında yapılan çalışmaları içerir. Müşteri kullanıcı öykülerini (user story) oluşturur. Programcılar teknik altyapı için gerekli deney (spike) ve araştırmayı yaparlar.

• Planlama (Planning Phase): Bu aşamada. Müşteri kullanıcı öykülerine öncelik sırası vererek, yinemelerde hangi kullanıcı öykülerinin öncelikli olarak gerçekleştirilmesi gerektiğini belirler. Kullanıcı öykülerini gerçekleştirim süresi programcılar tarafından değerlenir. Bu aşama genel olarak birkaç gün sürer.

• Yinelemeden Sürüme ve Ürün Oluşturma Aşaması (Iterations to Release and Productionizing Phase): Kullanıcı öykülerinin gerçekleştirimi aşamasıdır. Her yinelemenin sonrası müşteri tarafından onay sınamaları belirlenir. Bu sınamalar programcılar ya da sınamacılar tarafından gerçekleştirilir. Her yineleme sonunda müşteriye, çalışır bir yazılım sistemi sunulur. Bu biçimde müşterinin sisteme ilişkin görüşleri alınır (geribildirim). Yineleme son bulduktan sonra, bir sonraki yinelemede gözden geçirilmek ve giderilmek üzere yineleme süreç planı yapılır.

• Bakım Aşaması (Maintenance Phase): Programın geliştirilmesiyle birlikte bakımının yapıldığı aşamadır. Bu aşamada küçük çapta eklemeler ve sistem yanlışlarının giderilmesi için işlemler yapılır. Ayrıca, kullanıcılar için eğitim seminerleri hazırlanır.

4. DAĞITIK UÇ PROGRAMLAMA (DUP)

4.1 Giriş

Dağıtık Uç Programlama ilk kez Uç Programlamanın dağıtık yazılım ekiplerinde kullanılması için Kircher, Jain, Corsaro ve Levine tarafından 2001 yılında önerilmiştir (Kircher vd. 2001) ve bu konuyu seçme nedenim ise, günümüzde dünyanın değişik yerlerinde olan kişilerin biri biri ile uluslararası ağ (internet) yolu ile kolaylıkla iletişimlerinin sağlanması, bu yöntemle sınırlamaları ve maliyeti azaltırken aynı zamanda devingenlik ve müşteri ile ilgili olmayı arttırmasıdır.

Kircher, Jain, Corsaro ve Levine'nin düşüncelerine göre, Uç Programlamanın 12 uygulamasından sadece 4'ü beraber çalışan ekipler için gereklidir. Bunlar: Çift Programlama, Sürekli Bütünleştirme, Oyun Planlamak ve Yerinde Müşteri ile Çalışmak.

DUP için en önemli olan geliştirme ekibi, müşteri ve yönetim arasında etkin ve güçlü bir iletişimin olması. Bu iletişim düzeyini sağlamak için geliştirme ekibinin fiziksel olarak beraber çalışmaları gerekir aynı zamanda müşterinin de sürekli geliştirme ortamında bulunması gerekmektedir. Bunun dışında güçlü iletişimin sağlanması bazı nedenlerden dolayı olanaksızdır. Örnek olarak geliştirme ekibinin fiziksel olarak biri birinden uzak olması, bireysel sınırlamalar, ekip çalışan sayısının fazla olması ve sanal geliştirme ortamına alışık olmamaları. İşte bu sorunları çözmek için ve fiziksel yakınlığına gerek duymadan bir ekip çalışanlarının beraber çalışması için Kircher, Jain, Corsaro ve Levine tarafından Dağıtık Uç Programlama (Distributed eXtreme Programming) düşüncesi ortaya atıldı (Kircher vd., 2001).

Oyun Planlamak durumunda, öykü kartları (story cards) müşteri ile geliştirici arasında proje ile ilgili bilgi alışverişinin ve iletişimin sağlanmasının temel aracıdır. Müşteri ve geliştiriciyi projenin son durumu hakkında bilgilendirmek için kullanılır. Bu kartların yaratılması, üstüne değişiklik yapılması, yeniden tahmin edilmesi ve seçilmesi iletişime dayanmaktadır. Bu nedenle, DUP içinde öykü kartları, uzaktan erişilebilir olması ekip üyelerinin arala rında daha etkili iletişim kurmasını sağlamaktadır.

Çift Programlama durumunda, iki geliştirici aynı bilgisayarda oturup aynı kod parçası üzerinde çalışmaktadırlar. Bu iki geliştiriciden biri kod yazarken öteki yazılan kodu gözden geçirmektedir, tasarım ile ilgili düşünüce ve önerilerini söyler ve yorumda bulunur ve roller sık sık değişir. Örnek olarak 15 dakikada bir roller değişir ve bu çiftler görevler değiştikçe değişir. Çift Programlamada İletişim ve uygulama paylaşımı çift için çok önemlidir. Dağıtık Çift Programlamada (Distributed Pair Programming (DPP)) (Stotts vd. 2002), iki geliştirici değişik ekranlarda kodu izler, birbirinin sesini duyar ve sadece iki geliştiriciden biri kod üzerinde değişiklik yapabilir.

Sürekli Bütünleştirme durumunda, görev bitmeden sistemle bütünleşmesi gerekir. Bütünleştirme işlemi günde bir kaç kez ve toplu bir biçimde yapılır. Bu nedenle, geliştiriciler değişiklikler ve bütünleştirmelerden haberdar olmalıdır.

Müşteri ile yerinde çalışmak durumunda, müşteri geliştirme ekibinden biri olarak geliştiricilerle beraber çalışır ve sorularını yanıtlayıp onaylama sınamalarını (acceptance tests) yazmak için. Bu yüzden bazı iletişim araçları (görüntülü toplantı veya masaüstü telefon uygulamaları gibi) müşteri ile geliştirme ekibi arasında iletişim sağlamak için kesinlikle olmalıdır.

Dağıtık Uç Programlamayı proje geliştirme ekibinin üyelerinin fiziksel yakınlığına gerek duymadan gerçekleştirilen Uç Programlama olarak tanımlıyoruz. DUP'de UP'nin ilkelerini dağıtık ve devingen ekip ortamına uygulamaktadır. DUP'de ekip üyeleri isteğe bağlı olarak ve yüksek devingenlikle biri birinden istediği uzaklıkta olabilmektedir. Bu yöntemin bazı düşünceleri "Yazılım Mühendisliği XP 2000'de Uç Programlama ve Esnek Süreçler" (eXtreme Programming and Flexible Processes in Software Engineering XP2000) konferansında anlatıldı (Kircher ve Levine, 2001), (Schuemmer ve Schuemmer, 2001).

DUP bazı değişiklikler dışında UP'nin çoğu ilkesini içerir. İkisinin arasındaki en keskin fark ise proje ekiplerinin beraber çalışıp çalışmamasıdır. DUP'da proje ekipleri biri birinden uzak yerlerde olur, UP ise proje ekibi aynı yerde çalışmak zorundadır (Kircher vd. , 2001).

Alttaki çizelge DUP için gerekli olan veya olmayan konuları özetlemektedir:

Çizelge 4.1 UP uygulamasının aşamaları ve ekip çalışması

UP Uygulaması	**Ekibin beraber çalışması gerekli mi?**
Oyun Planlamak (Planning Game)	Evet, proje ile ilgilenen kişilerin (Yazılım Geliştiriciler ve Müşteriler) beraber çalışmalı.
Çift Programlama (Pair Programming)	
Sürekli Bütünleştirme (Continuous Integration)	
Müşteri ile yerinde çalışmak (On-Site Customer)	
Küçük Sürümler (Small Releases)	Hayır, işlemler proje ekibi beraberken veya dağıtık bir biçimde çalışırken gerçekleşebilir.
Benzetme (Metaphor)	
Kolay Tasarım (Simple Design)	
Sınama (Testing)	
Yeniden Düzenleme (Refactoring)	
Ortak İyelik (Collective Ownership)	
Haftalık 40 çalışma saati (40-Hour Week)	
Kodlama Standartları (Coding Standards)	

Bu çizelgeden çıkarsıyoruz ki, etkili bir DUP geliştirmek için Oyun Planlamak, Çift Programlama, Sürekli Bütünleşme ve Yerinde Müşteri ile Çalışmak aşamalarına Dağıtık Ekip Ortam (Distributed Team Environment) ilkelerini uygulamak zorundayız (Kircher vd., 2001).

Bu işlemde ("Yani Dağıtık Ekip Ortam" - Distributed Team Environment ilkelerini uygulamada) yeniden düzenleme söz konusudur ve bu düzenleme Çift Programlama aşamasında gerçekleştirilmelidir (Cowan, 2001).

4.2 DUP Varsayımları

DUP'nin etkili olması için belli araçlar ve teknolojilerin var olması kesin bir koşuldur. Bazı UP varsayımları yanı sıra, ortak dil konuşma ve genel açıklık gibi, DUP varsayımları ise:

- Bağlantı (Connectivity): Bazı bağlantıların ekip üyeleri arasında olması gerekmektedir. Uzaklıktan dolayı bu bağlantı yapılamıyorsa uluslararası ağ bir bağlantı aracı olarak kullanılabilir. Ayrıca şirketin yerel iletişimi için de uluslararası ağ kullanılabilir.
- E-Posta: Her yerde bulunması nedeni ile E-Posta, DUP'in en önemli anahtarlarından biri durumuna geldi. Bilgi alışverişi veya DUP'in herhangi bir aşamasını programlamak için kullanılabilen uygun bir araçtır.
- Kurulum (Configuration) Yönetimi: Etkili bir programlama yöntemi geliştirmek için bazı kurulum yönetimi araçlarını kullanmak gerekmektedir, bu da ortak iyelik için önemli bir kolaylaştırıcı olarak destek vermektedir.
- Uygulama Paylaşımı: UP ilkelerini Dağıtık Ortama uygulamak için bazı uygulamalar veya masaüstü paylaşım yazılımları ekipte olmak zorundadır.
- Görüntülü Toplantı: Biri birinden uzak olan ekip üyeleri arasında etkili bir iletişim için ses ve görüntülü toplantıları (Nehrstedt ve Steinmetz, 1996) kullanılmalıdır.
- Tanışıklık (Familiarity): Anlaşılan gibi DUP sadece ekip üyeleri biri birilerini iyi tanıdığı durumunda başarılı olabilir ve bu iyi tanımanın bir uzantısı olarak, ekip üyeleri arasında düzenli ve uyumlu çalışmayı gözetleyebiliriz (Kircher vd. , 2001).

4.3 Neden DUP?

UP'nin uygulanması için proje ekibi üyeleri arasındaki fiziksel yakınlık büyük önem taşımaktadır. Yalnız bu fiziksel yakınlığın olanaksız olduğu durumlardan dolayı, ortaya DUP çıktı. Bir şirket veya bir projede altta belirlenen nedenlerden dolayı DUP'yi benimsemek zorunda:

- Durum kısıtlamaları (Constrained by situation): Bir şirket veya projede, geliştirme seçenekleri proje ekiplerinin fiziksel dağılımına göre belirlenir.

Çoğu projelerde proje ekibinin üyeleri fiziksel olarak biri birinden uzak olmasına rağmen projeler gerçekleşir.

- Bireysel kısıtlamalar (Individual Constraints): Birey kişisel nedenlerden dolayı şirketin merkezinde çalışamaya bilir ve bu nedenlerden dolayı şirket merkezi dışında çalışmayı seçebilir. Bu durumda önemli olan bu bireyin fiziksel olarak proje ekibi ile beraber olması değil, bu bireyin geliştirme etkinliklerinin bir parçası olarak kalabilmesidir (Kircher vd. , 2001).

Bir şirket veya projede koşullar zorlanmasa bile DUP seçilebilir. Bunun nedeni ise, DUP'in uygulanması UP'nin yararlarını korur ve ek olarak yeni yararlar da sağlayabilir. Bunlardan:

- Maliyet (Cost): Yazılım sektöründe bir büyüme eğilimi görmekteyiz ve bu büyüme nedeni ile projenin tamamının veya bir kısmının maliyetini karşılanması için dış kaynak sağlamaktır. Yazılım projeleri çoğu Hindistan veya Çin gibi bazı ülkelerde geliştirilmesi daha uygundur. Sonuç olarak, çeşitli projeler iki veya daha fazla ülkeye dağıtılarak geliştirilmektedir.
- Uygun Müşteri Katılımı (Convenient Customer Involvement): DUP müşterinin proje ekibi ile beraber çalışma zorunluğunu ortadan kaldırır. UP'de müşteri proje ekibi ile aynı yerde olmak zorunda. Bu olay müşteriye çekici gelemeyebilir, ayrıca müşteri proje ekibi ile çalıştığı sürede kendi şirketinde olanlardan ve iş akışından uzak kalır. DUP'de bu sorun ortadan kalkıyor çünkü müşteri proje ekibi ile aynı yerde olamazsa bile görüntülü toplantı ile proje ekibine bağlanabilir.
- Devingenlik (Mobility): Birçok kuruluşta, bazı ekip üyeleri müşteri isteklerini karşılamak için, konferanslara katılmak için veya başka nedenlerden dolayı sık sık seyahat etmektedir. DUP'de devingen ekip üyelerine proje ekibi ile bağlantılı kalmaları için iyi bir olanak sunar. Devingen ekip üyelerinin ekibin geri kalanı ile çeşitli devingen yöntemleri kullanarak bağlanabilir (küçük bir kamera, bir ISDN...) ve geliştirme etkinlikleri istenirse gün boyunca veya birkaç saatliğine katlanabilir.

Böylece hem UP'nin sunduğu yararlardan hem de DUP'nin sunacağı ek olanaklardan yararlanabiliriz (Kircher vd., 2001).

4.4 DUP'nin zorlukları

UP'de temel koşul olarak belirlenen fiziksel yakınlık aslında çok şey kolaylaştırmaktadır. DUP'de ise bu fiziksel yakınlık olamadığı için bazı zorluklar ortaya çıkmaktadır. Bunlardan:

- İletişim (Communication): İnsanlar biri birilerini anlamaları için önemli bir konu karşısındaki insanın davranışlarını, yüz ifadelerini, hareketlerini ve tepkisini görmektir. Bu davranışları, yüz ifadeleri, hareketleri ve tepkileri görerek karşı tarafın düşüncelerini anlar ve o kişi ile ilgili kararı verebilir. DP'de insanlar fiziksel olarak görüşemediklerine göre bunların belirlemesi zordur.
- Eşgüdüm (Coordination): Projede ekibinde çalışan iki kişi ayrı yerlerde çalışıyorlar ise bunların arasında eşgüdüm sağlamak bir zorluktur. Bu zorluklar arasında eşleme durumu, saat farkları için ayarlama ve dağıtık olan etkinlikleri bütünleştirmek ile eşgüdümünü sağlamak yer almaktadır. Bunun yan ısıra dağıtık ekip üyeleri arasında belge değişimi de bir sorun durumuna gelebilir.
- Altyapı (Infrastructure): İletişim ve eşgüdüm için sağlam bir altyapı gerekmektedir. Bu altyapı donanım, yazılım ve ağ bağlantısının bağ genişliğini içerir. Güçsüz bir altyapı DUP'de sorun olan fiziksel uzaklığın çözülmesini iyice zorlaştırır.
- Erişilebilme (Availability): Dağıtık ekip üyeleri ayrı ayrı zamanlarda uygun olabilirler. Bazıları başka projelerde çalışıyor olabilir ve o yüzden süreler de sınırlı olur. Bunun yanı sıra çalıştıkları yerlere göre saat farklılıkları da olabilir.
- Yönetim (Management): Ekip üyeleri sürekli yöneticiden uzak olduğu için yönetici ile çalışanları arasında çok yüksek düzeyde güven olmalı ve uzaktaki çalışanları denetlemek ve yönlendirmek için yeni stratejiler geliştirilmelidir (Kircher vd., 2001).

