T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BİR ÇEVİK YAZILIM GELİŞTİRME SÜRECİNİN UYARLANMASI VE UYGULANMASI

Derya AKBAYIR

Yüksek Lisans Tezi

Tez Danışmanı
Prof. Dr. Fuat İNCE

İSTANBUL – 2010

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BİR ÇEVİK YAZILIM GELİŞTİRME SÜRECİNİN UYARLANMASI VE UYGULANMASI

YÜKSEK LİSANS TEZİ

Derya AKBAYIR

Tez Danışmanı
Prof. Dr. Fuat İNCE

İSTANBUL – 2010

Bu tez çalışması, Maltepe Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 28 / 05 / 2010 tarih ve 2010 / 07 sayılı kararıyla oluşturulan jüri tarafından ***oy birliğiyle Yüksek Lisansı Tezi*** olarak kabul edilmiştir, 03/06/2010.

JÜRİ

Prof. Dr. Fuat İNCE
Danışman

Prof. Dr. Kemal KÖYMEN
Üye

Prof. Dr. Murat ESİN
Üye

ÖZET

Yüksek Lisans Tezi, Bir Çevik Yazılım Geliştirme Sürecinin Uyarlanması ve Uygulanması, T.C. Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Çevik, dünyada yazılım süreçlerini daha esnek ve güçlü kılmak için kullanılan aynı zamanda yazılım süreçlerini de kısaltan kavramsal bir yazılım geliştirme metodolojisi yaklaşımıdır. 1990'lı yıllarda "çevik" adı altında toplanan yeni yazılım geliştirme yaklaşımları, kısa süreli ve değişken projeler için yeni metodolojiler önermiştir. Çalışan yazılımı kapsamlı belgelemeye tercih eden, plan izleme yerine değişime açık ve müşteriyle sıkı bir etkileşim içinde olan bu metodolojiler sayesinde klasik yaklaşımlarla başarılması riskli projeler, kabul edilebilir ölçüde başarılı olabilmiştir.

Bu tezin konusunu, hem çevik yaklaşımlarla yazılım geliştirme hem de kullanılması gereken internet tabanlı programlama teknolojilerine aşina olmayan bir yazılım ekibi tarafından üniversite otomasyon sisteminin geliştirilmesine yönelik proje süreci oluşturmuştur.

Bu tezin amacı, gerçekleştirilen projede tek bir çevik metodolojiye bağlı kalınmadan ekibe ve müşteri gereksinimlerine özgün pratiklerin seçilerek uyarlanmasıdır. Bu uyarlama ile tercih edilen pratikler, projenin büyüklüğü, gereksinimlerin karmaşıklığı, yazılımın kapsamı ve ekibin yapısı değerlendirilerek belirlenmiştir. Ayrıca çevik metodolojiler içinden seçilen pratiklerin proje sürecine nasıl uyarlandığı ve uygulandığı, uygulama aşamasında hangi sıkıntılarla karşılaşıldığı ve bu sıkıntıları çözümleme yolları anlatılmıştır. Bu pratikleri uygulayacaklar için önerilerde bulunulmuştur.

Bu tez 2010 yılında tamamlanmıştır ve 104 sayfadan oluşmaktadır.

Anahtar Kelimeler: Çevik, Çevik Yazılım Geliştirme, XP, Scrum, Uyarlanma, Team Foundation Server.

ABSTRACT

Master Thesis, Implementation and Tailoring of An Agile Software Development Process. T.C. Maltepe University, Institute of Natural Sciences, Department of Computer Engineering.

During the 1990s, a new approach to software development, called “agile”, brought forth new methodologies for projects of relatively short duration and changing requirements. Software projects which start with uncertain requirements are considered risky under classical approaches, but they became fairly well manageable and successful by using these newer methods. Agile methods emphasize working software to detailed documentation, remaining open to change and advocating strong interaction with customers.

This study reports the results of a project development effort deriving its development process from different agile methods adapted for the circumstances. The project aimed to develop a university automation system by a software team which was initially unfamiliar with the agile software development approach and internet based programming technologies.

The aim of this thesis is that the original practices are tailoring by choosing related to the requirements of the software team and customer without sticking to only one agile methodology in the applied project. The preferred practices with this tailoring have been determined the size of the project, the complexity of the requirements, the scope of the software and the structure of the software team by evaluating them. Furthermore, this thesis describes the approach and the evaluation on how to tailor and implement the chosen practices from the different agile methods the reasoning behind the selection, which challenges were addressed during the implementation, and the ways to solve these challenges. We comment on the experience gained and make recommendations on the implementation of these practices.

This thesis has been completed in 2010 and consists of 104 pages.

Keywords: Agile, Agile Software Development, Extreme Programming, Scrum, Tailoring, Team Foundation Server.

TEŞEKKÜR

Seçtiğim tez konusu üzerinde çalışmama olanak sağlayan Maltepe Üniversitesi rektörü sayın Prof. Dr. Kemal KÖYMEN'e, rektör yardımcısı ve Bilgisayar Mühendisliği bölüm başkanı sayın Prof. Dr. Murat TAYLI' ya ve Fen Bilimleri Enstitüsü Müdürü sayın Prof. Dr. İlhami YAVUZ'a, tez süreci boyunca destek ve yardımlarını esirgemeyen değerli bilgilerinden istifade ettiğim danışman hocam Prof. Dr. Fuat İNCE' ye, çalışmalarım sırasında gerekli kaynakların sağlanmasına yardımcı olan sayın Kadir ÇAMOĞLU'na, süreçler konusunda yardım ve desteklerini benden esirgemeyen sayın Öğr. Gör. Fatih YÜCALAR ve sayın Yrd. Doç. Dr. T.Tugay BİLGİN'e, tez sürecinde bana gösterdiği olağanüstü anlayış ve destek için çok değerli eşim Recep AKBAYIR' a, manevi desteğini benden hiçbir zaman esirgemeyen çok değerli aileme ve çalışmalarım sırasında emeği geçen herkese teşekkürlerimi sunarım.

İÇİNDEKİLER

KISALTMALAR

Kısaltma	İngilizcesi	Türkçesi
ASP.NET	Active Server Pages Framework	Aktif Sunucu Sayfaları Çerçevesi
CMMI	Capability Maturity Model Integration	Yetenek Olgunluk Model Entegrasyonu
DSDM	Dynamic System Development Methodology	Dinamik Sistem Geliştirme Metodu
FDD	Feature Driven Development	Özellik Tabanlı Geliştirme
http	HyperText Transfer Protocol	Zengin Metin Aktarma Protokolü
IIS	Internet Informatin Services	İnternet Bilgi Servisleri
MSF	Microsoft Solution Framework	Özel ürün adı: Yazılım geliştirme aracı
RUP	Rational Unified Process	Özel ürün adı: Yazılım geliştirme aracı
SDLC	Software Development Life Cycle	Yazılım Geliştirme Yaşam Döngüsü
SOA	Service Oriented Architecture	Servis Odaklı Mimari
SQL	Structured Query Language	Yapısal Sorgu Dili
TDD	Test Driven Development	Test Güdümlü Geliştirme
TFS	Team Foundation Server	Özel ürün adı: Yazılım geliştirme aracı
VSTS	Visual Studio Team Suite	Özel ürün adı: Yazılım geliştirme aracı
WSS	Windows Sharepoint Services	Özel ürün adı: Paylaşım siteleri geliştirme platformu

XML	eXtensible Markup Language	Genişletilebilir Bağlantılı Metin Dili
XP	Extreme Programming	Uç Programlama