4.5 Zorluklar ve çözümleri

DUP ‘de birçok zorluk da bulunmaktadır. Ancak, tüm bu zorlukların ele alınması ve çözülmesi gerekmektedir.

- İletişim (Communication): Bir biri ile sıkı bağlantıları olan ve aralarındaki fiziksel uzaklığı yok edecek düzeyde aralarında iyi bir iletişimin olduğu bir ekip varsayalım. Örnek olarak, karşıdaki kişiyi oldukça tanıdığını varsayalım. Karşıdaki kişiye ilişkin birkaç görüntü o kişinin nasıl düşündüğünü ve verdiğiniz komutlara nasıl tepki verdiğini öğrenmek için yeterlidir. Ekip üyeleri bu fiziksel uzaklığı çözebilmek için buna benzer değişik yöntemlere başvurabilir. Buna örnek olarak görüntülü toplantı kullanılabilir veya biri birine e-posta atabilirler. Bunun yanılıra ekip üyelerinin bağlarını güçlendirmek, biri birilerini daha iyi tanımalarını sağlamak ve projeyi daha iyi bir biçimde yönetmek için dönemsel toplantılar da yapılabilir. İletişim için ortak bir forum oluşturmak istersek değişik etkenlerin onaylanması gerekmektedir. Bu forum donatım ve kullanım maliyeti, yolculuk maliyeti, zaman maliyeti, ağ bağlantısının bağ genişliği ve forumda görevli ile yapılması gereken özel iletişim, tüm bunları içerir.

  Uzaktan iletişim ve işbirliği belge değişimi ile büyük ölçüde başarılı olabilir. Web teknolojileri ucuz olduğu için çok kullanılmaktadır. Yeni iletişim yöntemleri (İnternet, İntranet, görüntülü toplantı, …) proje ekibi arasında yakınlık sağlamaktadır.

- Eşgüdüm (Coordination): Dağıtılan ekip üyeleri arasında yapılacak olan etkinlikler için iyi bir eşgüdüm planlaması yapmamız gerekmektedir. Olası yanlışlıkları önlemek ve iletişimi kolaylaştırmak için değişik ve birden çok iletişim yöntemi kullanılmalıdır. Örnek olarak, değişik yerlerde olan iki ekip üyesi günlük planlarını, yapılanların ve yapılması gerekenleri biri birilerine e-posta ile gönderebilirler. Daha sonra gün içinde bir proje üzerinde çalışmak için belirli ortak bir yer belirleyebilirler. Bunları yaparken unutulmaması gereken başka bir konu ise zaman farkıdır.

- Altyapı (Infrastructure): Gerekli teknik altyapının durumu ve tüm ekip üyeleri için yeterli yazılım ve donanımın bulunması çok önemlidir. Yazılım açısından, diğer araçlar ile yazılım arasında uyum, kullanım kolaylığı, diğer

araçlarla uyum sağlaması ve değişik düzlemlerdeki varlığı önemli konulardır. Donanım da çok dikkatli bir biçimde seçilmelidir. Aslında her geliştirici için kendine özgü bir çalışma ortamı ve çoklu ortam gerekmektedir.

- Erişilebilme (Availability): DUP ekibi tarafından bütün ekip üyelerine erişebilmek için gerekli kurallar ve ilkeler düzenlemesi gerekmektedir. Ekibin herhangi bir üyesinin günlük veya haftalık programı tüm ekip üyelerince erişilebilir olmalıdır. Çift Programlama veya Sınama aşamalarında yapılacak olan planlama tüm ekip üyelerinin erişebilme durumunu göz önüne alarak ve en yüksek düzeyde bilgi yayılımını sağlayacak biçimde yapılmalıdır.
- Yönetim(Management): Proje önderi ve üst düzey yöneticiler dağıtık bir ekibin ve değişik yerlerde olan ekip üyelerinin nasıl yönetileceğini öğrenmelidir. Bu işlemi gerçekleştirmek için günlük veya haftalık biçimde bilgiler gönderilmelidir. Bunun yan ısıra ekip üyelerine ekibin bir parçası olduklarına ilişkin duyguyu ulaştırmak için düzenli bir biçimde geribildirimler verilebilir. Ayrıca düzenli ekip ekinlikleri düzenlemek ekip üyeleri arasında güveni ve özendirmeyi (motivation) arttırır (Kircher vd., 2001).

4.6 DUP'de olan UP uygulamaları ve değerleri belirlemek

DUP zorluklarını belirlediğimizde önemli olan DUP'nin uygulamalarının ve değerlerinin çiğnenmemiş olması. Daha önce belirlediğimiz gibi UP uygulamalarından sadece dördü dağıtık bir ekip ortamından etkilenmektedir (Oyun Planlama, Çift Programlama, Sürekli Bütünleştirme ve Yerinde Müşteri ile Çalışmak). Bu bölümde DUP'nin ışığında UP uygulamaları anlatılır ve bu uygulamaların DUP ile gerçekleştirilmesi için sunulan önerileri gösterir.

- Oyun Planlama (Planning Game): Oyun planlama yapmak istediğimizde uzak olan müşteriler için görüntülü toplantı ve paylaşım yazılımlarının desteği çok önemlidir. Örnek olarak paylaşım uygulamaları öykü kartlarını yazmak için kullanılabilir. Başarılı olması için bu işleme iki kişiden fazla kişi katılmalıdır.

• Çift Programlama (Pair Programming): Proje ekibinin üyelerinin değişik yerlerde olduğu durumda çift programlama yapmak için, Uzaktan Çift Programlama (Remote Pair Programming) kullanılmalıdır. Bu işlemi yapmak için ve Tümleşik Geliştirme Ortamını (Integrated Development Environment) sağlamak için, görüntülü toplantı ve paylaşım uygulamaları gereklidir.

• Sürekli Bütünleşme (Continuous Integration): Uzak olan ekip üyesi ayrı bir makinada çalıştığı için bu bütünleşme işlemi tanımlanmalıdır. Merkezde çalışan bir ekip üyesi uzak olan ekip üyesini belli makinada ortak bütünleşmeyi yapmak için davet etmelidir (örnek olarak VPN kullanarak). Eğer her iki üyede makinadan uzakta ise bu durumda bütünleşme yapmak olanaksızdır ve bu durumda her üye kendi makinasında bütünleştirmeyi yapmak zorundadır.

• Yerinde Müşteri ile çalışmak (On-Site Customer): Uzak olan müşteriler için görüntülü toplantı kullanılabilir. DUP'de müşteri gerçek yerinden Müşteri değildir, Sanal Yerinde Müşteridir (Virtual On-Site Customer). Bu durumda müşteri de eşgüdüm ve erişebilirlik gibi bazı kurallara uyum sağlamalıdır. UP'nin temel özellikleri de herhangi bir değişime uğramadan DUP'ye uygulandığına ilişkin emin olmak için dört değerden emin olmalıyız: İletişim, Kolaylık, Geribildirim ve Atılganlık.

• İletişim (Communication): Var olan iletişim araçlarını kullanarak fiziksel uzaklıktan etkilenmeden etkili ve güçlü bir iletişim kurabiliriz çünkü DUP'de de iletişim UP'e kadar önemlidir.

• Kolaylık (Simplicity): Felsefede "Make it simple - Kolaylaştır" kuralı ekip üyelerinin fiziksel yakınlığa dayanmamaktadır o yüzden DUP'de etkilenmemektedir.

• Geribildirim (Feedback): Geribildirim UP'de olduğu gibi DUP'de de çok önemlidir. Temel fark ise, DUP'de biri birinden uzak olan ekip üyelerine bu geribildirimin dağıtılması gerektiğidir. DUP'de iletişim araçları en iyi düzeyde olmalıdır çünkü iletişim araçlarında herhangi bir eksiklik geribildirimlerin yayılmasını etkiler.

• Atılganlık (Courage): Bu değer DUP'de doğrudan doğruya etkili değildir. Bu nedenle, DUP UP'nin dört temel değerini değiştirmemektedir (Kircher vd. , 2001).

4.7 Deneme Bilgisi

Kircher ve diğerlerinin dağıtık ekip çalışması için uygulaması "Masaüstü Web Projesi - Web-Desktop Project" adı altında gerçekleştirildi. Ekip üyelerinin yerleri ise:

Prashant, Hintli, Delhi, Hindistan,

Michael, Alman, Munich, Almanya,

Angelo, İtalyan, St. Louis, ABD,

Catania, İtalya ve Irvine, ABD arasında çalışıyor.

David, Amerikalı, Pittsburgh, ABD

Bu bölümde onları projeleri ve ekip olarak DUP uygulayarak çalışma biçimlerini anlatıyoruz.

4.8 Proje Tanımı

Projemizin amacı DUP için çalışma ortamı sağlayacak Masaüstü Web- Web Desktop adlı yazılım geliştirmektir. Web Desktop, web sayfası üzerinden erişilebileceğimiz bir uygulamadır. Web Desktop, yazılım geliştirme ve süreç yönetiminde kullanılan bir dizin uygulamadan oluşuyor. Bunların yanı sıra isteğe ve gereksinime bağlı başka uygulamalar ve özelliklerde eklenebilir. Projede çalışanlarının çalışma yerleri sürekli değiştiği için kullanılan araç 100% devingenliği desteklemelidir. Bu yazılım, her hangi bir ekip üyesi aynı özellikler, görünüm ve çalışma ortamını uluslararası ağa bağlandığı zaman diğer ekip üyeleri ile bağlantıya geçip işlerine devam edebilir. Ekip çalışanları devingen olarak çalıştığı için araç çalışanlar için gereken esnekliği tanımalıdır. Ekibin her hangi bir üyesi, her hangi bir internet kafeye gittiği zaman ve kamerası ile mikrofonunu kullanarak araca bağlanıp diğer ekip üyeleri ile bağlantıya geçebilmelidir. Bağlantıya geçebilmek için ekip üyelerinin bağlanacağı bilgisayara ne yazılım indirme nede bir yazılım kurulumu yapılmamalıdır. Proje kapsamında, her bir ekip üyesi için rol tanımlandı. Michael, Angelo ve Prashant yazılımcı olarak, David ise müşteri oldu. Süre çok kısa olduğu için (yalnızca 3 hafta) sadece

UP'nin dört uygulamasına odaklanarak onlardan emin olmak istedik (Kircher vd., 2001).

4.8.1 Oyun Planlama

David ile müşterimiz olarak, kullanıcı öykülerini tartışırken bir kaç görüntülü toplantı yaptık. Kullanıcı öykülerini oluşturmak için sıradan bir düzenleyici kullandık ve bir uygulama paylaşım yazılımı yolu ile paylaştık (MS Netmeeting, 2009). Kullanıcı öyküleri yazılımcılar arasında tartışıldı ve değerlendirildi. Son olara, David öncelikleri belirlenmiş ilk yinelemede ortaya çıkan kartlardan birini seçer, bu yinelemeler diğer aşamalarda da devam eder.

4.8.2 Çift Programlama

Kullanıcı öykü kartlarını bir çift yazılım geliştiricisine verdik ve onlar yazılım geliştirmeye başladılar. 6. da olduğu gibi Uzaktan Çift Programlama (Remote Pair Programming) kullandık, Uzaktan Çift Programlama görüntülü toplantı ve uygulama paylaşımında kullanımı yaygın olan bir yöntemdir ve buluşmaları ayarlamak için E-Posta ile haberleşmeyi kullandık.

4.8.3 Sürekli Bütünleştirme

Sürekli bütünleşmeyi yapmak için yapılanış yönetim (Configuration Management) aracı olarak CVS(GNU, 200) kullandık. Değişikleri ise bütünleştirme bilgisayarı bulunmadığı için geliştirme dalından ana dala bilgisayarları değiştirmeden doğrudan bütünleştirdik.

4.8.4 Yerinde Müşteri İle Çalışmak

Proje yaşam döngüsü boyunca kendi aramızda ve müşteri ile görüşmek için görüntülü toplantıyı kullandık. Yaklaşan toplantıların saat ve tarihini belirlemek için ise E-postayı kullanarak iletişim kurduk (Kircher vd., 2001).

4.8.5 Kullanılan Kaynaklar

Biz projemizde iyi desteklenen, kolayca kullanılan ve kolayca çalışma ortamımız ile bütünleşebilen araçlar kullandık. Mümkün olduğunca, değişik düzlemlerde desteklenen araçları veya belli standartlara göre benzer yazılımlarla birlikte çalışabilen araçlar kullanmaya çalıştık. Örnek olarak NetMeeting ve

CuseeMe, ITU konferans standartlarına göre çalışmaktadır ve bunlar bütün masaüstü ve dizüstü bilgisayarlarda, çeşitli mikrofon ve hoparlör ve web kameraları ile çalışabilir. Biz NetMeeting'i (MS Netmeeting, 2009) görüntülü toplantı ve yazılım paylaşımı uygulaması olarak kullandık. Bağlanırlık (connectivity) için birçok bağlantı kullandık: 33Kbps modemler, 64Kbps ISDN, 100Mbps LAN bağlantıları.

4.8.6 Karşılaştığımız Engeller

Projeyi gerçekleştirdiğimiz sırada karşılaştığımız engeller ise:

- Kullandığım görüntülü toplantı yazılımı, NetMeeting, her oturum için iki katılımcıdan fazlasına izin vermemektedir. Toplantıda iki katılımcıdan fazla olduğu zaman ek sunucu kullanmak zorunda kalıyorduk.
- Metin dosyalarını öykü kartı olarak kullanmak oldukça zordu. Daha iyi bir çözüm olarak özel bir WikiWikiWeb (Cunningham, 2001) kullanılabilir.
- NetMeeting uygulaması işlevleri değişik işletim sistemleri arasında uygulama paylaşımını sağlayamıyordu. Daha iyi bir çözüm Sanal Ağ Bilişimi (Virtual Network Computing) (AT&T, 2001) olabilir.
- Beyin fırtınası için yalın bir metin düzenleyici kullandık, buda süreci oldukça hantal duruma getirdi. MindMapper (Bosely Group, 2001) gibi araçları kullanarak yeni düşünceler ile ilgili tartışmaları kolaylaştırır.
- Dar bant bağlantıları (Narrow bandwidth connections), örnek olarak: çevirmeli (Dail-Up)'de görüntülü kullanımı engellenmiştir çünkü seste olan titremek (Jitter) uygulamanın paylaşım duyarlılığını azaltmıştır. Son çare olarak stratejimiz ise sadece sesli görüşmeyi kullanmak ve özel bir sohbet (Chat) kanalı kullanmak.
- Elektrik kesintisi Hindistan'da hâlâ bir sorundur. Bir dizüstü bilgisayar bataryası en azından kısa kesintiler durumunda yardımcı olabilir.
- Örnek olarak kaynak kod deposuna erişim eksikliği büyük bir engel değildir. Ancak bu sorunun sonuçları uzun vadede daha büyük etkileri olabilir. Parshant çoğu zaman güvenlik duvarı arkasından çalışmak zorunda olduğu için ekip havuzuna bağlanamıyordu. Diğer ekip arkadaşlarımız Parshant'a E-posta ile kodun anlık fotoğrafını (snapshot) göndermek zorunda kalıyorlardı. Bu süreç hata eğilimli ve sıkıcı oldu.

- Klavye ayarları bazı ekip üyeleri arasında değişikti. Parantez gibi bazı karakterler sadece iş için kullanılırdı ve toplantıdaki bütün katılımcılar aynı tür klavyeyi kullanırdı, örnek olarak hepsi Alman veya hepsi Amerikan klavyesi kullanırdı (Kircher vd. , 2001).

4.8.7 Alınan Dersler

DUP kullanarak yaptığımız projemiz oldukça başarılı idi ve bu süreç bize bazı değerli deneyimlerinde kazandırdı.