ŞEKİLLER

TABLOLAR

1. ÇEVİK METOTLARA GİRİŞ

Yazılım geliştirme süreci sıkıntılı ve uzun süren bir dönemdir. Yazılım projeleri genelde yönetimsel eksikliklerden dolayı ancak kısmi başarı ve memnuniyet ile tamamlanabilmektedir. Yazılım sektöründe, yazılım sürümlerinin zamanında ortaya çıkarılamaması, değişiklik isterlerine çabuk cevap verilememesi, yazılım hatalarının geç fark edilmesi ve zaman içerisinde gelen isteklere göre sistemin kendi yapısını geliştirememesi gibi çeşitli sorunlar ortaya çıkmıştır. Bu sorunların aşılmasına yönelik yapılan çalışmalar sonucu, 1990'lı yılların sonlarına doğru "çevik" olarak isimlendirilen metotlar geliştirilmiştir. [1]

Çevik kelimesi, hızlı düşünmek olarak ifade edilebilir. Geleneksel yazılım geliştirme süreçleri, projelerde değişen gereksinimleri karşılamak ve beklenen kalite faktörlerine erişmek açısından çok katı ve ağırdır. Çevik yazılım geliştirmede ise, süreçlerin katı kurallarla yönetilmesi yerine amaca yönelik verimli ve etkin pratiklerin yapılması esastır.

Çevik metotlar, piyasaya çabuk ürün çıkarabilme, değişen isteklere hızla yanıt verme ve en kısa sürede bir yazılım ürününü müşteri hizmetine sunmayı amaçlamaktadırlar. [2] Çevik metotlar, verimliliği yüksek, esnek, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadır. Bu metotlar, kendi içerisinde özü aynı fakat pratikleri farklılaşan çeşitli metodolojilere ayrılmaktadır.

Çevik süreçte yazılımcılar ile iş mantığı konusunda uzman kişilerin belgelerle iletişim kurması yerine yüz yüze etkileşimli iletişim kurmaları tercih edilmektedir. Bunun sonucunda aradaki gereksiz zaman kaybı ortadan kaldırılmakta ve müşterinin daha sık aralıklarla kullanabileceği ya da test edebileceği ürünlerin ortaya çıkması mümkün olmaktadır. Bu sürece dâhil olan ekibin uzmanlığı çok önemlidir. Sürecin en önemli kaynağı insan, yani uygulamayı geliştiren ekiptir. Bu ekibin, uzmanlık

düzeyi ne kadar iyi seviyede ise süreç o kadar başarılı olacaktır. Ekibin kendi kendini yönetebilmesi ve ortaya çıkabilecek bir sorunda alternatif yollar önerebilmesi beklenen davranışlardandır.

En önemli amaçlardan bir tanesi, yazılımın küçük geliştirmelerle yürütülmesi ve böylece ortaya çıkabilecek risklerin azaltılmasıdır. Çevik süreç için, deneyimli insanlarla çalışmak, çalışanları birbirlerine yakın tutmak ve iletişimi kolay bir ortam sağlamak, müşteri ya da kullanıcılara yakın olmak, geri bildirimleri hızlı değerlendirmek ve çalışmaların hızlı şekilde belgelenmesini sağlamak gerekir.

1.1. Çevik Yazılım Geliştirme Metodolojisi

Çevik yazılım geliştirme metodolojisi, çevik tanımı, manifestosu, prensipleri, getirileri, hangi durumlarda kullanılabileceği, bu sürecin farkı ve avantajları ve bu metodu kullanan takımların özellikleri gibi alt başlıklarla bu bölümde incelenmiştir.

1.1.1. Çevik Nedir?

"Çevik", dünyada yazılım süreçlerini daha esnek ve güçlü kılmak için kullanılan aynı zamanda yazılım süreçlerini de kısaltan kavramsal bir yazılım geliştirme metodolojisidir. [3] Bu metodolojide projenin ölçeği ne olursa olsun, proje küçük iterasyonlara ayrılır ve her iterasyon başlı başına bir proje gibi ele alınarak geliştirilir. Her iterasyonun sonunda da proje ekibi tarafından müşteriye, projenin ne kadarının gerçekleştirildiğine dair bilgi verilir. "Çevik" ile her bir iterasyonun 2-4 hafta kadar sürmesi planlanmaktadır. Her iterasyonun kendi içerisinde çalışan bir sistem olması sonucu müşteriye sürekli çalışan bir yazılım teslim edilerek, müşteri memnuniyetinin artması sağlanmaktadır. "Çevik" in hızı proje ekibinde çalışan tüm ekip üyelerinin sürekli iletişim halinde olmasından kaynaklanmaktadır. Ayrıca projenin küçük parçalardan oluşması da geriye dönük hataların düzeltilmesini kolaylaştırmaktadır.

Genel hatlarıyla çevik metotlar, verimliliği yüksek, esnek, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadır.

1.1.2. Çevik Metodların Manifestosu

2001 yılında Kent Beck ve 16 arkadaşı tarafından aşağıda yer alan çevik yazılım geliştirme manifestosu oluşturulmuştur.

Tablo 1.1 Çevik yazılım geliştirme manifestosu

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

Türkçesi:

1. Kişiler ve iletişim süreç ve araçlardan önce gelir.
2. Çalışır durumda olan program kapsamlı dokümantasyondan daha önceliklidir.
3. Müşteri ile beraber çalışmak sözleşme görüşmelerinden daha önceliklidir.

4. Değişikliklere ayak uydurmak bir planı takip etmekten daha önemlidir.

Tablo 1.1'in sağ bölümde yer alanlar (altı çizili olmayanlar) değer taşımakla beraber, sol bölümde altı çizili olan değerler bizim için daha kıymetlidir. Bu manifesto ile çevik sürecin bir değer sistemine sahip olması ve geleneksel yazılım metotlarından elde edilen deneyimler doğrultusunda daha pratik ve çevik bir yapıda olması gerektiği dile getirilmiştir.

1.1.3. Çevik Metotların Prensipleri

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

1. *En önemli öncelik erken ve sürekli olarak kullanılabilir programlar oluşturarak, müşteriyi tatmin etmektir.*

Bu prensip ile aslında programın müşteri tarafından talep edildiğini ve müşteriyi sadece istekleri doğrultusunda geliştirilmiş ve çalışır durumda olan bir programın tatmin edebileceği dile getirilmektedir. Peki, müşteri nasıl tatmin edilebilir? Bunun için proje ekibinin proje başlangıcından itibaren ve sürekli olarak çalışır durumda olan programlar oluşturarak, müşteriye sunması ve görüşlerini alması gerekmektedir. Bu sayede müşteri inceleyebileceği ve çalışır durumda olan bir program aracılığıyla gereksinimlerinin ne oranda gerçekleştirimle (kodlama) örtüştüğünü kontrol edebilir. Gereksinimlerin değişmesi ya da müşterinin istekleri dışında bir yazılım yapılması durumunda müşteri yazılım sürecine müdahale edebilir ve gerekli değişiklikleri talep edebilir. Bu durum proje sonunda müşteri gereksinimleri ile yüksek oranda örtüşen bir programın oluşmasını sağlar.

2. *Yazılımın ilerleyen dönemlerinde gelse bile talep edilen değişiklikler hoş karşılanmalıdır. Çevik süreçler, değişiklikleri müşterinin rekabetteki avantajını korumak ve sağlamak için kullanılırlar.*

Geleneksel yazılım metotlarının uygulandığı projelerde mimarinin ve planlamanın büyük bir bölümü gerçekleştirim öncesi oluşturulur. Geleneksel yazılım, hazırlanan planlardan sapmadan gerçekleştirilmeye çalışılır. Bu, müşteri tarafından talep edilen değişikliklerin göz ardı edilmesi anlamına gelmektedir. Böyle bir sürecin sonunda müşteri isteklerini tatmin etmeyen ve müşterinin piyasadaki rekabet kabiliyetini sınırlayan programlar oluşur. Bunu engellemek için her zaman müşteriden gelen değişiklik taleplerinin gerçekleştirim esnasında dikkate alınması gerekmektedir. Değişiklik ne zaman gelirse gelsin, gerçekleştirim bu değişiklik doğrultusunda adapte edilebilmelidir.