- Proje sırasında daha etkin olmak için görüntülü toplantı gibi anuyumlu veya E-posta gibi zaman uyumsuz birleşimler kullandık. Biz uygulama paylaşımı ile birlikte görüntülü toplantı kullanılsak bile, tamamen UP tarafından sunulan fiziksel yakınlık gibi etkin olamazdı. Ortağın bir görüntüsü o an onun düşündüğünü veya bir tepki vereceğini anlamak için yeterlidir. Buna rağmen, yinede ortağın fiziksel varlığının olması daha iyi oluyor çünkü herhangi bir görüntülü toplantı aracı ortağın fiziksel olarak yanımızda bulunmasının yerini alamaz.
- Paralel yazılım geliştirme kaynak kodun bütünlüğü sorununu gündeme getiriyor. CVS (CVS, 2000) ve ClearCase (Rational Clear Case, 2001) gibi araçlar sorunu belirler. Bu araçlar dağıtık geliştirmeyi desteklese bile yine bazı özel düzenlemeler yapmak anlaşmazlıkları önlemek için yardımcı olur. Bu nedenle, değişikliklere erişebilmek için aynı kod parçası üzerinde çalışanlar E-posta yolu ile haberleşebilir (Kircher vd., 2001).

4.8.8 Sonuç

DUP uzak ve devingen ekip üyelerinin arasında geliştirme işlemi sırasında verimli bütünleşme sağlar ve bu nedenle geleneksel UP için değerli bir uzantısıdır. Ayrıca, yerinde müşteri ile çalışmak olanağını tanıyan DUP proje içerisinde müşteriye daha etkin bir rol tanıyor özellikler fiziksel olarak müşteri ile beraber çalışama durumumuz olanaksız olduğu zamanlarda. DUP yolu ile UP'yi hafif yazılım geliştirme süreci olaraktan kabul ettirebiliriz. Biz bilgisayar destekli etkileşim yoluyla gerçekleştirdiğimiz sanal toplantıların asla doğrudan insanların yüz yüze yapacağı etkileşimin yerini alamayacağının farkındayız, ama yüz yüze iletişimin olanaksız odluğu durumlar için UP çok başarılı bir yaklaşımdır. Daha öncede belirttiğimiz gibi Bilgisayar Destekli İşbirlikli işlemler'de (Computer

Supported Cooperative Work) ağır bir alana değinmekteyiz. Bu konuda istediğimiz konuma gelmek için DUP ile ilgili daha fazla araştırma yapılmalıdır. Bilgisayar destekli etkileşimin başarılı olması için temel konular, canlı görüntü ve ses, yani görüntülü toplantı olarak belirledik. DUP'in gelecekte projelerde nasıl gerçekleştirileceği ile ilgili bir kılavuz hazırlanmalıdır. Burada sunulan çözüm insan etkileşiminden genel bir devrim niteliği taşımaktadır ve bu devrim ne yazık ki uzun zamandır çoklu ortamlarda (multimedia) gerçekleşemedi (Kircher vd., 2001).

5. DAĞITIK UÇ PROGRAMLAMAYI (DUP) DESTEKLEYEN BAZI ARAÇLAR

Kircher, Jain, Corsaro ve Levine'nin satışa hazır bazı araçları kullanarak (MS Netmeeting, CVS, CUSeeMe) Masaüstü Web'i (Kircher vd., 2001) geliştirdi. Yöntemin temel düşüncelerinden biri olan en düşük fiyatla gereken işlemi yapmak için araç geliştirmek yerine hazır araçlar kullandı.

Önerilerine göre, uyumlu olup ve olmayan belli araçlar arasında bağlantı kurarak aynı kod üzerinde değişik kişilerin çalışması sırasında yapılan bütünleştirme işleminde yapılacak sorunları en az düzeye indirebilmek.

Açık kaynaklı (open source) yazılım projeleri sanal ortamlarda geliştirildiği için Maurer ve diğerlerinin (Maurer, 2002) (Maurer ve Mardel, 2002) aklına bu projeleri çözümleme düşüncesi geldi. Çözümlenmesi gereken sorular aşağıdaki gibiydi:

- Yazılım geliştiren ekip üyeleri arasında sanal iletişim, işbirliği ve eşgüdüm nasıl sağlanır?
- Bilgi edinme ve bilgi (information and knowledge) paylaşımı nasıl yapılır?
- Hangi araç bunu destekler?
- Hangi yerlerde eksikler ortaya çıkar?

Bu çözümlemelerin sonucunda açık kaynaklı yazılımların geliştirilmesinde ve bakımından kullanılacak en iyi yöntem uluslararası ağ üzerinden yapılan, bilgi değişimi olduğunu, uluslararası ağa dayalı yapılanış yönetimi sistemler (genellikle CVS) kaynak kod ve belgelere erişmek için kullanılır, web tabanlı sorun izleme sistemleri hatların kaydını tutmak ve iş akışını yöneterek hataları çözmek için kullanılır, tartışma grupları ve E-posta listeleri yeni uzantılar ve özellikler önermek için kullanılır. Görüldüğü gibi geliştiriciler için bilgilerin çoğu çekme durumunda (pullmode) bulunmaktadır. Bu şu anlama gelir, geliştiriciler genelde bilgiyi proje sitesine erişerek alır veya CVS istedikleri zaman ve/veya zamanları olduğunda. İtme durumunda (push mode) yanlış raporlarını dağıtmak, yeni

özellikler tartışmak veya E-posta ile destek durumunda çevrimiçi destek için kullanılır (Maurer, 2002).

Maurer ve diğerleri temel olarak DUP ve açık kaynak yazılım süreçler kullanarak sanal ekiplerin çalışma sürecinin iyileştirilmesini önermişlerdir. UP projelerinde, açık kaynaklı yazılım projelerine göre, çalışma daha eşgüdümlü yürütülmektedir, proje eşgüdümünü (coordination) arttırmak için bazı şeyleri gerçekleştirmek lazımdır:

- Geliştiriciler arasında görev paylaşımı,
- Son günün (deadline) belirlenmesi,
- Eş zamanlı iletişim (eş zamanlı iletişim, oyun planlama ve çift programlamada yüz yüze iletişimin gerçekleştirilmesi için önemlidir),
- Etkin bildirim (Active notification) ve bilgi yönlendirmesini (information routing) sağlamak için sürekli bilgi alışverişi yapılmalıdır.

Süreç uygulaması (process execution) bilgi yönetimi (knowledge management) ile deneyim içeriklerini güncel tutarak bütünleştirilmelidir, bunun yanı sıra örgütsel eğitimin ve çalışan (stuff) değişikliğine karşı her zaman hazırlıklı olmak için süreç uygulaması ve bilgi yönetimi günlük süreçlerle de bütünleştirilmedir.

Çevik Yöntemler'de bilgi paylaşımı geleneksel (traditional) yöntemlerden çok daha verimlidir, Çevik Yöntemler belgeye dayalı bilgi değişimi yerine yüz yüze iletişim kurmayı önermektedir. Ayrıca, Çevik Süreçler 'de bilgi paylaşımı için bazı yüreklendirici (encouraging) uygulamalar bulunuyor, UP durumunda, özgür bırakmak (release), yineleme planları (iteration planing), çift programlama, çift çevirme (pair rotation) ve müşteri ile yerinde çalışmak (on-site customers). Çarpışma durumunda, çarpışma günlük toplantıları, çapraz-işlevsel ekipler ve proje geçmişine dönük işlemler (Project retrospectives) (Chau ve Manrer, 2004).

Özgür bırakmak ve yineleme planları durumunda ise, sistem gereksinimleri ve etki alanı (domain) ile ilgili bilgiler müşteri ile geliştirme ekibi arasında paylaşılır. Ayrıca tartışmalar gereksinimleri düzeltmek ve ekip arası daha iyi anlaşmak için yararlıdır. Çift Programlamayı uygulama sırasında geliştiriciler arasında örtülü bilgi (tacit knowledge) paylaşılır, çift çevirme sırasında ise ekip arasında bilgi değişimi yapılır. Günlük Çarpışma (Scrum) toplantılarında ise ekip üyeleri aralarında proje ile ilgili gelişmeleri ve amaçları paylaşır ve biri birilerine

karşılaştıkları sorunlar ile ilgili bilgi alışverişinde bulunurlar. Çapraz-işlevsel ekip üyeleri aralarında deneyimlerini paylaşarak geliştirme süresini azaltırlar. Proje geçmişine dönük işlemler proje sırasında başarı etkenlerini ve sorunları tanımlamayı kolaylaştırır ve sürekli öğrenmeyi sağlar.

Yukarda görüldüğü gibi Çevik Yöntemlerin bütün araçlarında bilgi paylaşımı gerçekleştiriliyor. Bulunan çeşitli araçlar farklı açılardan dağıtık yazılım geliştirmeyi destekliyor, bu araçlarda DUP ile yazılım geliştirmek içinde kullanılmaktadır. Bu araçların birçoğu açık kaynak araçlarıdır, bunun yanı sıra ticari ürünlerde bulunmaktadır. Bu araçlardan bazıları MILOS (En Kısa Yayılımlı Uzun Vadeli Örgütsel Destek - Minimally Invasive Long-term Organizational Support) , MASE (MILOS Çevik Yazılım Mühendisliği - MILOS Agile Software Engineering), COACH-IT (Bütünleşme ve Sınama için Bileşene Dayalı Çevik İşbirlikçi İşleyicisi - Component Oriented Agile Collaborative Handler of Integration and Testing); Geliştirme Destek Araçlarına örnek olarak: TUKAN, Sangam; Yönetim Destek Araçlarına örnek olarak: xPlanner, XPWeb. Aşağıdaki bölümde her araç kısaca sunulacaktır.

5.1 DUP destekleyen araçlardan örnekler

5.1.1 MILOS: (MILOS, 2009)

MILOS bazı UP uygulamalarını kullanarak çevik yazılım geliştirmeyi desteklemektedir. Oyun planlama sırasında kullanıcı öyküleri oluşturulabilir ve geliştirme yaşam döngüsü sırasında değiştirilebilir. Sistem duyuruları, yinelemeleri, kullanıcı öykülerini ve görevler geliştirme programını içinde tutar. Çift programlama eşgüdümü ve başlatılması MS NetMeeting bütünleştirmeyi kullanarak yapılabilir. MILOS aynı zamanda paketler, sınıflar ve yöntemler için boyut ve karmaşıklık ölçütleri üreten ölçüt yardımcı programını da içerir.

5.1.2 MASE: (Kerlesky,2008),(MASE, 2009)

MASE, DUP'yi desteklemek için aşağıda belirlenen birkaç yeni özellik ile MILOS'un genişletilmiş bir sürümüdür.

- Kullanıcı öyküleri: Kullanıcı öyküleri için yeni bir ürün tipi eklenir ve ne zaman yeni bir kullanıcı öyküsü eklenirse, MASE kendiliğinden bu öyküyü görev listesinde uygulamak için bir görev ekler.
- Sürüm ve yineleme planlama: MASE sürümlerin, yinelemelerin,

kullanıcı öykülerinin ve görevlerin kolaylıkla tanımlanmasına ve değiştirilmesine olanak tanır. Ayrıca projede ne olduğu ile ilgili farkındalık da sağlar.

- MS NetMeeting bütünleştirme: Dağıtık çift programlama ve eş zamanlı iletişim MS NetMeeting ile sağlanır.

Ekip üyeleri MASE sunucusuna bağlanarak ve oturum açarak web tarayıcısını kullanarak çalışma alanlarına kolaylıkla erişebilir. Güncel projelerin listesine, kullanıcı öykülerine, şu anda var olan görevlere, görev tahminine ve çift programlama olanaklarına çalışma alanlarından erişilebilir.

MASE Wiki teknolojinin yardımıyla (Wiki Technology, 2009) bilgi paylaşımını kolaylaştırır. MASE bilgi sağlamayı ve bilgiye erişimi kolaylaştırır. Wiki kullanıcıları sadece bir web tarayıcı kullanarak herhangi bir web sayfalarına erişebilir, göz atabilir, değiştirebilir yapılandırabilir ve güncelleyebilir. Ve ayrıca Wiki teknoloji konuların adı o sayfada yazılı ise wiki sayfasından belirli konuların sayfalarına kendiliğinden bağlantı vererek bakım sorunlarının da üstesinden gelir. Ayrıca MASE herhangi içerikle ilgili tam metin arama yetenekleri de sunar.

5. 1.3 COACH-IT: (Maurer ve Read, 2009)

Ölçeklenebilirlik çevik yöntemler için bir sorundur ama mimari odaklı yaklaşımı kullanarak Schwaber'in "Kalabalığın Kalabalığı" (Scrum of Scrums) (çarpışmaların çarpışması) durumunda olduğu gibi üstesinden gelinebilir. Mimari odaklı strateji ön planlamaya dayalıdır. Bu yaklaşıma göre ekiplerin hangi birimlerde çalışacağını ve en sonunda bu birimlerin nasıl bütünleştirileceği tanımlamaktadır. Bunun için çevik yöntemlerin önerdiği bir durum değildir. Ama ön planlama sadece gerekli olana odaklanarak çevik bir şekilde yapılabilir. Sürekli bütünleme sadece birimler için yapılmaz aynı zamanda bağımlılıklarını ve ilişkilerini açıklamak için birimler arasında da yapılabilir. Girdilerin ve çıktıların daha iyi anlaşılması için sınamalar birim sağlayıcıları tarafından değil birim kullanıcıları tarafından yazılmalıdır.

Özet olarak, mimari odaklı yazılım aşağıda belirtilen ilkeleri izleyerek çevik ruhu koruyarak çevik yazılım geliştirme süreçleri ile birleştirilebilir:

1. Birim mimarisini hızlı ve kolay biçimde tasarlamak,

2. Mimari değişikliklere açık bir biçimde sunmak,

3. Ara yüz düzeyinde bir sınama modeli geliştirmek

4. Sınamalar birim sağlayıcıları tarafından değil birim kullanıcıları tarafından yazılması,

5. Sınamacılara bağımlı birimlerde müşteri gibi hareket etmek (müşteri ile yerinde çalışmanın yanı sıra),

6. Birim ekiplerinin kullanıcı öyküleri birikimlerine dayanarak ürünü tanımlaması,

7. Birimler arası sürekli bütünleştirme işlemini yapmak.

COACH-IT aşağıda belirtilen sıra ile gerçekleştirilir:

1. Kullanıcılar COACH-IT girdi web uygulamasını (the COACH-IT input web application) kullanarak bir mimari tanımlar.

2. Her birimde kod değişiklikleri yapmak için çoklu havuzlar izlenir.

3. Değişiklik saptandığı zaman birim ve ilgili birimler indirilir

4. Birimler uygulanır ve ara yüz ile uyumluluğundan emin olmak için sınanır

5. Ekiplere elektronik posta ile sorunlar hemen bildirilir

6. Sistemin sağlığı hakkında web sayfası ile ekiplere bilgi verilir.

COACH-IT araçlarını ile jUnit sınamaları bir bütün olarak kullanılır (CVS, 2011). Böyle birimler geliştiriciler tarafından hemen tanımlanabilir. Birim adları ve (isteğe bağlı olarak) tanımlar/ek açıklamalar, birim havuzlarının yerleri, birim dosyasının yerleri, birim ara yüzleri, birim ekibi iletişim bilgileri (e-posta), birim ilişkileri (tek yönlü), ilişki sınama ilişkileri, havuzunun yerlerini sınamak, dosyasının yerlerini sınamak ve iletişim bilgilerini (e-posta) sınamak COACH-IT'in çekirdeği olan Mimari tanımlama dil dosyasında tanımlanır. Sadece bu öğeler sürekli bütünleştirme ve basit bir mimari yaratmak için kullanıcı girdisi olarak gerekir. Hız Denetim Düzeni (Curise Control)(Cruise Control, 2009) sürekli bütünleştirme aracı birimleri izlemek için kullanılır ve ANT oluşturma dosyası birimde bir değişiklik göründüğü zaman bütünleştirme ve sınamayı yapmak için çağrılır. Bireysel geliştiricilere ve ekiplere de sistem sağlığında değişiklik yâda sınamanın başarısız olması durumunda elektronik posta kullanarak doğrudan bilgi verilir.

5. 1.4 TUKAN: (Stotts vd., 2002)

TUKAN, UP dağıtık ekipler ile yapıldığı zaman ortaya çıkan sorunları

çözmek için UP etki alanına birkaç ekip yazılımı araştırma sonuçlarını uygulayan eş zamanlı dağıtık ekip-programlama ortamıdır. Ekip yazılımı paylaşılan ortama ara yüz sağlayan ve ortak görevle (ya da ortak amaçla) ilgilenen kişi gruplarını destekleyen "bilgisayar tabanlı sistemler" olarak tanımlanır. İletişim, eşgüdüm ve işbirliği UP'de yazılım ekibini yönetmek ile ilgili en önemli üç sorundur. İletişim eşzamanlı olamayabilir (e-postada olduğu gibi) yâda eşzamanlı olur (görüntülü toplantıda olduğu gibi) gerçekleştirilebilir. Eşgüdüm sistemi, geliştiricilerin eylemlerini planlamak, görüntülemek ve uygulamak için gereklidir, aynı zamanda eşgüdüm sistemi iş akışı yönetim sistemlerini, belge yönetim sistemlerini ve sürüm yönetim sistemlerini içerir. İşbirliğinde, temel iletişim desteği ve eşgüdüm desteği paylaşılan bir ortamda birleştirilir. Ekip yazılımı bölümlerinin yanı sıra TUKAN sanal uzaklığa bağlı olarak birkaç renk kodu kullanarak çalışma alanı ile ilgili değişiklikleri destekler.

5. 1.5 SANGAM: (Sangam, 2009)

Sangam dağıtık çift programlama için geliştirilmiş Eclipse IDE (Eclipse, 2009)'nin uzantısıdır. İstemci – sunucu çifti arasında Eclipse ortamının paylaşılmasını ve aynı düzenleyicide kod yazma alışverişini sağlar.

5. 1.6 UPWeb: (XPWeb, 2009)

UPWeb, UP projeleri için web tabanlı yönetim aracıdır. Oyun planlamayı destekler: kullanıcı, kullanıcı öykülerini tanımlayabilir, bunları görevlere ayırabilir, üyeleri görevlere atayabilir ve onları değerlendirebilir. Daha sonra kullanıcı yinelemeler yaratır ve yinelemeler için kullanıcı öyküleri ve görevler seçer. Sürekli olarak uygulama ilerlemesini güncelleyerek ve kendiliğinden hesaplanmış kuramsal ilerleme ile karşılaştırarak geri bildirimleri iyileştirir. Benzetme (metaphor,), CVS, sınamalar ve diğer belgeler UP kaynaklarının merkezileştirilmesini sağlar. UPWeb çok güçlüdür çünkü planlama, süreç izleme, belgelendirme ve sınamayı tek ortamda bütünleştirir. UPWeb Tamamen UP uygulamalarını izler ve o biçimde dağıtık geliştirmeyi kolaylaştırır. Ayrıca her özellik ve kullanımları ile ilgili yardım konuları ile de desteklenir.

5. 1.7 xPlanner: (xPlanner, 2009)

XPlanner Uç Programlama (UP) ekiplerin için proje planlama ve izleme aracıdır. Xplanner, UP'nin planlama sürecini desteklemek için geliştirilmiştir.

Xplanner, kolay planlama modeli; sanal not kartları; projeleri, yinelemeleri, kullanıcı öykülerini ve görevleri kaydetme ve izleme desteğini, bitmemiş öyküleri

akıllıca devam ettirme; dağıtık bütünleştirme simgesi (eposta bildirimi ile), çevrimiçi süre izleme ve birey/ekip düzeyinde süre sayfası (time sheet) oluşturma, ölçüt oluşturma (Ekip hızı, bireysel saatler, v.b), yineleme hızı çizelgeleri, notları öykülere ve görevlere ekleme özelliği (ekler ile birlikte), yineleme tahminin doğruluğunu görüntüleme, sayfa görüntüleme görevi ve bireysel geliştiriciler ve müşteriler için öykü durumunu göstermeyi sağlar.

XPlanner sanal not kartları, bildirimlerin kullanılması ve kullanıcı öykülerine notlar ekleyerek bir tür iletişim iyileştirmesi sağlar. Ayrıca farklı düzeylerde ölçütleri görüntüler bu da olası sorunların ilk aşamada saptanması ve çözümlenmesi için yararlı olabilir.

Çizelge 5.1 Seçilmiş UP uygulamaları için ilgili araç desteği yer almaktadır

UP Uygulamaları	**Araç desteği**
Oyun planlama, Yerinde müşteri ile Çalışmak	CSCW (Computer supported cooperative work – Bilgisayarla İşbirliği Çalışmalarını Destekleyen) sistemleri, ekip yazılımı, belge ve uygulama paylaşımı, programların ve atamaların dağıtık yönetimi
Küçük yayımlar, sınamalar, sürekli bütünleştirme, toplu sahiplik,	Dağıtık yazılım yapılandırma yönetimi (SCM) sistemleri; hızlı yapım süreçleri; otomatik ve hızlı sınama; sınama durumlarının, sonuçların ve geri bildirimin yönetilmesi, rapor hazırlama
Programın içyapısını yeniden düzenleyerek iyileştirme, basit tasarım	Programın içyapısını yeniden düzenleyerek iyileştirme kaynak kodu ve model desteği, bölümsel tasarım bilgilerini işleme, tasarımın

Çift programlama, açık Çalışma alanı	Çoklu kullanıcı düzenleyicileri, uziletişimli toplantı (teleconferencing); belge ve uygulama paylaşımı, fiziksel nesnelerin sanal gösterimi

Kichler ve diğerleri Tablo 1'de çevik yöntemler için yukarıda belirtilen araç gerekliliklerine göre (Ketler vd., 2002) UP uygulamaları için ilgili araç desteğini özetlemiştir. Pek çok görev için araç desteğinin zaten var olduğundan fakat ele alınması gereken çok sayıda gerekliliklerden söz ederler. Çevik yazılım geliştirme ortamının kullanımının kolay olması, otomasyon ve bilgi alışverişi için, işbirliğini, eşgüdümü desteklemek için ek hizmetler sunması ve yeterli esnekliğe sahip olması gerektiği biçimde sonuçlandırılır

5.2 Araçların değerlendirilmesi

Araçlar, çevik yazılım geliştirme yöntemlerindeki ana konulara göre değerlendirilir. Ölçeklenebilirlik, çevik ekipler genellikle 8-12 kişide oluşur. Dağıtıklık, çevik ekipleri yan yana çalışmalıdır. Ve aynı zamanda iletişim, işbirliği ve eşgüdüm çevik yazılım geliştirme yöntemlerinin ana değerleridir ve temel olarak alınır. Kircher ve diğerlerinin belirttiği üzere ekiplerin dağıtık olduğu durumlarda, oyun planlama, çift programlama, sürekli bütünleştirme, yerinde müşteri ile çalışmak özel bir biçimde desteklenmesi gereken UP uygulamalarının birkaçı ve aynı zamanda değerlendirme ölçütleri olarak alınmaktadırlar. UP uygulamalarını değerlendirme ölçütleri olarak alma nedenlerinden bir tanesi ise var olan araçların çoğunun UP uygulamalarına ve ilkelerine göre geliştirilmesidir.

Web teknolojilerinin böyle bir duruma ulaşmasından dolayı dağıtıklık temel ve belki de üstesinden gelinmesi en kolay sorundur. Ayrıca web teknolojilerinin iletişim desteği de ekibin aynı yerde çalışma gerekliliğini azaltmak ve belki de ortadan kaldırmak için web tabanlı araç oluşturma gerekliliğini oluşturur. Dağıtık sorunlarının üstesinden gelmek için ve çevik yöntemlerin gücü büyük ölçüde kişi odaklı davranışlara bağlı olduğu için araçlar ekiplerde ortaya çıkabilecek iletişim,

işbirliği ve eşgüdüm sorunları için çözüm sunmalıdır. Var olan araçların çoğu işbirliğini ve eşgüdümü desteklemek için tasarlanmıştır ve bunların kullanımı ile genel olarak iletişimin desteklenmesi amaçlanır. Dağıtık sorunlarının üstesinden gelmek için ve çevik yöntemlerin gücü büyük ölçüde kişi odaklı davranışlara bağlı olduğu için araçlar ekiplerde ortaya çıkabilecek iletişim, işbirliği ve eşgüdüm sorunları için çözüm sunmalıdır. Var olan araçların çoğu işbirliğini ve eşgüdümü desteklemek için tasarlanmıştır ve bunların kullanımı ile genel olarak iletişimin desteklenmesi amaçlanır (XPWeb, COACH-IT).

Bunların birkaçı ek araçların yardımı ile iletişim desteği sunar (MILOS, MASE, her ikisi de MS NetMeeting bütünleştirmeyi kullanır). Xplanner ekip üyelerini bilgilendirmek için dağıtık bütünleştirme simgelerini (eposta bildirimi ile) kullanır. TUKAN temel ses ve sohbet desteği ile donanımlıydı ve aynı kod bölümü üzerinde çalışan ya da birbirini etkileyebilecek kod bölümleri üzerinde çalışan ekip üyelerini bilgilendiren farkındalık desteğini sağlar. Eşzamanlı olmayan iletişimin kolay bir yolu olarak bazı araçlar ekip üyeleri arasında bilgi ve konu paylaşımını sürdürmek ve yönetmek için Twiki desteğini kullanır (MASE, Xplanner). Araçların çoğu UP'nin ilkelerine ve uygulamalarına göre yapıldığı için oyun planlama, çift programlama, sürekli bütünleştirme ve yerinde müşteri ile çalışmak gibi yan yana yerleştirilmiş ekipleri gerekli kılan UP uygulamalarının üstesinden gelmek için çözüm bulması gerekir. Verilen araçların çoğu proje planlama ve izleme için yapılır bundan dolayı oyun planlama da benzer şekillerde gerçekleştirilir. İlk olarak veri tabanına kullanıcı öyküleri eklenir, proje yöneticisi ekip üyelerini seçer, yayım planı yaptıktan ve kullanıcı öyküleri yinelemelere eklendikten sonra kullanıcı öyküleri ekip üyeleri ile ilişkilendirilir ve ayrıca ekip üyeleri bazı kullanıcı öyküsü uygulaması için eşleştirilebilir. xPlanner sanal not kartı ve notları kullanıcı öykülerine iliştirme özelliğine sahiptir bu da oyun planlama aşamasını kolaylaştırır. Çift programlama MILOS ve MASE'de MS NetMeeting bütünleştirme ile desteklenir fakat TUKAN çift programlamayı kendisi geliştirme ortamı sunarak destekler. Ayrıca SANGAM sadece Eclipse ortamı üzerinde çift programlama gerçekleştirmek için yapılır. Sürekli bütünleştirme CVS havuzlarının kullanımı ile araçların çoğu tarafından desteklenir. Yerinde müşteri ile çalışmak tüm araçlar web tabanlı olduğu için sorun değildir ve müşteriye herhangi zamanda web teknolojilerini kullanarak ulaşılabilir. Ölçeklenirlik çevik yazılım geliştirme yöntemlerinin

büyük ekiplere ve büyük projelerle başarılı olup olamayacağını sorgulayan bir konudur. COACH-IT ekiplere iletişim kurulabilen ve eşgüdümün sağlandığı merkezi mimariyi kullanarak bu sorunun üstesinden gelmeye çalışan tek araçtır. Araçların çoğu UP'nin ilkelerine ve uygulamalarına göre yapıldığı için oyun planlama, çift programlama, sürekli bütünleştirme ve yerinde müşteri ile çalışmak gibi yan yana yerleştirilmiş ekipleri gerekli kılan UP uygulamalarının üstesinden gelmek için çözüm bulması gerekir. Verilen araçların çoğu proje planlama ve izleme için yapılır bundan dolayı oyun planlama da benzer şekillerde gerçekleştirilir. İlk olarak veri tabanına kullanıcı öyküleri eklenir, proje yöneticisi ekip üyelerini seçer, yayım planı yaptıktan ve kullanıcı öyküleri yinelemelere eklendikten sonra kullanıcı öyküleri ekip üyeleri ile ilişkilendirilir ve ayrıca ekip üyeleri bazı kullanıcı öyküsü uygulaması için eşleştirilebilir. xPlanner sanal not kartı ve notları kullanıcı öykülerine iliştirme özelliğine sahiptir bu da oyun planlama aşamasını kolaylaştırır. Çift programlama MILOS ve MASE'de MS NetMeeting bütünleştirme ile desteklenir fakat TUKAN çift programlamayı kendisi geliştirme ortamı sunarak destekler. Ayrıca SANGAM sadece Eclipse ortamı üzerinde çift programlama gerçekleştirmek için yapılır. Sürekli bütünleştirme CVS havuzlarının kullanımı ile araçların çoğu tarafından desteklenir. Yerinde müşteri ile çalışmak tüm araçlar web tabanlı olduğu için sorun değildir ve müşteriye herhangi zamanda web teknolojilerini kullanarak ulaşılabilir. Ölçeklenirlik çevik yazılım geliştirme yöntemlerinin büyük ekiplere ve büyük projelerle başarılı olup olamayacağını sorgulayan bir konudur. COACH-IT ekiplere iletişim kurulabilen ve eşgüdümün sağlandığı merkezi mimariyi kullanarak bu sorunun üstesinden gelmeye çalışan tek araçtır. Tablo 2'de bazı var olan DUP araçlarının karşılaştırması özetlenmektedir:

Çizelge 5.2 Kullanılan DUP araçlarının birkaçının karşılaştırması

Araç / Destekler	MILOS	MASE	COACH-IT	TUKAN	Sangam	XPWeb	xPlanner
Oyun planlama	2	2	2	2	0	2	2
Çift programlama	0	1	0	3	3	0	0
Sürekli bütünleştirme	1	1	3	1	0	3	2
Yerinde müşteri	1	1	1	2	0	2	3
Ölçeklenirlik	0	0	2	0	0	2	2
Dağıtım	2	3	2	3	2	3	3
İletişim	1	3	3	3	2	2	3
Eşgüdüm	3	3	3	3	0	3	3
İşbirliği	1	1	2	3	3	2	3
Ekstralar	MS NetMeeting Bütünleştirme	MS NetMeeting Bütünleştirme	Hız Denetim Düzeni jUnit, ANT bütünleştirme	Farkındalık desteği	Yok	CVS ile bütünleştirme; belgelendirme desteği	Twiki desteği

0: Desteklemez 1: Ek araç yardımı ile destekler 2: Destekler 3: Tamamen destekler

5.3 Kullanılacak Araçlar Nasıl Olmalıdır?

- Araç web tabanlı olmalıdır ve çift programlama gibi belirli geliştirme konuları dışında yüksek bant genişliği gerektirmez.
- Araçta kullanıcı öyküsü yaratma, oyunu planlama ve yayım planlama sırasında üyeler arasında iletişim kurulması için web tabanlı iletişim ya da bir tür duyuru paylaşımı olmalıdır,
- Araç kullanıcı dostu olmalı, erişimi kolay özelliklerinin öğrenilmesi ve bu özelliklerle ilgili yeterli belgelendirmeyi desteklemelidir.
- Sınama ekipleri ve günlük yapımlar için kullanılacak araçlar yolu ile bir tür raporlama süreci gerçekleştirmelidir,
- Kullanılan araçlar birkaç ölçüt tutmalı ve bunları kullanarak projeyi izlemelidir. Ve aynı zamanda geçmiş projelerin ölçütlerini tutan ve bunları güncel proje için deneyim temeli olarak kullanılmalıdır,
- Proje sırasında araçla desteklenen sesli iletişim varsa bu geliştirenlerin ilgisini dağıtmadan iletişimin etkinliğini ve anlaşılabilirliği arttırabilir,
- Yazılımların ersimi kolay olmalıdır ve en düşük maliyetli veya ücretsiz olmalıdır,
- Görevlerin özellikleri hakkında araç bir tür geçmiş tutmalıdır ve bir tür duyuru paylaşımı yâda bilgi paylaşma ortamı olmalı ve her ikisi de ekip üyeleri arasındaki iletişimi iyileştirmek ve geliştiricilere yardımcı olmak için bir tür arama motoru ile donanımlıdır,
- Kircher ve diğerlerinin uygulamasında karşılaştıkları sorunları çözmelidir.