3. *Kısa zaman aralıkları tercih edilerek iki haftadan iki aya kadar çalışır yazılım ortaya koyulmalıdır.*

Çevik süreçlerde programlar hem iterasyonlu hem de artırımlı olarak geliştirilir. Bir iterasyon, yazılım için gerekli tüm safhaları ihtiva eden belirli bir zaman biriminde gerçekleştiriminin gerçekleştirildiği süreçtir. Program ayrıca artırımlı oluşturulur, çünkü ekip üyeleri her iterasyonda müşterinin seçtiği gereksinimlere yoğunlaşarak programı yavaş yavaş oluşturur. Her iterasyon sonunda program müşteriye sunularak, görüşleri alınır.

4. *Müşteri ve yazılımcılar proje süresince beraber çalışmalıdır.*

Çevik projelerde müşteri ile proje ekibinin beraber çalışması doğaldır. Programdan olan beklentileri en iyi müşteri bilebileceği için, sürekli müşteriden geri dönüm alınması gerekmektedir. Bunun en kolay yolu müşteri ile proje ekibinin beraber çalışmasıdır.

Proje ekibi, özellikle yazılımcılar müşteri tarafından dile getirilen gereksinimlerin (requirement) fizibilitesini (yapılabilirlik) araştırırken, her gereksinim için kodlama zamanını tahmin ederler. Bu doğrultuda müşteri; kendi tanımlamış olduğu gereksinimlere bir öncelik sırası vererek, hangi gereksinimlerin öncelikli olarak

gerçekleştirilmesi gerektiğini belirler. Yazılımcılar bu konularda müşteriye yardımcı olarak, gereksinimlerin daha somut hale getirilmelerini sağlarlar. Bunlar genellikle birkaç cümleden oluşan kullanıcı hikayelerine (user story) dönüştürülür. Müşteri ve yazılımcılar arasındaki sürekli diyalog yanlış anlaşılmaları ortadan kaldırır. Bu yüzden müşterinin her gün yazılımcının erişebileceği bir mesafede olması gerekmektedir.

Geleneksel yazılım metotlarında bunun böyle olmadığı görülmektedir. Projenin ilerleyen safhalarında müşteri tarafından talep edilen değişiklikler olumlu karşılanmaz, çünkü bu proje başlangıcında yapılan anlaşmalara ters düşmektedir. Başlangıçta ne yapılmasına karar verildiyse, yazılımcılar rahatsız edilmeden mevcut gereksinimlerin gerçekleşmesine odaklanabilmelidirler. Böyle bir metotla oluşan programın, müşteri gereksinimlerini ne ölçüde tatmin edebileceği ortadadır.

5. Projelerin motivasyonu yüksek bireyler tarafından yapılmasını sağla, onlara ihtiyaç duydukları ortamı ve desteği ver ve işi bitirebileceklerine inan.

Çevik projeler bireylerden oluşur ve her yazılımcı kendi bireysel karakteri ile takımın bir parçasıdır. Ekip elemanlarının değişik karakterde olmaları doğaldır. Yazılımcıların, onlara duyulan güvenin hissettirilmesiyle motivasyonları arttırılır. Onların sorumluluk almaları sağlanarak, özgüvenlerinin pekişmesi sağlanır.

Yazılımcılar birbirlerine yardım ederler. Onlar arasında kıdem farkı yoktur. Bilen bilmeyene öğreterek, kısa bir zamanda aynı teknik seviyeye gelmeleri sağlanmış olur.

6. Bilgi alışverişinde en verimli ve efektif yöntem takım içinde yüz yüze konuşmaktır.

Çevik projelerde bilgi alışverişi yüz yüze gerçekleşir, çünkü bilginin transfer esnasında en az hasara uğradığı yöntemlerinden birisi budur. Yazılımcılar arasındaki kişisel konuşmalar güveni artırır ve yanlış anlaşılmaları ortadan kaldırır. Sorunlar

hemen açıklığa kavuşturulabilir.

7. Çalışır durumda olan program ilerlemenin ana göstergesidir.

Müşteriye kısa aralıklarla çalışır durumda olan lakin henüz tamamlanmamış bir program sunulduğunda, müşteri mevcut programın kendi gereksinimlerini tatmin edecek durumda olup olmadığını kontrol edebilir. Bu yüzden müşteri çalışır durumda olan programı kullandıktan sonra gereksinimlerine uygunluğunu tespit edebilir.

8. Çevik süreçler etkili yazılım yöntemlerini destekler. Müşteri, yazılımcılar ve kullanıcılar sabit bir tempoda beraber çalışabilmelidirler.

Çevik projelerde sabit bir çalışma temposunun oluşturulması büyük önem taşımaktadır. Yazılımcılar arasında görev eşit bir şekilde paylaştırılır. Fazla mesai yapılması hoş karşılanmaz. Bu ayrıca çalışanların motivasyonunu olumlu etkiler. Proje ilerledikçe iş azalır. Sonradan yazılımcıları kötü sürprizler beklemez, çünkü ortada iyi test edilmiş ve çalışır durumda olan bir program vardır.

9. Teknik mükemmelliğe devamlı özen gösterilir ve iyi tasarım çevikliği kuvvetlendirir.

Çevik projelerde kalite beklentisi yüksektir. Yazılım temin edilen en iyi araç ve yetenekli yazılımcılarla gerçekleştirilir. Bu süreçte programın tasarımı sürekli optimize edilir. Yazılımcılar yeniden yapılandırma (refactoring) esnasında buldukları tasarım hatalarını hemen giderirler.

10. Sadelik (basitlik) esastır.

Bir programı mümkün olan en basit tarzda gerçekleştirmek, programın karmaşıklığını azaltır. Karmaşıklığı düşük olan bir programın bakımı ve geliştirilmesi kolaydır. Yazılımcılar, yazılım esnasında sadece kendilerinden o anda

olan beklentiyi tatmin edecek kadar kod yazarlar. Bu, gereksiz kodun oluşmasını engeller ve test edilebilir bir yapının ortaya çıkmasını sağlar.

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

Çevik takımlar kendi başlarına organize olabilme özelliğine sahiptir. Böyle takımlar görevleri kendi aralarında eşitçe paylaşırlar. Takımlar ve bireyler arasındaki görüşmeler ve bilgi alışverişi sonucunda mimari ve tasarım gözden geçirilir ve optimize edilir. Ayrıca bireyler arası yapılan konuşmalar müşteri tarafından dile getirilen gereksinimlerin açıklığa kavuşturulmalarını ve daha iyi anlaşılmalarını sağlarlar.

12. Belirli zaman dilimlerinde takım nasıl daha etkin olabileceği konusunda kendini sorgular ve edindiği bilgiler doğrultusunda çalışma tarzını adapte eder.

Çevik takımlarda bilgi alışverişi ve devamlı öğrenme çok önemlidir. Belirli zaman aralıklarıyla takım çalışanları bir araya gelerek, fikir alışverişinde bulunurlar ve çalışma yöntemlerini sorgularlar. Edinilen tecrübeler doğrultusunda yazılım sürecinde adaptasyonlar gerekebilir. Nihai amaç en verimli şekilde çalışabilmek ve müşteri tarafından kabul görmüş bir program oluşturabilmektir. [5, 6, 7, 8]

1.1.4. Çevik Metotların Getirileri Nelerdir?

Düşük Risk:

Tekrarlanan yazılım geliştirme metotları, proje risklerini azaltıp başarıyı arttırmakta ve hata oranlarını düşürerek verimliliği yükseltmektedir. Bunun arkasında yatan en temel etken, daha projenin başlarında geliştirilen program parçacıkları sayesinde, proje ekibinin yetkinliklerinin ve projede kullanılan her türlü araçların önceden denenerek eksiklerinin görülebilmesidir. Ayrıca, iterasyonlu geliştirme süreci

içerisinde proje hızlı olarak şekillenmekte ve proje başlangıcında fark edilemeyen riskler ciddi sorunlara yol açmadan önce, görülebilir hale gelmektedir. [9]

Değişime Açık ve Yatkın Olma:

Yazılım projelerinde değişiklik kaçınılmazdır. Orta çaplı projeler dahi proje süresince başlangıçlarına göre %30 oranlarında değişeme uğramaktadır. Bu nedenle, değişim yazılım projelerinin doğasıdır ve bu gerçeklik çevik metodolojilerin de üzerinde önemle durduğu bir etkendir. [9]

Çevik metodolojiler değişime karşı gelmek yerine değişimi müşteri avantajına dönüştürmeye yönelik olarak çalışırlar. Çevik metodolojiler, önerdiği iterasyonlu yazılım üretimi ve her adımdaki güçlü bilgi alışverişiyle, değişim gereksinimlerinin mümkün olduğunca erken fark edilmesini ve projenin değişime hızlı bir şekilde adapte olabilmesini sağlarlar.

Karmaşıklığın Yönetimi:

Yazılım projelerinin hacimleri büyüdükçe karmaşıklıkları artar ve buna bağlı olarak hata oranları da artar. Çevik yöntemler ise projeleri, daha kolay yönetilebilir küçük parçalara bölerek ele alırlar. Böylece, projelerin büyüklüğü ne olursa olsun, küçük parçalara ayrılarak ele alınan projeler için karmaşıklık en düşük seviyeye indirilir. [9]

Sürekli Yazılım Teslimi:

Çevik metodolojilerde her iterasyon sonunda çalışır bir programcık meydana getirilmektedir. Projenin başlarından itibaren devam ederek büyüyen bu parçacıklar, müşterilere elle tutulur aşamalar olarak sunulmakta ve bu da müşteri memnuniyetini arttırmaktadır. Yapılan araştırmalar, müşterilerin tamamlanmış ama çalışmayan programlar yerine, çalışır durumda ama tamamlanmamış programları tercih ettiklerini göstermiştir. [9]

Yüksek Kalite:

Tekrarlanan yazılım geliştirme metotlarının bünyesinde, test odaklı yazılım geliştirme mantığı bulunmaktadır. Proje başından başlayarak tüm test süreçlerinin yazılım geliştirme süreci içerisinde beraber yürütülmesi sayesinde, hatalar büyümeden fark edilerek hızlı bir şekilde düzeltilebilmektedir.

Proje hacimleri büyüdükçe hata oranları da artmaktadır. Ancak çevik metodolojiler, projeleri parçalara bölerek ve her bir parçayı kendi gelişimi içerisinde de test ederek hata oranlarının düşürülmesini sağlamaktadır. [9]

Müşteri ihtiyaçlarına daha iyi cevap veren çözümler:

İterasyonlar arasında gerçekleştirilen fikir alış verişleri, çevik yöntemlerin değişikliğe olan yatkınlığı ve müşteri odaklı yaklaşımları sayesinde, projeler müşteriler ile birlikte değişerek gelişmekte ve proje sonunda da müşteri ihtiyacını en iyi derecede karşılayabilecek programlar ortaya çıkmaktadır.

Müşterilerin çoğunlukla proje başlangıcında tam olarak ne istediklerine dair net birer fikirleri yoktur ve istekler projenin gelişim süreci ile birlikte şekillenmektedir. Bu gerçeklik karşısında, çevik yöntemler müşteri odaklı ve esnek yapısıyla, müşteri memnuniyetini en üst seviyede tutmayı başarabilmektedirler. [10]

1.1.5. Çevik Metotlar Hangi Durumlarda Kullanılabilir?

Çevik metotların kullanılmasının en uygun olduğu durumlar şunlardır:

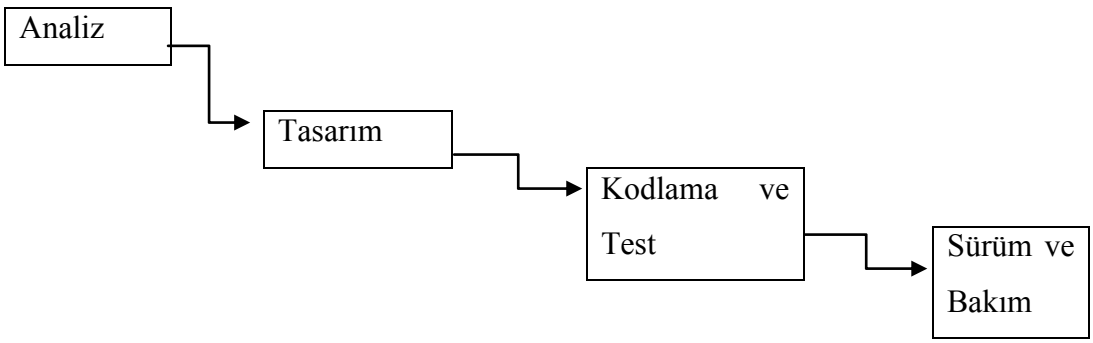
- Projenin yazılım evresinde müşteriden gelebilecek talep değişikliklerinin tahmin edilemez olması.
- Projenin parçalarının önce tasarlanıp ardından hemen geliştirilmesinin gerekmesi ve önceden ne yapılacağını, ayrıntılı yol haritasını ve tasarımını tahmin etmenin çok güç olması.

- Analiz, tasarım ve test etme süreçlerinin ne kadar zaman alacağının önceden bilinememesi.
- Yazılım ekibinin birlikte çalışmak, hiyerarşiye önem vermemek, sağlam iletişim kurmak gibi özelliklere sahip olması.

Elbette bazı durumlarda çevik programlamadan vazgeçilmesi gerekebilir. Örneğin çok kişinin dâhil olduğu projelerde çevik metotlar ile proje geliştirmek mümkün olmayabilir ya da aynı yerde bulunmayan takım arkadaşları ve hiyerarşik yapının her an hâkim olduğu şirket ortamlarında klasik yöntemler daha uygun olabilir. [3,11]

1.1.6. Çevik Metodoloji Sürecinin Farkı ve Avantajları

Yazılım geliştirme süreci analiz, tasarım, kodlama, test, sürüm ve bakım gibi safhalardan oluşur. Geleneksel yazılım metotlarında bu safhalar çağlayan modelinde olduğu gibi doğrusal olarak işler. Her safha başlangıç noktasında bir önceki safhanın ürettiklerini bulur. Kendi bünyesindeki değişikler doğrultusunda teslim aldıklarını bir sonraki safhanın kullanabileceği şekilde dönüştürür.



Şekil 1.1 Geleneksel yazılım geliştirme sürecinin safhaları

Çağlayan modelinin özelliklerini şu şekilde sıralayabiliriz:

1. Çağlayanın her basamağında yer alan aktiviteler eksiksiz olarak yerine getirmeden bir sonraki basamağa geçilmez.