6. 4S BİLGİ TEKNOLOJİLERİNİN ASIACELL PROJESİ

6.1 Giriş

4S Bilgi Teknolojileri kısa zaman öncesine kadar ürün odaklı satış amaçları olan bir firma idi. Günümüzün gerekliliklerine yanıt verebilecek yapıda, yenilikçi, hızlı, ileriyi de bir yapıda ilerleyen, katma değerli projelere imza atan, sunduğu çözümler ve gösterdiği amaçlar sonucunda kendisine sektörde yer edinmiş bir firma olarak faaliyet göstermektedir. 4S Bilgi Teknolojileri, önümüzdeki süreç içerisinde de Sistem bütünleştirici görevini ve çözüm üreten politikasını sürdürecek, yatırımlarını arttırarak emin adımlarla ilerleyecektir.

4S Satış ekibi, bilişim çalışmalarına destek verebilecek e-devlet uygulamalarına en etkin ve güvenilir çözümleri sunmak konusunda üstlendiği öz görevin farkındadır. 4S, E-Türkiye çalışmalarında üreten ve çözüm sunan taraf olarak rolünü üstlenmiş, uzmanlaştığı konularda kamuda marka olmayı başarmış az sayıdaki firmalardan birisidir. Bu konuda önceliğin müşteri gereksinimlerini doğru çözümlemek ve doğru çözümü sunmak olduğunun farkında olan satış ekibi, ihtiyaçlar karşısında ileriyi de kapsıyor bir davranış ile müşteri gözünde en uygun ve gelişmeye açık, ölçeklenebilir çözümleri bir araya getirip, beklentileri tam ve eksiksiz olarak karşılayacak açılımlar sunabilmektedir.

Gereksinimlerin ortaya çıkması durumunda; satış öncesi danışmanlık, uzmanlaştığı konularda Ar-Ge ve ‘Anlayışın Doğrulaması – Proof of Concept’ hizmetlerini sunabilecek bir ekip yardımı ile beklentileri karşılayacak en doğru çözümleri sunmak, 4S Satış ekibinin temel amacıdır (4S Bilgi Teknolojileri, 2011).

4S Bilgi Teknolojileri 2001 yılından beri iş teknolojileri alanlarında faaliyet göstermektedir ve 2009 yılında MEMA (Orta Doğu, Akdeniz ve Afrika) bölgesinde “HP Yılın İTE İş Ortağı” (İş Teknolojileri en iyileştirilmesi) ödülünü almaya hak kazandı ve günümüzde de Irak, Bahreyn, Mısır, Kuveyt ve birçok ülkede projelerini yürütmektedir.

4S Bilgi Teknolojilerinin Türkiye'de Ankara, İstanbul ve Edirne teknokentte, yurtdışında ise Dubai Internet City'da ofisleri bulunmaktadır. Ayrıca Bulgaristan'da bir destek ekibi vardır. Bizim Projemiz yoğun olarak İstanbul'da bulunan İş Teknolojileri eniyileştirilmesi (Business Technology Optimization (BTO)) ekibi ile birlikte yürütülmektedir. Dağıtık Uç Programla ile ilgili uygulamayı 4S Bilgi Teknolojilerin Irak'ın en büyük GSM operatörü olan Asiacell ile gerçekleştirdiği projeyi örnek gösteriyorum.

6.2 Michael Kircher ile yazışmalarım

Dağıtık Uç Programlama düşüncesini ortaya atan kişi (Michael Kircher) ile yazışmalarım sırasında bana bu düşünceyi uygulamak için ekip üyeleri için gereken iki özellik olduğunu söyledi:

1. Ekip üyelerinin deneyimli olması,
2. Ekip üyeleri biri birilerini iyi tanıması,

Bu iki noktanın 4S Bilgi Teknolojileri çalışanları arasında olması beni bu projeyi Dağıtık Uç Programa yaklaşımı ile gerçekleştirmek için 4S Bilgi Teknolojileri yönetimine düşüncemi sundum.

6.3 Neden Geleneksel Süreçler yerine Çevik Süreçler seçildi?

Gerçekleştirilen yazılım projelerinin temel sorunları olan müşteri gereksinimlerinin anlaşılmaması ve bu gereksinimlerinin sürekli değişmesi artık geleneksel süreçlerle çözülmemektedir. Bu sorunların çözülmesi için yazılım geliştirme yöntemlerinin daha çok müşterinin isteyeceği ani değişiklikleri karşılayabilecek ve müşterinin anlayamadığı gereksinimlerinin anlamasında yardımcı olabilecek biçimde olmalıdır, bunun gerçekleştirmek için projenin bütün aşamalarında, Çevik Süreçlerin yaptığı gibi, müşteri ile beraber çalışarak veya müşteri ile sürekli iletişimde olarak sağlanır. Bu nedenle Çevik Süreçler, geleneksek süreçlere göre günümüzde hızlı değişen müşteri gereksinim ve isteklerine daha çok ayak uydurmaya yardımcı olmaktadır ve daha çok kullanılmaktadır.

6.4 Çevik Süreç modellerinin arasından neden Dağıtık Uç Programlama seçildi?

Çevik Süreçlerde temel ilkelerden biri hem müşteri ile hem de proje ekibi kendi aralarında fiziksel olarak aynı çalışma ortamında olmalıdır. Bütün Çevik Süreçler yöntemlerine göre de müşteri ile fiziksel olarak beraber çalışma zorunluluğu vardır. Fiziksel olarak beraber çalışma zorunluluğu ortadan kaldırma olanağı tez konum kapsamında incelediğim kadarı ile bütün Çevik Süreçler yöntemleri arasında sadece Dağıtık Uç Programlamada görünmektedir. Günümüzde gelişen iletişim araçlarını projeye süresi ile maliyetini en alt ve niteliğini en üst düzeye çekebilmek için, bu iletişim araçlarını etkin bir biçimde kullanarak projemizi Dağıtık Uç Programa ile gerçekleştirdik. Bu kapsamda ulaşım sıkıntıları ve özel nedenlerden dolayı müşteri veya ekip arkadaşları ile aynı ortamda bulunamayan ekip üyelerimiz içinde bu iletişim araçlarını etkili kullanımı yararlı olmuştur.

6.5 Proje Tanımı

Bugününkü Telekom sektöründe, artan rekabet ve hızlı teknolojik değişimler nedeni ile dinamik ve hızla bir biçimde değişen iş ortamı bulunmaktadır. Bu ortamda olan büyüme ve çeşitlendirilme eşi görülmemiş fırsatlar yaratmaktadır. Ancak, aynı zamanda her geçen gün büyüyen altyapı ve görünürlük eksikliği iş faaliyetleri üzerindeki etkisi nedeniyle gerçek bir tehlike oluşturmaktadır. Artan kesintileri, azalan verimlilik ve yetersizlik Telekom sektörünü olumsuz bir biçimde etkilemektedir. Karlılığını korumak ve rekabet ortamında kazanmak için, Telekom operatörleri maliyetleri denetlemek için ve verdiği hizmeti iyileştirmek için üst düzeyde bir performans sunmaya devam etmelidir.

HP, Asiacell Grup yönetiminden anahtar paydaşları ve BT ekibi ile yapılan görüşmelerde ve iş gereksinimlerini anlamak için bir dizi görüşmeler yaptı ve bu görüşmeler sonucu Asiacell'in gereksinimlerini karşılayacak olan HP yazılımlarını belirledi.

HP ile Asiacell arasında yapılan anlaşmada verilecek olan yazılımların:

- Hizmet niteliğini yükseltmek,
- BT yönetimi için harcanan çabayı azaltmak,
- Altyapıyı kullanırlığını ve başarımını ölçmek,
- Son kullanıcı memnuniyetini ve kritik servisler için en iyi başarım (performance) sunabildiğini ölçmek,
- Hizmetin kullanırlığını ve başarımını izlemek.

HP yaklaşımı ile Asiacell çalışanlarının ekip halinde çalışması ve bilgi paylaşımının artması sağlar. Bu biçimde Asiacell çalışanları çözümleri ve süreçleri daha iyi anlar ve daha verimli ve başarılı bir destek vermelerini sağlar.

6.6 Kullanılan HP Yazılımları

HP'nin çok geniş bir ürün yelpazesi bulunmaktadır, bu yazılımlardan hangisinin Asiacell'in gereksinimlerini karşılayacağını anlamak için çok ayrıntılı bir gereksinim çözümlemesi yapıldı. Bu çözümleme yapıldıktan sonra kullanması gereken yazılımlar belirlendi. Bunlar:

6.6.1 Ağ Düğümü Yöneticisi (Network Node Manager (NNM))

(Günaydın, 2011).

HP Ağ Düğümü Yöneticisi (NNM) kurum içerisinde sağlıklı bir ağ kurmak için kullanılan bir yazılımdır. NNM sürekli olarak ağ düğümlerini (anahtarlar ve yönelticiler gibi) bulmak için kullanılır ve bir ağ ilingesi (network topology) kullanarak bu bilgileri sürekli güncelleştirir.

NNM ağ için doğru bir görüntü tutar, ayrıca olay ilintisi (event correlation) ve kök neden çözümlemesi (root cause analysis (RCA)) kullanarak ağ yönetimi ile ilgili sıra dışı durumları ve sorunları belirleme yeteneği vardır.

Diğer ağ yönetimi yazılımlarından farklı olarak, NNM sürekli olarak gelişmiş RCA algoritması uygulamaktadır, sürekli değişen ağ ilingesi devingen aksaklık (fault) yönetimini desteklemek için kullanmaktadır. NNM işletmeler için bunları sağlamaktadır:

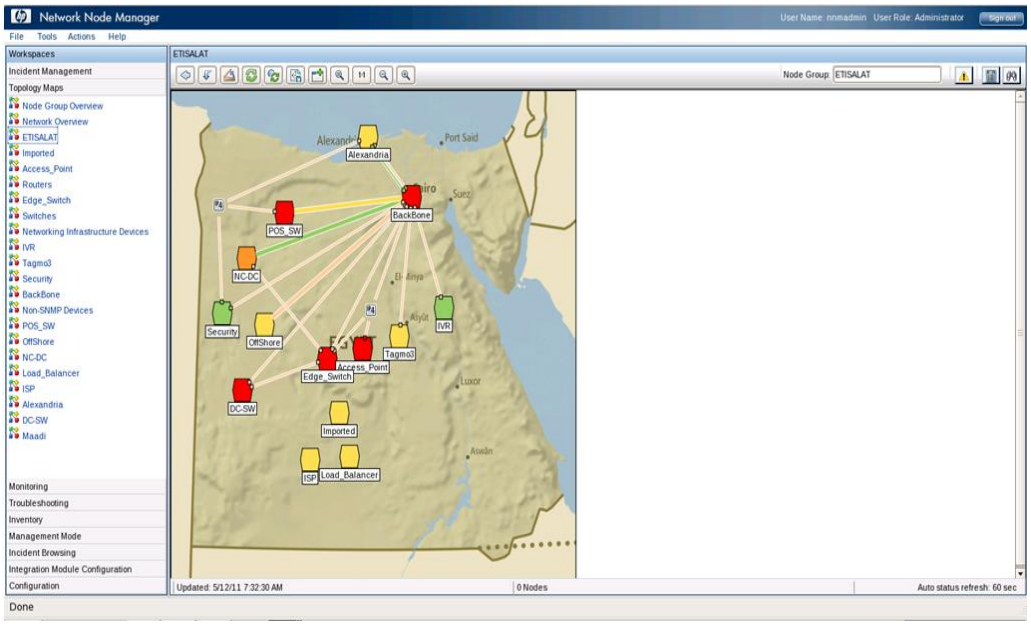
6.6.1.1 İşlevsel Özellikler

- Düşük maliyet ile en fazla kullanırlıkla ağ teslimi,

- Kendi ağının altyapısını sağlamlaştırmak (örneğin yönetim sunucu sayısını azaltmak),
- Yüksek özellikli ve düşük maliyetli bir üyelik güvenceye almak,
- Akıllı otomasyon kullanarak daha fazla çalışan verimliliğine ve etkinliğine ulaşmak ve yeni bir kullanıcı ara yüzü,
- Belirleyici (deterministic) ve uyabilen nedensel motor ve diğer akıllı otomasyon özelliklerini kullanarak MTTR'yi azaltmak.

6.6.1.2 Özellikler

- N-katmanı mimarisi,
- Kural dışı durum (exception) yönetimi
- Sürekli sarmal bulgu (discovery)
- Devingen ağ üzerinde doğru kök neden çözümlemesi (root cause analysis),
- Zengin web kullanıcı arayüzü,
- ITIL olay yönetimi,
- Tümleşik aksaklık (fault) ve başarım yönetimi,
- Değeri kısa sürede bulmak,
- Yüksek ölçeklenebilirlik. (Bkz. Şekil 6.1)



Şekil 6.1 Ağ Düğümü Yöneticisi (Network Node Manager (NNM)).

6.6.2 HP Evrensel Yapılanış Yönetimi Veri Tabanı (HP Universal Configuration Management Data Base (HP uCMDB)) (Günaydın, 2011).

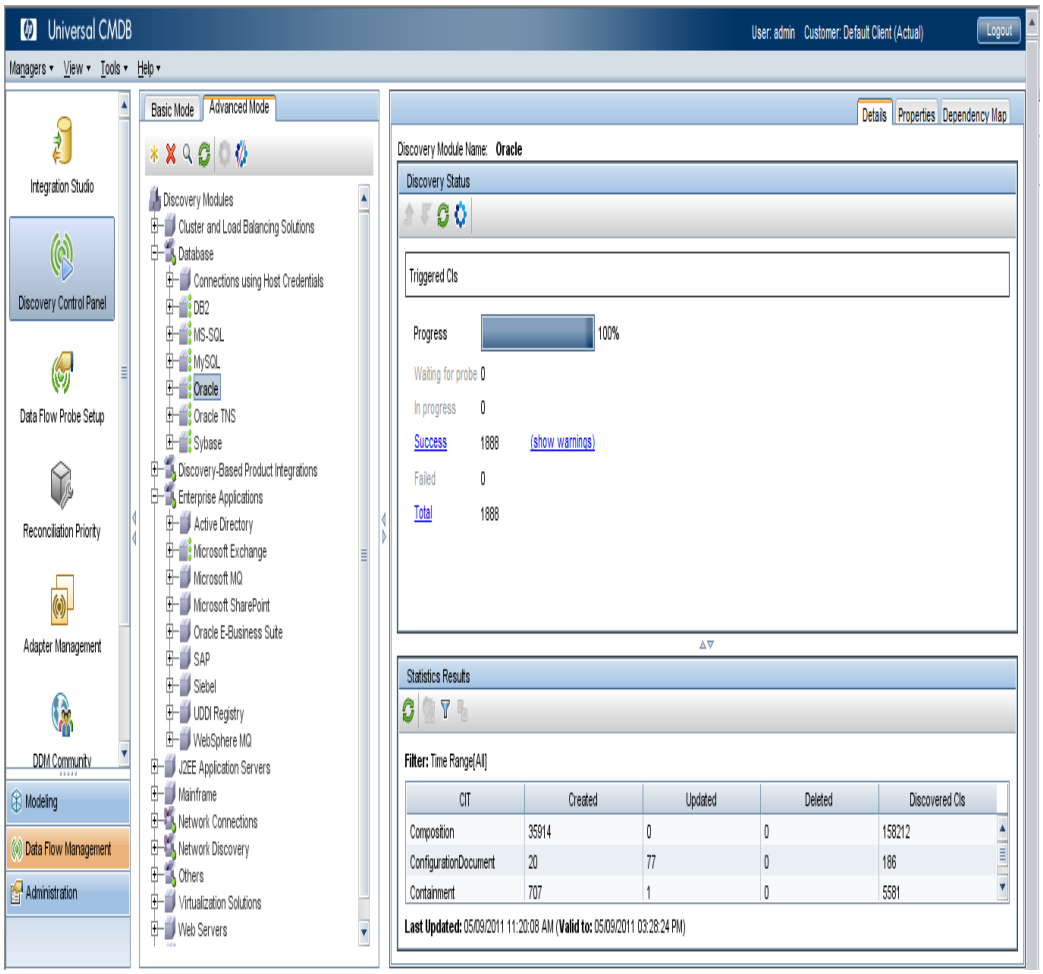
6.6.2.1 İşlevsel Özellikler

HP Evrensel Yapılanış Yönetimi Veri Tabanı (uCMDB) kurumsal BT kurumlarının iş hizmeti tanımı ve ona bağlı olan altyapı arasındaki ilişkileri belgelendirmek ve tutmak için bir yapılanış yönetimi veri tabanıdır (Configuration Management Data Base).

CMDB kişiler ve altyapı, uygulamalar ve iş hizmetleri arasındaki ilişkiler ile ilgili tek bir sürüm ile otomatik işlemlerin gerçekleştirilmesini sağlar. Bu yazılım, diğer bütün HP yazılımları ile sorunsuz ve bütünleşik bir biçimde çalışır, otomatik olarak altyapı ve uygulama ilişkileri ile ilgili olarak güncel ve doğru bilgiler tutmaktadır. uCMDB HP'nin yazılımları kendi aralarında bütünleşmeleri için anahtar bir yazılımdır.

6.6.2.2 Teknik Özellikler

- BT'nin varlıklarını (assests) anlayarak iş hizmetleri üzerinde olan etkisini ve iş hizmetlerini nasıl denetlediğini görüntülemek,
- BT ve iş ekipleri arasında iş birliğinin geliştirmesini sağlamak,
- Yapılan değişikliklerin etkisini etkili bir biçimde çözümleyerek çekinceleri (risk) azaltmak.
- Teknik depolar üzerinde olan baskıyı azaltmak,
- Otomasyon ve ön yapılanış aşamalarında süreyi azaltmak. (Bkz. Şekil 6.2)



Şekil 6.2 HP Evrensel Yapılanış Yönetimi Veri Tabanı (HP Universal Configuration Management Data Base (HP uCMDB)).

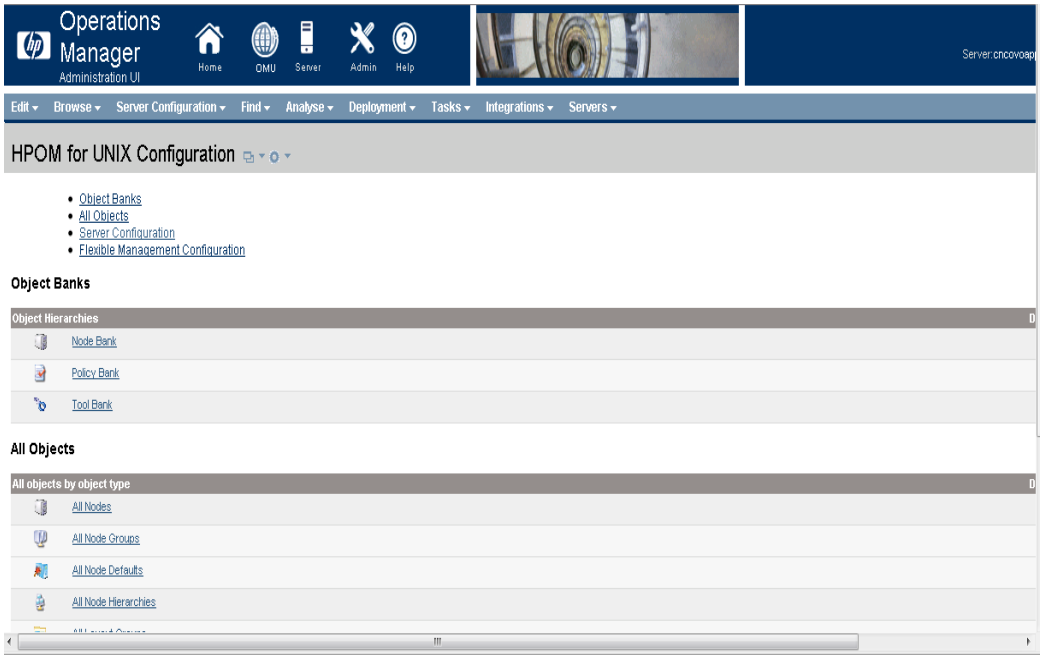
6.6.3 HP İşlemler Yönetimi (HP Operations Manager (OM)) (Günaydın, 2011).

6.6.3.1 İşlevsel Özellikler

HP İşlem Yönetimi (OM), kapsamlı bir olay yönetimi sunmaktadır, etkili başarım, izleme, otomatik uyarı, raporlama ve işletim sistemleri için grafikler, ara katman yazılımı (middleware) ve uygulamalar. HP OM kurumsal işletmelerde işletmen uçbirimlerinin(console) içindeki BT altyapısını birleştirmek için kullanılan bir yazılımdır. Ağ işletim çözümünde olayların birleştirilmesi ve son kullanıcının deneyimlerinin izlenmesi BT işletim çalışanlarına yönetilen ortamla ilgili eşsiz ve kapsamlı bilgi sağlamaktadır.

6.6.3.2 Teknik Özellikler

- Hizmet tabanlı izleme,
- Web tarayıcı üzerinden ileti gönderme yeteneği,
- Üçüncü parça için akıllı ve uyumlu ek yazılım bulunmaktadır(Oracle, SAP vd.),
- Üçüncü ürün için akıllı yazılım bulunmaktadır (Oracle, Sybase, SAP, vd.).
- HP raporlama yazılım ile bütünleşmesi,
- NNM, SM ve uCMDB ile bütünleşmek. (Bkz. Şekil 6.3)



Şekil 6.3 HP İşlemler Yönetimi (HP Operations Manager (OM)).

6.6.4 HP Hizmet Yöneticisi (HP Service Manager) (Günaydın, 2011).

6.6.4.1 İşlevsel Özellikler

HP Hizmet Yöneticisi (SM), hizmet düzeyini iyileştirmek, kaynakları dengelemek, maliyetleri denetlemek ve kurum çekincelerini azaltmasını sağlayan kapsamlı ve bütünleşik BT hizmet yönetimi yazılım ekibidir. Bir birimdeki kayıtlarla (records) diğer birimdeki tutanaklara bağlanabilir. Olay biletleri sorun (problem) kaydına bağlanabilir ve sorun tutanaklarda değişiklik isteklerine bağlanabilir. Bütün kayıtlar CI kayıtlarına ve SLA'lere bağlanabilir. Bütün tutanaklar arasında bağlantılar vardır ve kullanıcı bu bağlantılara tıklayarak tutanaklar arasında gezebilir.

Bütün HP SM birimleri tam olarak ITIL koşulları ile uyumlu (tanımlar ve işlemler). ITIL'ın 3. Sürümünün süreç modeli tam olarak HP SM içinde gömülüdür. Kullanıcı ara yüzünün bütünleşik bir parçası olarak ve

Süreç motorun ayrılmaz bir parçası olarak, HP SM, ITIL'in 3. sürümünün beş sürecinden oluşur:

- Belgeleme sürecinden 0, 1 ve 2. düzeyler,
- Önceden tanımlanmış roller, örnek SLA ve SLOS ve anahtar başarım göstergeleri,
- İyileştirilmiş, standartlaştırılmış ekran düzeni ve bilgiler ile ilgili akllı göstergeler ile uyarma,
- Sihirbazları yükseltmek için Richer çapraz birim (Richer cross-module) işlevselliğini kullanmak,
- HP'nin en iyi uygulamalarını uygulamak, yüksetmek ve bakım yapmak için gelişmiş seçenekler.

6.6.4.2 Teknik Özellikler

Yardım Masası aşağıdaki alt birimleri içerir:

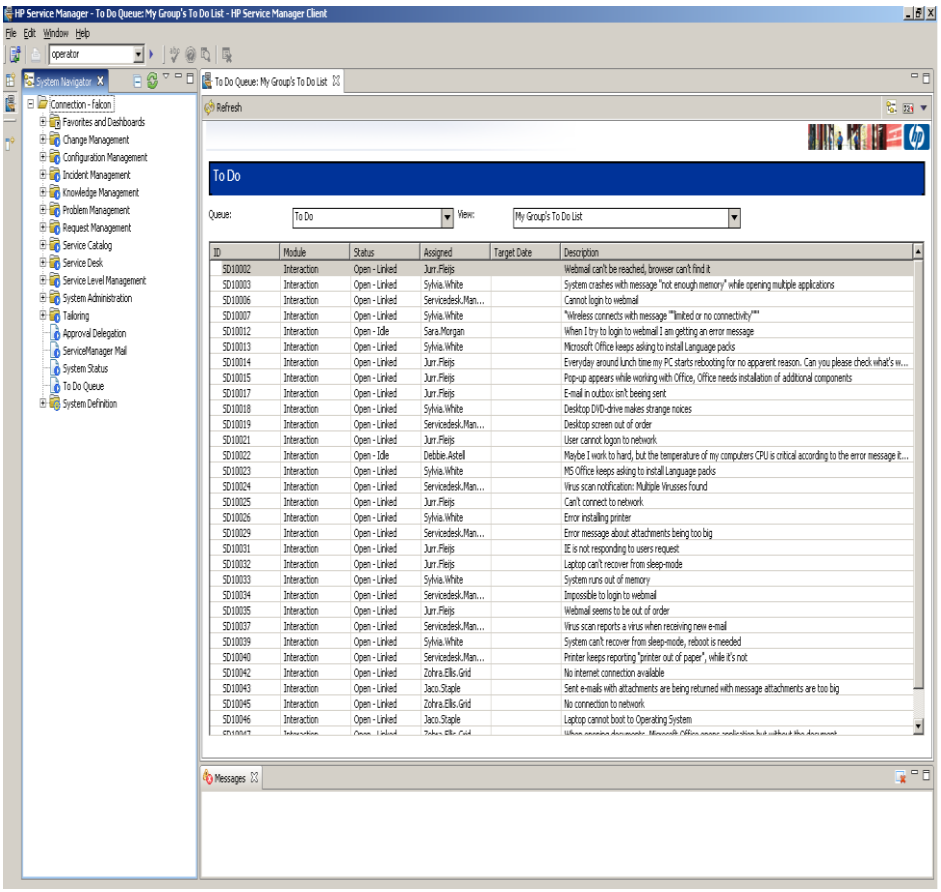
- Ön Hizmet Sunuşu: son kullanıcı sorgularının bilgisi, günlükler, güncellemeler, onaylamalar, istekleri kapatmak.
- Hizmet Masası (SD): SD aracılarının günlükleri, izler, görüntüleme, güncellemeler, atamalar ve ilgili kartlar,
- Olay Yönetimi: IM çözümleyicilerinin günlükleri, izler, görüntülemeler, güncellemeler, atamalar ve ilgili kartlar,
- Sorun Yönetimi: PM çözümle çözümleyicilerinin günlükleri, izler, görüntülemeler, güncellemeler, atamalar ve ilgili kartlar,
- Değişiklik Yönetimi: Değişikliklerin yaşam döngüsünü yönetmek.
- Yapılanış Yönetimi: CI'lerin yaşam döngüsünü yönetmek.

Yüksek süreç olgunluğu elde edildiği zaman alttaki adımlar uygulanır:

- Bilgi (Knowledge) Yönetimi: bilgi sorgulamasını sağlar ve bilgi belgelerini yönetir,
- Hizmet Düzeyi Yönetimi: SLO ve SLA'leri tanımlamayı ve yönetmeyi ve Süreç ile birimlerle bütünleşmeyi sağlar,
- İstek Yönetimi: İsteklerin yaşam döngüsünü yönetmek,

• Bakım Zamanlaması (Scheduled Maintenance): Dönemsel bakımları tanımlamayı ve programlamayı sağlar,

• Sözleşme Yönetimi: ürüne dayalı sözleşmeleri tanımlamayı ve yönetmeyi sağlar. (Bkz. Şekil 6.4)



Şekil 6.4 HP Hizmet Yöneticisi (HP Service Manager).

6.7 Kullanılan Diğer Araçlar

4S Bilgi Teknolojilerinin proje ekibi değişik yerlerde bulunması nedeni ile Dağıtık Uç Programlama kullanıldı. Dağıtık Uç Programlamanın en önemli ilkelerinden olan en düşük maliyetle ve en kolay biçimde proje gerçekleştirmektir. O yüzden yeni araç geliştirmek yerine hazır olan araçları kullanmak seçilmiştir ve DUP'in daha çok önem verdiği ekip çalışması ve ekibin yapılandırılmasına önem verilmiştir. Kullanılan iletişim araçları sonraki alt bölümlerde belirtilmektedir:

6.7.1Web tabanlı uygulamalar (Gmail)

4S Bilgi Teknolojileri çalışanları arasında ve çalışanları ile müşteri arasında temel olarak Gmail uygulaması kullandı. Gmail uygulaması hazır geliştirilmiş web tabanlı bir uygulama ve 100 kullanıcıya kadar ücretsiz olarak kullanılır. Gmail, çalışanlar arasında E-posta, Sesli toplantı ve görüntülü toplantı yolu ile haberleşme olanağını sağlamaktadır. Bunun yanı sıra müşteri ile E-Posta yolu ile haberleşmek için kullanılır.

Müşteri ile yapılacak görüntülü toplantılarda (Skype gibi) bazı hazır araçlar kullanılır yalnız projemizin kapsamında böyle bir toplantı gerçekleştirmediğimiz için bu araçları kullanmadık. (Bkz. Şekil 6.5)

Şekil 6.5 Gmail E-posta ve sohbet penceresi

6.7.2 Telefon

Uluslararası ağ bağlantılarında sorun olduğu zaman telefon kullanıyoruz. Müşteri ile yaptığımız bazı telefon görüşmelerimizde sorunlarını anlayıp ve uzaktan bağlanarak müşterinin sorununa çözüm buluyoruz.