2. Her safhanın sonunda bir tamamlama dokümanı oluşturulur. Bu yüzden çağlayan modeli doküman güdümlüdür.

3. Kullanıcı katılımı başlangıç safhasında mümkündür. Kullanıcı gereksinimleri bu safhada tespit edilir ve ayrıntılandırılır. Daha sonra gelen tasarım ve kodlama safhalarında müşteri ve kullanıcılar ile diyaloga girilmez.

Bu modelin beraberinde getirdiği problemleri şu şekilde sıralayabiliriz:

1. Safhaların birbirinden kesin olarak ayrı tutulmaları gerçekçi değildir. Gerçek projelerde safhalar arasındaki bu sınırlar yok olabilir.
2. Teoride safhalar birbirlerini takip edeler. Gerçek projelerde bunun bazen mümkün olmadığını ve önceki safhalara geri dönülmek zorunda kalındığını görebiliriz.
3. Safhalar arası geri dönüm yetersizdir. Model değişikliğe açık değildir.
4. Müşteri gereksinimlerinin proje öncesi ayrıntılı olarak kâğıt üzerinde oluşturulması ilerde sorun yaratabilir. Müşteri gereksinimleri değişikliğe uğrayabileceği için, yazılım sisteminin de yapısal değişikliğe uğraması kaçınılmaz olabilir. Böyle bir durum maliyeti artırır, çünkü yeni ve değişen gereksinimleri gerçekleştirimi için modelde yer alan safhaların birkaç kere uygulanması gerekebilir.
5. Sistemin tamamlanarak kullanılır hale gelmesi uzun zaman alabilir. Böylece eksikleri, hataları erkenden görmek mümkün olamaz.
6. Başlangıçta yapılan hataların tespiti çok uzun zaman alabilir. Bu hataların giderilmesi maliyeti yükseltir.
7. Modül gerçekleştirimleri için zaman tahminleri proje planlarını oluşturan yöneticiler tarafından yapılır. Teknik bilgiye sahip olmayan şahıslar tarafından yapılan bu tahminler çoğu zaman doğru değildir. Bu durum proje planlama sürecini negatif etkiler.

Proje başlangıcında her ayrıntıyı göz önünde bulundurmak mümkün olmadığı için, çağlayan modeliyle geliştirilen yazılım sistemlerinin müşteri gereksinimlerini tam tatmin etmediğini görmek olağandır. Bunun önüne geçebilmek için projenin başlangıç safhasında analiz için çok zaman harcanır ve müşteri gereksinimleri en ince ayrıntıya kadar tespit edilir. Aslında proje başlangıcıyla oluşturulan dokümanlar ileride eskimiş olur. Çünkü müşteri gereksinimleri piyasa ve rekabet koşulları gereği değişikliğe uğramış olabilir. Ne yazık ki çağlayan modeli bunları dikkate almaz ve müşterinin talep ettiği değişiklikleri aza indirmeye çalışır. Bunun bir sebebi de sonradan gelen değişiklik taleplerinin maliyeti yükseltmesidir, çünkü bu durumda çağlayan modelinde yer alan safhaların birkaç kere uygulanması gerekebilir.

Bu çerçeveden bakıldığında proje sonunda oluşan program müşterinin aktüel gereksinimlerini tatmin etmez durumda kalır. Program daha çok müşterinin proje başlangıcında sahip olduğu gereksinimleri tatmin edecek şekilde tasarlanmıştır. Değişiklik talepleri alınır fakat işleme konmaz. Bir sonraki sürüme bırakılır. Baştaki plan ve analiz bozulmasın istenir. Ama bu da sürümlerin geç kalmasına, arkadan gelmesine neden olur.

Çevik süreçlerde durum farklıdır. Çevik süreç değişimi kabul eder ve onunla yaşamayı kolaylaştırmak için yeni yazılım metotları sunar. Çevik süreçlerde iterasyon bazında çalışmalar sürdürülür. Her iterasyon bir ila dört haftalık zaman dilimlerinden oluşur ve çağlayan modelinde yer alan safhaları ihtiva eder. Aslında her iterasyon bir mini çağlayan modeli ihtiva ediyor diyebiliriz.

Çevik süreç modelinin avantajları şöyledir:

1. Çevik projelerde müşteri gereksinimleri ve bu gereksinimlerin karşılığı olan program paralel olarak gelişir. Müşteri projenin gidişatına her zaman müdahale etme yetkisine sahiptir. Bu uzun süren projelerde önem taşımaktadır, çünkü çoğu zaman proje başlangıcında müşteri kendi gereksinimlerini tam olarak bilmeyebilir. Zaman içinde oluşan ilk prototipler

müşterinin kafasında nasıl bir sistem istediği hakkında daha net bir resmin oluşmasını sağlar. Projedeki geri besleme mekanizmalarıyla müşteri gereksinimleri her zaman değişikliğe uğrayabilir.

2. Bu modelde geri besleme merkezi bir rol oynamaktadır. Çeşitli safhalarda sağlanan geri besleme ile projenin hangi durumda olduğu saptanır. Yazılımcılar hazırladıkları testler aracılığıyla geri besleme sağlayarak, gerçekleştirdikleri bileşenlerin hangi durumda olduklarını tespit ederler. Sürekli tümleştirme yapılarak programın hangi durumda olduğu geri besleme olarak elde edilir. Müşteriye kısa aralıklarla programın yeni sürümü sunularak, geri besleme sağlanır.

3. Her iterasyon başlangıcında müşteri tarafından dile getirilen gereksinimler analiz edilir ve uygulamaya alınacak olanlar seçilir. Müşteri her gereksinim için bir öncelik sırası belirler. Öncelik sırası yüksek olan gereksinimler öncelikli olarak uygulamaya alınır. Her iterasyon sonunda gerekli değerlendirmeler yapılarak, oluşan problemler tartışılır ve tekrar meydana gelmelerini engellemek için gerekli önlemler alınır.

4. Test güdümlü çalışıldığı için kod kalitesi çok yüksek olur. Gün boyunca yazılımcılar kendilerine değişik bir yazılımcıyı takım arkadaşı olarak seçerek *(pair programming),* gerçekleştirimi gerçekleştirirler. Kısa bir zaman sonra yazılımcılar arasındaki teknik bilgi aynı seviyeye gelir ve her yazılımcı programın herhangi bir bölümünde çalışacak hale gelir. Bu şekilde bir yazılımcının kod hakkında bilgi monopolüne sahip olması engellenir.

5. Yazılımcılar aktif olarak proje planlamasında yer alırlar. Onlar gereksinimlerin tespitinde müşteriye yardımcı olurlar ve zaman tahminlerinde bulunarak, proje planlaması için gerekli verilerin oluşturulmasını sağlarlar. Bu yazılımcılara belirli bir sorumluluk yükler. Kendisine güvenildiğini bilen ve sorumluluk sahibi bir yazılımcının özgüveni ve motivasyonu artar.

6. Çevik projelerde iyi bir çalışma ortamının ve temposunun oluşturulması fazla mesai yapılmasını engeller. Fazla mesai yapılmayacak diye bir kural yoktur. Lakin fazla mesai bir kural haline gelmemelidir. Bu durum tüm ekibin motivasyonunu negatif etkiler. Hatalar genelde isteksizce yer alınan fazla mesailerde meydana gelir. Proje çalışanları sekiz saat olan ve fazla mesai yapılmayan iş günlerinde daha verimli olurlar.

7. Müşteriye kısa aralıklarla çalışabileceği bir sürüm sunulur. Program tamamlanmamış olsa bile, müşteri hazır bölümleri kullanarak, yaptığı yatırımın hızlı şekilde geri dönmesini sağlanır.