6.7.3 Sanal Özel Ağ (Virtual Private Network (VPN))

Ağlara güvenli bir şekilde uzaktan erişimde kullanılan bir teknolojidir. Sanal bir ağ uzantısı yarattığından uzaktan bağlanan makine konuk gibi değil, ağa fiziksel olarak bağlıymış gibi görünür. Firmalar tarafından yaygın olarak kullanılan VPN, yöneticilerin, uzak ofislerin, bayi, acenta, satış temsilcilerinin güvenli bir şekilde özel ağlara bağlanmalarını sağlar. Uluslararası ağ gibi halka açık uzak iletişim altyapılarını kullanarak kullanıcıları veya uzak ofisleri düzeninin yerel bilgisayar ağına güvenli bir şekilde eriştirmeyi sağlamak için geliştirilmiş sanal bilgisayar ağı yapısıdır. Yapı genel olarak, uzak ofisler içi noktadan noktaya hatlar (lease line vb.) yerine standart bağlantılar üzerinden daha düşük sahip olma maliyetleri ile aynı hizmeti sağlar, tekil kullanıcılar için ise uzaktan (herhangi bir yerden) sanki fiziksel olarak ofis içerisindeymiş gibi çalışma olanağı sağlar. VPN ayni özel ağda bulunmayan, bir veya daha fazla ağ cihazı arasında güvenli bir şifreleme yöntemi kullanılarak güvenli veri aktarımı yapar. Güvenli şifreleme yönteminin kullanım amacı verinin özel yâda kamusal alandaki diğer ağ cihazlarından gizlenmesidir. Sanal özel ağ (VPN), komşu ağlar arasında gizli ve özel bir bilgi akışını sağlamaya yönelik kurulur. Paketler uluslararası ağ üzerinden gitse dahi, tünelleme ve kullanılan güvenlik yazılımları sayesinde ağ dinlense bile şifrelenmiş paketlere saldıran kişiye elde ettiği bilgiler hiçbir anlam ifade etmeyecektir. Buradan, büyük şirketlerin kendi aralarında uluslararası ağdan faydalanmaksızın, özel ağ kurmalarının çoğu yerde gereksiz olduğunu düşünebilirsiniz.(VPN, 2011) VPN yolu ile İstanbul veya Ankara 4S ofislerinden Irak'taki makineye bağlanıp yapılması gereken uyarlamaları yerinde çalışıyor gibi yapıyoruz ve gün sonunda müşteriyle bir yandan ve diğer yandan çalışma arkadaşlarımız ve yöneticilerimizle yapılanların rapor şeklinde gönderiyoruz.

6.7.4 Microsoft Office uygulamaları

Microsoft Office, 1989 yılında Microsoft tarafından tanıtılan, Microsoft Windows ve Mac OSX işletim sistemlerinde birbiriyle ilişkili masaüstü uygulamaları, sunucular ve hizmetler sunan bir ticari ofis yazılım paketidir. Yapılan araştırmalara göre, Haziran 2009 itibariyle işletmelerin %80'i Microsoft

Office'in herhangi bir sürümü kullanmaktadır (Office 2007'yi kullanan işletmelerin oranı %64 civarındadır) (MS Office, 2011).

Microsoft Office uygulamalarını (Word, Powerpoint ve Excel) günlük raporlarımızda ve müşteriye ziyaretlerimizde sunum yapmak için kullanmaktayız.

6.7.5 Gerektiği durumlarda müşteri ziyaretleri ve müşteri ile beraber çalışmak

Bazı durumlarda müşteri ile aynı ortamda çalışmak zorunda kalıyoruz ve bu durumda her yazılım için sadece bir kişi gönderiyoruz ve o yazılım ile ilgilenen diğer ekip üyeleri aralarında DUP yaklaşımını kullanarak çalışmaya devam ediyoruz. Proje sırasında ilk başta gereksinim çözümlemesi yapmak için bir ziyaret gerekir ve en sonun da da müşteri onay sınamasını yapmak için gitmek zorunludur. Bu iki durum dışında proje gerçekleştirmesi sırasında teknik destek için yazılımlarımızın ekiplerinden her yazılım için bir kişi gerektiğinde yerinde destek için gönderilir ve o ürünle ilgilenen diğer çalışanlar uzaktan bağlanarak gereken desteği sağlıyor.

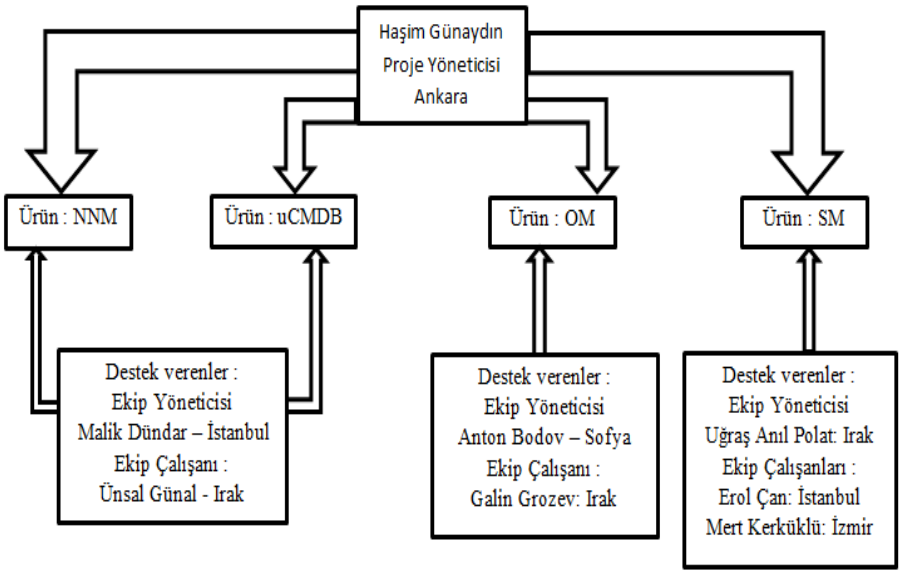
6.7.6 Raporlama, proje durumu ve harcamaları belirlemek için kullanılan araçlar

- Openair: Çalışma programını, proje durumunu ve hangi çalışanın hangi aşamada olduğunu ve ne iş yaptığını gösterir (Günaydın, 2011).
- Promaster: Projeler yurt dışında değişik yerlerde gerçekleştirildiği için bu program yolu ile projede yapılan harcamaları (uçak parası, otel ve diğer giderler) tutar ve bu harcamaları proje yöneticisi ve şirket yönetimine bildirir.

6.8 Proje Ekibi

4S Bilgi Teknolojilerince Irak'ta gerçekleştireceği projenin ekip üyeleri görüldüğü biçimde dağılmaktadır:

1. Proje Yönetici: Haşim Günaydın - Ankara.
2. NNM ve uCMDB destek ekibi: Malik Dündar – İstanbul ve Ünsal Günal.
3. OM destek ekibi: Anton Bodov – Sofya ve Galin Grozev – Irak.
4. SM destek ekibi: Uğraş Anıl Polat – Irak, Erol Çan – İstanbul ve Mert Kerküklü – İzmir. (Bkz. Şekil 6.6)



Şekil 6.6 4S Bilgi Teknolojilerinin Asiacell için gerçekleştireceği projenin üyeleri

Görüldüğü gibi her ürün için destek ekibi ayrı yerlerde bulunmaktadır ve ayrıca projenin tamamından sorumlu olan proje yöneticisinde hiç bir proje çalışanın olmadığı bir yerdedir.

Dağıtık Uç Programlara projemizde hem bir ürünün destek ekibi arasında aynı zamanda ayrı ayrı ürünleri destekleyen çalışma ekipleri ile proje yönetici arasında gerçekleştirilmektedir:

- Aynı üründe çalışan ekip üyeleri: Gün boyunca VPN, E-posta ve görüntülü toplantıları kullanarak çalışmaktadır. Günün sonunda ekip üyeleri raporlarını ekip yöneticisine göndermektedir ve ekip yöneticisi bunları birleştirip denetledikten sonra proje yöneticisine göndermektedir.

- Ekipler ile proje yöneticisi arasındaki bağlantı: Ekip yöneticiler günün sonunda raporlarını proje yöneticisinde gönderdikten sonra, proje yöneticisi 4S Bilgi Teknolojileri yönetimine ve aynı zamanda Asiacell BT bölümüne proje durumunu belirli aralıklarla belirtmektedir.

6.9 Projenin farklı aşamalarında ve katmanlarında DUP'nin uygulanması:

Projenin farklı aşama ve katmalarında DUP uygulanmaktadır. DUP, ister proje yöneticileri tarafından, ister teknik destek veren proje ekibi üyeleri veya müşteri ile kurulan bağlantıda farklı katman ve aşamalarda uygulanmaktadır. Bu katman ve aşamalar:

1. Proje yönetici ile proje ekibi arasında: Dağıtık Uç Programlama günlük veya haftalık raporlar ve toplantılar biçiminde hem proje ekibi ile proje yöneticisi arasında hem de proje yöneticisinin şirketin üst yöneticilerine verdiği raporlarda ve yapılan günlük veya haftalık toplantılar biçiminde bağlantıyı devam edilerek uygulanmaktadır.
2. Proje üyelerinin kendi aralarında: Dağıtık Uç Programlama proje üyelerini kendi aralarında (oyun planlamak, çift programlama, sürekli bütünleşme) kullanılmaktadır. Proje ekibi müşteri gerekesinim ve isteklerini daha iyi bir biçimde anlamanın yanı sıra, geliştirme ve uyarlama aşamalarında ortaya çıkan sorunları anlamak ve en kısa zamanda en uygun çözüme ulaşmak için yardımcı olmaktadır. Ayrıca genel olarak Çevik Süreçlerin sağladığı deneyim paylaşımı ve öğrenerek çalışmak ilkeleri bu aşamada çok açık bir biçimde görünmektedir. Projenin teknik destek ekibi aralarında yaptıkları teknik toplantılar, daha önce teknik deneyimi olmayan arkadaşlar açısından deneyim kazanmak ve yeni bilgiler öğrenmek açısından çok yararlı olmaktadır, bunun yanı sıra deneyimli proje üyelerinin deneyimlerini ve teknik bilgilerini paylaşmak yeni düşüncelerin ortaya çıkmasında yardımcı olmaktadır.
3. Proje ekibi ile müşteri arasında: Dağıtık Uç Programlama proje üyeleri ile müşteri arasında bağlantıyı sağlamak için (oyun planlamak, müşteri ile yerinde çalışmak, sürekli bütünleşme) kullanılmaktadır. Proje ekibi müşteri

gerekesinim ve isteklerini daha iyi bir biçimde anlamak için müşteri ile ister sanal veya gerektiğinde gerçek olarak beraber çalışmalıdır. Bunun yanı sıra, geliştirme ve uyarlama aşamalarında ortaya çıkan sorunları anlamak ve en kısa zamanda en uygun çözüme ulaşmak için ve müşterinin istekleri doğrultusunda küçük sürümlerin oluşturulmasını, bundan sonra yine müşterinin ortaya çıkan sürüm üzerinde düşüncesini belirtmesi için yardımcı olmaktadır. Bu aşamada iki ekibin arasında dil ve çalışma kültürü farkı nedeni ile bazı sorunlar ortaya çıktı ve bu sorunları çözmekte ben proje ekibine destek oldum. Bu sorunlardan mesai saati örnek olarak gösterilebilir, Irak'ta mesai saatleri daha kısadır ve çalışanlar fazla mesaiye kalmaya alışkın değiller.

4. İlgili Kişi (Contact Person): Bu görevi ben üstlendim ve amacı çalışma ekibi ile müşteri arasında bir bağlantı kurmaktır. Projenin son aşamalarında ve müşterinin gereksinim ve istekleri tam olarak belirginleştiği aşamada artık proje ekibinin teknik çalışanları Irak'ta bulunmak zorunda değil çünkü çıkan sorunlar E-Posta ve uzaktan bağlantı yolu ile çözülebilecek sorunlardır. O yüzden tek bir kişinin Irak'ta bulunması yeterli olmaktadır ve bu kişinin görevi ortaya çıkan sorunları veya yeni istekleri anlayarak bir rapor biçiminde İstanbul'daki şirket ekibine göndermektir. Yalnız bu görevi yapan kişinin çok sağlam bir altyapısı ve çok iyi bir iletişim yeteneği olmalıdır ve bu özelliklerde bir kişi bulmak biraz zordur. Bu görevi üstlenen kişi olarak teknik bilgilerimin yetersiz olması nedeni ile çok zorluk çektim; diğer yandan Arapçayı bilmem ve o coğrafyada olan çalışma kültürü ve düşünce tarzına alışık olmam bu görevi kolaylaştıran etkenler oldu.

6.10 Karşılaştığımız Sorunlar ve Kircherin yaşadığı sorunların çözülmesi:

Daha önce Kircher ve arkadaşlarının uygulamasını incelediğimiz için karşılaşacağımız çoğu sorunu daha önceden tahmin ediyorduk, yalnız projenin Irak'ta gerçekleştirilmesinden ötürü proje ekibinin öngöremediği bazı sorunlarla da karşılaştık. Karşılaştığımız sorunlar:

- Ayrı bir ülkede çalışan Türk ekip hem dil hem de çalışma biçimi açısından anlaşmakta zorlandı,
- Irak'ta yaşanan güvenlik sorunları,
- Elektrik kesintileri,
- Uluslararası ağ bağlantısında olan sorunlar,
- Arapça karakterlerin yazılımlar tarafından desteklenmemesi,
- Kircher ve arkadaşlarının karşılaştığı toplantıda iki kişiden fazla olma sorunu Skype kullanarak çözüldü,
- Kircher ve arkadaşları tarafından yaşanan saat sorunu. Proje ekibimiz, Irak ve Türkiye arasında saatin aynı ve Bulgaristan ile de yaklaşık olması sayesinde, bu konuda pek sorun yaşamadı,

6.11 DUP'nin projeye sağladığı katkılar

Dağıtık Uç Programlamayı uygulamak 4S Bilgi Teknolojileri tarafından ASIACELL için uygulanan projeye farklı alanlarda katkıda bulundu. Bu alanlar:

- Süre: Ocak 2011 tarihinde başlayan ve Eylül 2011'de bitirmesi ön görülen projemiz Temmuz 2011'in ilk haftasında bitmiş oldu. Böylece 9 ay süremesi beklenen proje 6 ayda bitmiş oldu ve öngörülen proje süresinde yaklaşık %33'lük bir azalma sağlandı. Geliştirici ve müşterinin, ister gerçek olarak isterse sanal ortamda, beraber çalışması projelerde genelde soruna dönüşen müşterinin gereksinimlerini anlama aşaması hızlı bir biçimde bitmiştir. Müşteri ile bire bir bağlantı kurmak, müşterinin tam olarak ne istediğini anlamakta ve aniden değişen isteklerini anlamayı ve en doğru biçimde uygulamakta yardımcı olmaktadır. Ayrıca proje ekibinin kendi aralarında düşüncelerini tartışmaları ve aralarında bire bir bağlantı kurmaları müşterinin isteklerini tam olarak anlamayı ve karşılanan sorunlara çözümlemekte ve birden çok çözüm bulmalarında büyük bir destek sağlamaktadır.
- Nitelik: Müşteri ile aynı ortamda, sanal veya gerçek biçimde, çalışmak müşterinin kullanılacağı yazılımların, yapılan uyarlama ve eklemeler sonucu isteğini tam olarak karşılamasını sağlamıştır. Müşterinin gereksinimlerine göre uyarlanan yazılımları, küçük sürümler biçiminde müşteriye sunmak ve

müşterinin onayını alarak devam etmek veya verdiği önerilere göre gereken değişiklikleri yapıp yeniden sunum yapmak. Projenin bütün aşamalarında geliştirici ile müşterinin yan yana veya etkin bir iletişim içerisinde olması, müşterinin tam olarak anlaşılmayan gereksinimlerini hem müşteri hem de geliştirici tarafından saptanması ve sunulacak ürünün müşterinin isteklerini tam olarak karşılaması için yardımcı olmaktadır. Ayrıca proje yöneticimizin düşüncelerine göre, projemizi daha önce uyguladığı projeler ile karşılaştırdığı zaman, müşterinin beklentilrini karşılamak açsından, DUP'nin daha başarılı olduğunu söyledi. Yalnız belli bir ölçüm yöntemi bulunamadığından bu başarı oranını soyut bir biçimde gösterememekteyiz.