8. Yazılımcılar ve müşteri arasında devamlı iletişim vardır. Yazılımcılar soru ve sorunları müşteri ile paylaşarak, kısa sürede çözüm üretebilirler. Müşterinin piyasadaki değişikliklere ve bununla birlikte rekabet ortamına ayak uydurabilmesi önemlidir. Çevik süreç hızlı reaksiyon göstererek, bu değişikliklere ayak uydurulmasını sağlar. Rekabete ayak uydurabilmek için hızlı reaksiyon gösterebilmek hayatı bir önem taşımaktadır. [8]

1.1.7. Çevik Metot Kullanan Takımların Özellikleri

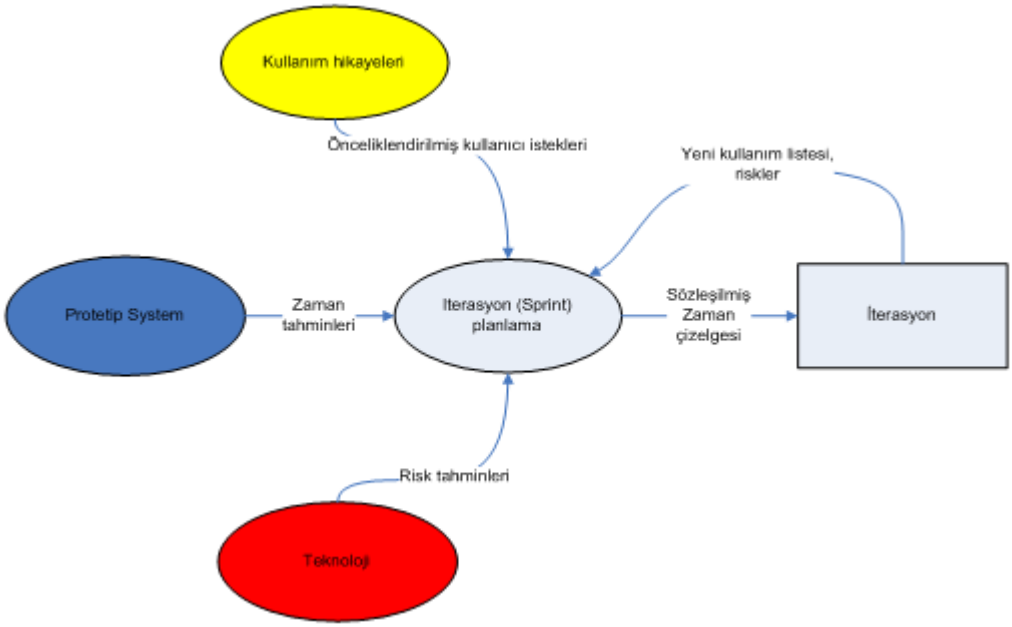
Bütün çevik yazılım geliştirme takımları hangi yaklaşımı (Scrum, XP, vs.) kullandıklarından bağımsız olarak aynı ana değerleri paylaşırlar. Bütün çevik takımlar:

Kısa aralıklarla ürün çıkarırlar: Takımlar her iki veya dört haftada bir, müşteriye çalışan işlevler üretirler. Ürünün teslim edilme zamanı değiştirilemez fakat zamanında teslim etmek için, verilecek işlev miktarı değiştirilebilir. Bu yüzden planlanan işlevlerin ne kadarının gerçekte verilen süre içinde üretilebildiğini raporlamak önemlidir.

Büyük ölçüde müşterileriyle birlikte çalışırlar: Mümkün olan her yerde tam zamanlı bir müşteri yazılım sürecine dâhil olur. Bu müşteri uygulama ilgili her konu üstüne karar verme yetkisine ve sorumluluğuna sahiptir. Bu çevik takımlar müşteriden çok daha sık geri besleme ve fikir alabiliyor demektir.

Erken ve sık testin önemini bilir: Çevik takımlar sürekli olarak kodlarını birleştirir ve hemen her durumda test etmeye çalışırlar. Teste yönelik uygulama geliştirme, daha çok hatanın üretim ortamına kadar geçmesini engeller.

Müşteri isteklerinin değişeceğini kabul eder: Her iterasyon başında takım müşteri isteklerini önem sırasına koyar veya tekrar düzenler ve o çalışılan iterasyonun kapsamını belirlerler. Takım bu müşteri istek listesinin zamanla değişeceğini kabul eder ve bütün isteklerin önceden tanımlanamayacağını ve tanımlanma zorunluluğu olmadığını anlar. Önem sırası belirleme işlemi, isteğin üreteceği değere göre yapılır. Böyle yaparak takım zamanla üretilen değeri raporlayabilirler.



Şekil 1.2 Basitçe çevik süreç

1.2. Çevik Metodolojiler

Çevik metotlar, kendi içerisinde özü aynı fakat süreçlerinde farklılaşan çeşitli alt kollara ayrılmaktadır. Zaman içinde, bir metamodel olarak kabul edebileceğimiz çevik süreci gerçekleştiren değişik çevik süreç türleri oluşmuştur. Yaygın olarak kullanılan Extreme Programming, Scrum, Agile Unified Process, Feature Driven Development (FDD), LEAN Development, Dynamic System Development Methodology (DSDM) ve Microsoft Solution Framework (MSF) çevik metodolojileri vardır. [1] Bu metodolojiler arasında en yaygın uygulananları XP ve Scrum'dır.

1.2.1. Extreme Programming

Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkarılmıştır. Extreme Programming kolay, grup içi iletişime önem veren, geri beslemelerin daha fazla olmasına imkân sağlayan bir yazılım geliştirme yöntemidir. [12] Yazılım geliştirmede kolaylığı ve esnekliği sağlamak için XP 12 farklı uygulamayı öngörür. Bunlar;

- Planlama oyunu
- Küçük ve kısa aralıklı sürümler (release)
- Metafor
- Basit tasarım
- Önce test
- Yeniden yapılandırma
- İkili programlama
- Ortak kod sahiplenme
- Sürekli tümleştirme (continuous integration)
- Haftada 40 saat çalışma
- Ekipte müşteri
- Kodlama standartları

1.2.2. SCRUM

SCRUM, Jeff Sutjerland ve Ken Schawaber tarafından 1990'ların ortalarında geliştirilen, çevik yazılım geliştirme metodolojileriyle uygulanabilecek bir proje yönetim yaklaşımıdır. Karmaşık yazılım işlerini küçük birimlere (sprint) bölerek geliştirmeye yarar. SCRUM karmaşık ortamlarda adım adım yazılım geliştiren küçük ekipler içindir. Gereksinimlerin kolaylıkla tanımlanamadığı ve kaotik durumların beklendiği projeler için uygun bir metodolojidir. Bu metodolojide bir birimin tamamlanması 30 günden fazla sürmemekte ve günlük 15 dakikalık toplantılarla sürekli iş takibi yapılmaktadır. SCRUM'ın 9 tane pratiği vardır. Bunlar;

- Genellikle 10-30 günlük iterasyon süreleri
- 5-10 kişilik takımlar
- Kısa süreli (3-12 ay) projeler
- İterasyon içerisinde takım dokunulmazdır
- Kendi-kendini düzenleyen, yönlendiren takım
- Her iterasyon müşteri güdümlüdür ve adapte olan bir planlama içerir.
- Kararlar kısa sürede alınır ve hemen uygulanır.
- Ayakta yapılan ve takım üyelerinin bir çember oluşturacak şekilde dizildiği ve belirli soruların sorulduğu günlük Scrum toplantıları.
- Her iterasyonun sonunda dış paydaşlara yapılan demo

1.2.3. Rational Unified Process

Agile Unified Process, Rational Unified Process (RUP)'in basitleştirilmiş bir versiyonudur. Agile Unified Process, RUP'a sadık kalarak çevik teknikleri ve kavramları kullanan yazılım geliştirme yaklaşımının anlaşılmasını kolaylaştıran bir metodolojidir.

Agile Unified Process aşağıdaki ilkelere dayanmaktadır:

- Personel ne yapacağını bilir
- Basitlik
- Çeviklik (beceriklilik)
- Yüksek değerli faaliyetlere odaklanma
- Araç bağımsızlığı
- İhtiyaçları karşılamak için, yazılım ürününün uyarlanması

1.2.4. Feature Driven Development

Feature Driven Development (FDD), Jeff De Luca tarafından doksanlı yılların sonunda geliştirilmiş bir çevik metodolojidir. FDD, özellik (feature, function) güdümlü çalışır. Sisteme yeni bir özellik kazandırılmadan önce, ayrıntılı bir tasarım çalışması yapılarak bu özelliği kapsayan mimari yapı oluşturulur. Bu yüzden FDD daha çok tasarım odaklı işleyen bir çevik süreçtir. 5 temel süreç vardır;

- Genel bir mimari geliştirme
- Özellik listesini oluşturma
- Özellikleri planlama
- Özellikleri tasarlama
- Özellikleri gerçekleştirme

1.2.5. Lean Development

Bob Charette tarafından geliştiren "Lean Development" 1980'de Japon otomobil endüstrisindeki yeniden yapılanmada kullanılan Lean üretimin prensipleri baz alınarak, 2000'li yılların başında yazılıma uyarlanmış bir çevik metodolojidir. Bob, geleneksel metodolojilerin risk olarak gördüğü değişimi, kısıtlayıcı yönetim uygulamalarıyla kontrol ederek bu değişimin yeni fırsatları üretilebilmesini sağlamıştır. 12 prensibi vardır;

- Müşteri memnuniyeti en önemli önceliktir
- Bütçeyi en iyi şekilde kullanır
- Müşteriyle sürekli iletişim başarıyı arttırır
- Her proje takım çalışmasıdır
- Her şey değişebilir
- Yama değil, kapsamlı çözüm
- Tamamla, yapılandırma
- Minimalism temeldir.
- Bugün %80 tamamlamış olmak, yarın % 100 tamamlayacak olmaktan iyidir.
- Teknolojiyi gereksinimler belirlesin
- Ürünün büyümesi demek özelliklerin büyümesi demektir
- Lean geliştirmeyi asla sınırlarının ötesine götürmeyin.

1.2.6. Dynamic System Development Methodology

Dynamic System Development Methodology (DSDM), hızlı yazılım geliştirme adımlarının kontrolünü sağlayan bir metodolojidir. Büyük Britanya' da bir konsorsiyum tarafından 1990'lı yılların ortalarında geliştirilmiştir. [1] Bu yaklaşımla fonksiyonel gereksinimlere öncelik verilmesi gereklidir. Aynı zamanda yüksek kalite ve değişen gereksinimlere adapte olunması önemlidir. Dokuz tane prensibi vardır. Bunlar;

- İş ihtiyaçlarına odaklanma
- Zamanında teslim
- İş birliği yapmak
- Asla kaliteden ödün verme
- İteratif geliştirme
- Sağlam temellerden artırımlarla büyür.
- Sürekli iletişimde olunmalı
- Kontrolde olduğunu gözle

1.2.7. Microsoft Solution Framework for Agile

Microsoft Solution Framework (MSF), ilk olarak Microsoft tarafından 1993 yılında versiyon 1.0 olarak yayınlanmıştır. Bunu sonraki yıllarda sırasıyla; 1997'de versiyon 2.0, 1999'da versiyon 2.5, 2002'de versiyon 3.0 ve en son 2005 yılında ise versiyon 4.0 yayınlanması izlemiştir. [13] MSF başarılı bilişim çözümleri için geliştirme ekibinin nasıl organize edileceği, projelerin nasıl planlanacağı, süreç yapısının nasıl gerçekleştirileceği, risklerin değerlendirilip kurulumun nasıl tamamlanacağıyla ilgili bir süreç yaklaşımıdır. MSF takım ve süreç olmak üzere iki modelden oluşmaktadır. [14]

Aşağıdaki prensipleri benimsemektedir:

- Müşteri ile birlikte geliştirme
- Açık iletişim kurmak
- Ortak bir vizyonda hareket etmek
- Kalitenin herkese ait ortak bir değer olduğuna inanmak
- Çevik olmak
- Günlük kurulum alışkanlığı edinmek
- Sürekli gelişim sağlamak
- Risk Yönetimi

1.3. Extreme Programlama

En popüler çevik süreçlerden birisi XP olarak bilinen Extreme Programming'dir. Kent Beck ve arkadaşları tarafından 1996 yılında Chrysler firmasında yapılan bir proje bünyesinde oluşan XP ihtiva ettiği basit ama bir o kadar etkili yöntemlerle yazılım sektöründe yeni bir rüzgârın esmesini sağlamıştır. XP ile oluşturulan çevik süreçte müşteri ve gereksinimleri merkezi bir rol oynamaktadır. Yazılım esnasında XP ile tam belirli olmayan ve çabuk değişikliğe uğrayan müşteri gereksinimlerine

ayak uydurulabilir. Bu konvansiyonel yazılım metotlarında zordur, çünkü proje öncesi müşteri gereksinimleri en son ayrıntısına kadar kâğıda dökülmüştür. Oluşan bir dokümantasyon temel alınarak, yazılım gerçekleştirilir. Proje ilerledikçe müşteri tarafından yapılması istenen değişikliklerin maliyeti çok yüksek olacaktır, çünkü mevcut yapı (tasarım) istenilen değişiklerin yapılmasını engelleyebilir ya da yeni bir yapılanmaya gidilmesi gerekebilir. XP, kullanıldığı projelerde formalite ve bürokrasinin mümkün en az seviyeye çekilmesine önem verir. Çevik olabilmek için az yükle yola çıkılması gerekmektedir. Bu yüzden proje öncesi geniş çapta tasarım ve dokümantasyon oluşturulmasına izin verilmez. [15,16]

1.3.1. XP Değerleri

XP dört değer üzerine kuruludur: Basitlik *(simp1icity),* İletişim*(communication),* geri besleme *(feedback)* ve Cesaret *(courage).*

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

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

Yazılımda en önemli konulardan birisi kalite kontrolüdür. XP projelerinde kalite kontrolü geri besleme üzerinden sağlanır. Yazılımcılar yazdıkları testlerden geri besleme alarak kaliteyi sağlarlar. Kısa zamanlarla yeni sürüm oluşturularak müşteri ve kullanıcılardan geri besleme aracılığıyla programın gereksinimleri tatmin edip, etmediği kontrol edilir. Yazılım esnasında sürekli entegrasyon yapılarak, programın

en son durumu hakkında geri besleme sağlanır. XP' nin uygulanabilmesi için değişik katmanlarda geri besleme mekanizmalarının oluşturulması gerekmektedir.