- Maliyet: Dağıtık Uç Programlamayı uygulamak, projenin maliyetinde etkilemektedir. Uç Programlamada müşteri ile geliştiricinin ve ayrıca geliştirici ekip üyelerinin beraber çalışma koşulu ortadan kalkınca proje maliyetine katkıda bulunmaktadır. Örnek olarak, proje ekibinin tamamını Irak'a göndermek yerine, her ürün için sadece bir kişi ve bazı durumlarda tüm proje ekibinden sadece bir kişi Irak'a giderek, hem müşteri ile proje ekibi arasında olan bağlantıyı sağlamıştır. Bu biçimde uçak bileti, ekibin kalma ve yemek harcamaları ayrıca ekip üyelerine yurt dışı çalışmaları için ödenen günlük harcırahlarda 10 kişiye ödemektense sadece bir kişiye ödenmiştir. Projenin ilk başladrından 10 kişilik bir proje ekibi ile Irak'a gidilirken aşamalı bir biçimde bu sayı önce 5 ve son 2 haftada sadece 1 kişinin ilgili kişi (Contact Person) olarak orada bulunması yeterli olmuştur. Özellikler projenin son aşamasında sadece 1 kişinin orada ilgili kişi (contact person) olarak orada bulunması ve mesai saatleri içerisinde müşteri ile yapılan toplantılar ve müşteri görüşlerini bir rapor biçiminde proje ekibine göndermesi proje ekibinin tamamının Irak'ta olma gereksinimini ortadan kaldırmaktadır.

7. SONUÇ VE TARTIŞMA

Uluslararası bir şirket olan 4S Bilgi Teknolojileri ve projelerinin belli bir oranın yurt dışında olduğu için uygulamamız oldukça yaralı olup şirket içinde yeni yöntemeler kullanmak için bir dönüm noktası olduğunu söyleyebiliriz. Benim açımdan daha önce lisans eğitimim sırasında aldığım eğitimi sektörde uygulama olanağı sağladı ve yüksek lisans tezimin gerçekleştirilmesine yaptığı katkının yanı sıra, bir iş deneyimi edinmeme de aracı oldu. Çevik Süreçler açısından ise, çevik süreçlerin sağladığı proje ekibi arasında etkileşim ve müşteri ile olan ortak çalışmalarda çevik süreçlerin geleneksel yöntemlerden olan üstünlüğünü göstermek aynı zamanda

- Proje ekibi kendi aralarında ve müşteri ile sürekli bağlantı kurması çok yararlı oldu.
- Kendi aralarında bağlantı kurulması deneyim paylaşımı, proje ekibinin arasındaki ilişkilerin güçlenmesi ve birlikte çalışmanın getirdiği özenme ile ekip çalışmasının yarattığı eğlenceli ve samimi iş ortamını görmelerini sağladı.
- Müşteri ile beraber çalışmak ise proje ekibinin hayatın her alanında iletişim becerilerine olumlu yönde katkıda bulundu; yüz yüze olmadan bir insanın sorununu iyice anlayıp çözümlemeyi başarmayı sağladı.
- Günümüzün gelişen iletişim araçlarını ve onları etkin kullanımını göstermek için projemiz iyi bir örnektir. Değişik yerlerde olan proje üyelerimiz kimi zaman özel nedenlerden dolayı kendi evlerinden bir çalışmaya devam etmektedirler.
- Yazılım projelerinin genel sorunları maliyet, nitelik ve süre için iyi bir çözüm örneğidir:
    - Maliyet: Ücretsiz ve yaygın yazılımları kullanmak proje maliyetini düşürmektedir ve iletişim araçlarını etkin bir biçimde kullanarak proje ekibimizin üyeleri hepsinin Irak'a gitmesine gerek duymadan projeyi gerçekleştirmek ve bu yurt dışı seyahatlarını azaltmak proje maliyetini ciddi bir biçimde etkilemektedir. Projenin ilk başladrından 10 kişilik bir proje ekibi ile Irak'a gidilirken aşamalı bir biçimde bu sayı önce 5

ve son 2 haftada sadece 1 kişinin ilgili kişi (Contact Person) olarak orada bulunması yeterli olmuştur.

- Nitelik ve Süre: Gelişen iletişim araçlarını kullanarak süreyi azaltmakta ve sürekli müşteri ile geliştirici arasında iletişimin sağlandığı için müşterinin projenin her aşamasında gelişmelerden haberdar olması ve müşterinin gereksinimlerini tam olarak karşılayan nitelikli bir proje sunmaktadır. Ocak 2011 tarihinde başlayan ve Eylül 2011'de bitirmesi ön görülen projemiz Temmuz 2011'in ilk haftasında bitmiş oldu. Böylece 9 ay süremesi beklenen proje 6 ayda bitmiş oldu ve öngörülen proje süresinde yaklaşık %33'lük bir azalma sağlandı. Ayrıca proje yöneticimizin düşüncelerine göre, projemizi daha önce uyguladığı projeler ile karşılaştırdığı zaman, müşterinin beklentilrini karşılamak açsından, DUP'nin daha başarılı olduğunu söyledi. Yalnız belli bir ölçüm yöntemi bulunamadığından bu başarı oranını soyut bir biçimde gösterememekteyiz.

8. ÖNERİLER

Günümüzde iletişim araçlarının gelişmesi bu yöntemi kullanmak için çok iyi bir ortam yaratmaktadır. İletişim araçlarının gelişmesini en verimli biçimde kullanmak için DUP çok iyi bir örnek göstermektedir, aynı zamanda hem kendi iş yerinde bulunup değişik projelerle ilgilenirken uzaktan diğer çalışma arkadaşlarına destek de verebilir. Özellikle bazı özel nedenlerden dolayı seyahat engeli olan ekip çalışanları ve onların deneyiminden en iyi biçimde yaralanmak için proje sürecinde sağladığımız koşullar karşılıklı verimlilik sağlamıştır.

Daha önce Kircher ve arkadaşları tarafından yazılım geliştirmekte kullanılan DUP, çalışmamızda yazılım uyarlama firması olan 4S Bilgi Teknolojileri aynı yöntemi değişik bir amaç için kullanmaktadır.

Bunun nedeni ise, günümüzde yazılım sektöründe yazılım geliştirmekten daha çok, hazır yazılımlar satılmaktadır ve sıfırdan yazılım geliştirmek yerine, geliştirilen hazır bir yazılımda bazı değişiklikler yaparak kendi amaçlarımızı karşılayan bir duruma getirerek kullanmaktır.

Bundan yazılım geliştirme yöntemi olan DUP, yazılım uyarlamada kullanmak ve onun, için uygun olduğunu bu çalışmada göstermekteyiz, ayrıca hazır yazılımları kullanmak; süre, maliyet ve nitelik açısından projemize daha çok katkı sağlamaktadır.

DUP'nin ana düşüncesi sadece yazılım geliştirmekte değil, belli konularda olan uzmanları veya yöneticilerin görüşlerini gelişen iletişim araçlarını kullanarak değişik alanlarda olan projelerde de kullanılabilir.

KAYNAKLAR DİZİNİ

4S Bilgi Teknolojiler, http://www.4s.com.tr/Hakkimizda/Sayfalar/4STanitim.aspx, (Son erişim tarihi: 28.05.2011)

Abrahamsson, P. and Salo, O. , 2002, Ronkaine, J. , Warsta, J. , Agile Software Development Methods-Review and Analysis, VTT Publications 478.

Acar, Ö., 2009, Extreme Programming, Pusula Yayıncılık, İstanbul, Türkiye.

Aegis, http://aegis.sourceforge.net, (Son erişim tarihi:10.04.2009)

Altın, E., 2007, Değişimi Kucaklamak, Bilgi Üniversitesi Özgür Yazılım ve Açık Kaynak Günleri.

Arch, http://gnuarch.org, (Son erişim tarihi: 10.04.2009)

AT&T Laboratories Cambridge, Virtual Network Computing, 2001, http://www.uk.research.att.com/vnc/, (Son Erişim Tarihi: 10.04.2009).

Bilgen, S., 2008, EE647 Yazılım Mühendisliği ders notları, Elektrik ve Elektronik Mühendisliği bölümü, Orta Doğu Teknik Üniversitesi.

Bosley Group, 2001, Mind Mapper, mind mapping software, http://www.mindmapper.com, (Son Erişim Tarihi: 10.04.2009)

Chau, T. and Maurer F., 2004, Knowledge Sharing in Agile Software Teams, in Proceedings Symposium Logic vs. Approximation. Springer http://ebe.cpsc.ucalgary.ca/ebe/attach?page=Root.Root.PublicationList%2FChauMaurer2004.pdf, (Son erişim tarihi: 10.04.2009)

Cockburn, A. and Highsmith, J., 2001, Agile Software Development: The People factor, Computer, Page: 131-133

Cohn, M. and Schwaburn, K., 2003, The Need For Agile Software Management, Agile Times, Number: 1.

Cowan, G. S., 10 November 2001, What kinds of tasks are best performed alone? in Pair Programming, Portland Pattern Repository, http://www.c2.com/cgi/wiki?PairProgamming, , (Son Erişim Tarihi: 10.04.2009.)

CruiseControl, http://cruisecontrol.sourceforge.net , (Son erişim tarihi:10.04.2009)

Cunningham, W., 5 January 2001, Wiki Wiki Web, Portland Pattern Repository, http://www.c2.com/cgi/wiki? WikiWikiWeb, (Son Erişim Tarihi: 10.04.2009.)

CUSeeMe, http://www.fvc.com, (Son erişim tarihi: 10.04.2009)

CUseeMe Networks, 2002, Voice and Visual Communications Over the Internet, http://www.cuseeme.com. (Son erişim tarihi: 10.04.2009)

CVS (Concurrent Versions System), http://www.cvshome.org, (Son erişim tarihi: 27.05.2011)

Eclipse IDE, http://www.eclipse.org, (Son erişim tarihi: 10.04.2009)

KAYNAKLAR DİZİNİ (Devam)

GNU Project – Free Software Foundation, 22 July 2000 CVS – Concurrent Versions System, http://www.gnu.org/-software/cvs/ (Son erişim tarihi: 10.04.2009.)

Günaydın, H., 2011, 4S Bilgi Teknolojileri Asiacell ile gerçekleştirdiği projenin kapsam belgeleri. Süleymaniye, Irak. 112 s.

Highsmith, J., 2000, XP Programming, Cutter Consortium, MA, USA

Holler, R. , 2006, Agile Myths, Better Software, Page: 22-25

jUnit, http://www.junit.org, (Son erişim tarihi: 27.05.2011)

Karlesky, M. and Voord, M., 2008, Agile Project Management(,or Burning Your Gantt Chart), Embedded Systems Conference.

Kelter, U. and Monecke, M. and Schild, M., 2003, Do we need 'agile' Software Development Tools?, p.412-430 in: Aksit, M.; Mezini, M.; Unland, R.: Objects, Components, Architectures, Services, and Applications for a Networked World (International Conference NetObjectDays, NODe 2002; Erfurt, Germany, October 2002; Revised Papers); Springer LNCS 2591

Kircher, M., Jain,P., Corsaro, A. and Levine, D., 2001, Distributed Extreme Programming, XP2001 - eXtreme Programming and Flexible Processes in Software Engineering, Villasimius, Sardinia, Italy, May 21-23, 2001.

Kircher, M. and Levine, D., 2001, The XP of TAO – eXtreme Programming of Large, Open-source Frameworks, ExtremeProgramming Examined, Addison-Wesley, (Son Erişim Tarihi: 10.04.2009)

Larman, C., 2003, Agile and Iterative Development: A Manager's Guide, Addison-Wesley Professional, 368 page.

Larman, C. and Basili, V., 2003, Ierative and Incremental Development: A Brief History, IEEE, Page: 47-56

MASE, http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.MASE , (Son erişim tarihi: 10.04.2009)

Martin, R., 2008, Agile Principles, Patterns, and Practices in C#, Prentice Hall, 699 page.

Marjanovic, O., 2009, System Sciences, HICSS '09. 42nd Hawaii International Conference, Page:1-10.

Maurer, F. 2002, Supporting Distributed Extreme Programming, Proceedings Agile Universe/ XP Universe 2002, Spring, 2002 http://sern.ucalgary.ca/%7Emilos/papers/2002/Maurer2002.pdf , (Son erişim tarihi: 10.04.2009)

Maurer, F. and Martel, S., 2002, Process support for distributed extreme programming teams, ICSE 2002 Workshop on Global Software Development.

KAYNAKLAR DİZİNİ (Devam)

MILOS, http://wwwagr.informatik.uni-kl.de/~milos/ , (Son erişim tarihi:10.04.2009)

Monotone, http://www.venge.net/monotone/ , (Son erişim tarihi: 10.04.2009)

MS Netmeeting, http://www.microsoft.com/windows/netmeeting/default.asp , (Son erişim tarihi: 10.04.2009)

Newkirk, J., 2002, Introduction to Agile Process and XP Programming, ThoughtWorks Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on 25 may 2002, Page : 695-696, Florida, USA.

Rational ClearCase, 2001, http://www.rational.com/-products/clearcase/index.jsp, (Son Erişim Tarihi: 10.04.2009.)

Read, K. and Maurer, F., 2003, Issues in Scaling Agile Using an Architecture-Centric Approach: A Tool-Based Solution, Proceedings XP Agile Universe 2003, Springer 2003, p. 157-165. http://sern.ucalgary.ca/~milos/papers/2003/ReadMaurer_b.pdf , (Son erişim tarihi: 10.04.2009)

Schwaber K., 2004 Agile Project Management with Scrum, Microsoft Press.

Shore, J. and Warden, S., 2008, Art of Agile Development, O'Reilly, 409 s.

Schuemmer, T. and Schuemmer, J., 2001, Support for Distributed Remote Pair Programming, Extreme Programming Examined, Addison-Wesley.

Steinmetz, R. and Nahrstedt, K., 1996, Multimedia Computing, Communication & Applications, Prentice Hall, NJ.

Sommerville, I., 2006, Software Engineering, Addison-Wesley, ST ANDREWS, UK

Sangam, http://sangam.sourceforge.net, (Son erişim tarihi: 10.04.2009)

Stotts, D., Williams, L., Nagappan, N., Baheti, P., Jen, D. and Jackson, A.,2002, Virtual Teaming: Experiments and Experiences with Distributed Pair Programming.

Subversion, http://subversion.tigris.org, (Son erişim tarihi:10.04.2009)

VPN, http://tr.wikipedia.org/wiki/Virtual_Private_Network, (Son erişim tarihi: 27.05.2011)

Wiki technology, en.wikipedia.org/wiki/Main_Page, (Son erişim tarihi:10.04.2009)

Williams, L. and Cockburn, A., 2003, Agile Software Development: It's about Feedback and Change, IEEE, Page: 39-43

Williams, L. A. and Kessler, R. R. , 2000, All I Really Need to Know about Pair Programming I Learned In Kindergarten, Communications of the ACM.

xPlanner, http://www.xplanner.org, (Son erişim tarihi: 10.04.2009)

XPWeb, http://xpweb.sourceforge.net, (Son erişim tarihi: 10.04.2009)

ÖZGEÇMİŞ

Mohammed Jamal Ahmed ABBASI, 1985 yılında Irak'ın Kerkük şehrinde doğdu. İlköğrenimini Kerkük'te, Arafa İlkokulu'nda, orta öğrenimini yine Kerkük'te, Almuhalleb Bin-Abisufra Ortaokul'unda, liseyi de Kerkük'te Kerkük Merkezi Erkek Lisesi'nde tamamladı.

2002 yılında Kerkük'te Kerkük Teknik Fakültesi'nde Yazılım Mühendisliği bölümünde lisans eğitimine başladı ve 2006 yılında başarıyla mezun oldu. 2006 yılında Türkiye'ye gelip 1 yıl Gazi Üniversitesinde TÖMER'de Türkiye Türkçesi eğitimi aldıktan sonra 2007 yılında Ege Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda Yüksek Lisans eğitimine başladı.