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

Basit çözümler, geri besleme ve iletişim için cesaret gereklidir. Bu değerler bireyler arası etkileşimi ve iletişimi artıracağı için, bireylerin kendi iç dünyalarını terk edip, takımın bir parçası olmalarını kolaylaştırırlar. Bu kişisel gelişmeyi sağlar ve temelinde kişisel cesaret yatar. [8]

1.3.2. XP Prensipleri

XP değerlerinden yola çıkarak 15 tane XP prensibi oluşturulmuştur. Bunlar:

1. Hızlı Geri Besleme (Rapid Feedback)

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

2. Basitliği Tercih Etmek (Assume Simplicity)

Basit çözümler kolay uygulanır ve kısa zamanda oluşturulur. Bu, geri beslemenin de hızlı bir şekilde gerçekleşmesini sağlar. Basit çözümlerin kavranması ve anlatılması daha kolaydır. XP, yazılımcılardan o anki gereksinimi tatmin etmek için basit çözümü bekler. Yazılımcı gelecekte oluşabilecek eklemeleri ve değişiklikleri

düşünmemeli, sadece ve sadece kendisinden o an için bekleneni en basit haliyle uygulamalıdır.

3. Artırımlı Değişiklik (Incremental Change)

Basit çözümler uygulasak bile, yazılım sistemleri zaman içinde karmaşık bir yapıya dönüşebilir. Yapılan en ufak bir değişiklik bile, sistemin düşünmediğimiz bir bölümü üzerinde hata oluşmasına sebep verebilir. Oluşabilecek bu hataları kontrol altında tutabilmek için değişikliklerin ufak çapta olması gerekmektedir. Büyük değişiklikler beraberinde büyük sorunları getirebilir. Bu sebepten dolayı değişikliklerin ufak çapta ve sıklıkla yapılması gerekmektedir.

4. Değişimi İstemek (Embracing Change)

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

5. Kaliteli İş (Quality Work)

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

6. Öğrenmeyi Öğret (Teach Learning)

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

7. Az Başlangıç Yatırımı (Small Initial Investment)

XP en modern ve pahalı araç gereçlerle projeye başlanmasını beklemez. Başlangıç giderleri ne kadar düşük tutulabilirse, projenin iptali durumunda kayıplar o oranda az olacaktır. Başlangıçta tüm takımın dar bir finansman korsesini giymesi sağlanarak, proje için daha önemli görevlere odaklanmaları sağlanır.

8. Kazanmak için Oyna (Play to Win)

XP takımları kazanmak için oynar. Her zaman gözlerinin önünde nihai sonuç vardır: Programı tamamlamak ve müşteriye teslim etmek.

9. Somut Denemeler (Concrete Experiments)

Verdiğimiz kararların sonuçlarını kontrol edebilmek için denemeler yapılır, çünkü alınan kararlar her zaman doğru olmayabilir. Bir kontrol mekanizmasına ihtiyacımız olduğu bellidir. Bu da somut denemeler aracılığıyla nerede olduğumuzu tespit etmektir. Bu somut denemeler yazılım sistemleri içinde geçerlidir. Örneğin testler hazırlayarak, oluşturduğumuz mimari ve tasarımı kontrol ederiz.

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

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

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

Bireysel içgüdünün yanı sıra, bireylerin oluşturduğu takımların da içgüdüsü vardır.

Eğer takım bir şeylerin doğru gitmediği hissine sahipse ve bunu dile getiriyorsa, o zaman bir şeyler yolunda gitmiyor demektir. Takımın içgüdüsüne kulak verilmelidir. Bunun göz ardı edilmesi, proje için olumsuz sonuçlar doğurabilir.

12. Sorumluluk Üstlenmek (Accepted Responsibility)

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

13. Sürecin Ortam Şartlarına Adapte Edilmesi (Local Adaptations)

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

14. 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.

15. Doğru Ölçüm (Honest Measurement)

Proje gidişatını kontrol edebilmek için değişik türde ölçümlerin yapılması gerekmektedir. Örneğin hazırlanan birim testleri ile sınıfların işlevleri kontrol edilir. Yapılan ölçümler doğru ve samimi yapıldığı taktirde kontrol mekanizması olarak

kullanılan ölçümlerin bir anlamı vardır. Yazılımcılar tarafından samimi ve doğru yapılmayan ölçümler projenin gidişatını olumsuz yönde etkiler. [10,12,17]

1.3.3. XP Pratikleri

Dört XP değer ve on beş XP prensibi on dört XP pratiği ile desteklenmektedir. XP pratikleri yazılımcıların XP değer ve prensiplerini uygulamada yardımcı olur. Kent Beck tarafından hazırlanan ilk XP versiyonunda 12 pratik yer almaktaydı. Diğer çevik süreçlerin de etkisiyle Standup-Meeting'ler ve geriye dönük toplantılar XP teknikleri arasına katıldı.

Bunlar:

1. Yazılımcıya yakın müşteri (On-site customer)

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

2. Ayakta toplantı (Standup-meeting)

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

3. Planlama Oyunu (Planning game)

XP projeleri iterasyonlu ve artırımlı yol alır. Bir sonraki iterasyonda yapılması gereken işleri planlama oyununda görüşülür ve sürüm ve iterasyonun içeriği tespit edilir. Planlama oyununa müşteri, kullanıcılar ve yazılımcılar katılır. Müşteri ve kullanıcılar daha önce kullanıcı hikâyesine dönüştürdükleri isteklerine öncelik sırası verirler. Yazılımcılar her kullanıcı hikâyesi için gerekli zamanı tahmin ederler. Kullanıcı hikâyelerinin öncelik sırası bu tahmine bağımlı olarak değişebilir. Planlama oyunlarında sürüm ve iterasyon planları oluşur.

4. Kısa aralıklarla yeni sürüm (Short releases)

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

5. Geriye bakış (Retrospective)

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

6. Mecaz (Metaphor)

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

7. Ortak mülkiyet (Collective ownership)

XP projelerinde yazılımcılar ortak sorumluluk taşırlar. Bu her kod parçasının herhangi bir yazılımcı tarafından gerekli durumlarla değiştirilebileceği anlamına gelir. Böylece yapılması gereken işler aksamaz, çünkü belli kod bölüm1erinden belli yazılımcılar sorumlu değildir. Aksine her yazılımcı programın her bölümü üzerinde çalışma hakkına sahiptir. Bir yazılımcının işe gelmemesi durumunda, başka bir yazılımcı kolaylıkla onun görevlerini üstlenebilir.

8. Sürekli entegrasyon (Continuous integration)

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

9. Kod standartları (Coding standards)

Yazılımcılar tarafından aynı kalitede kod yazılımı yapılabilmesi için, kod yazarken kullanılacak kuralların oluşturulması gerekmektedir. Kodun nasıl biçimlendirileceği, sınıfların, metot isimlerinin ve değişkenlerin nasıl isimlendirileceği kod

standartlarında yer alır.

10. Sürdürülebilir tempo (Sustainable pace)

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

11. Önce test etmek (Testing)

Oluşturulan programların kalite kontrolünden geçmesi gerekmektedir. Bu yazılım esnasında oluşturulan testlerle yapılır. Yazılımcılar, bileşenler için birim testleri hazırlar. Sınıf bazında yapılan bu testlerle bileşenlerin işlevleri kontrol edilir. Müşteri gereksinimlerini test etmek için kabul testleri hazırlanır. Bileşenlerin entegrasyonunu test etmek için entegrasyon testleri hazırlanır.

12. Sade tasarım (Simple design)

Yazılımcılar üstlendikleri görevleri *(task)* en basit haliyle kodlarlar. Bu programın basit bir yapıda kalmasını ve ilerde değiştirilebilir ve genişletilebilir olmasını sağlar. Sade bir tasarım yazılım sisteminin karmaşık bir yapıda olmasını önler. Bunun yanı sıra basit tasarımlar daha kolay ve daha hızlı gerçekleştirilir. Basit bir gerçekleştirimi anlamak ve anlatmak daha kolaydır.

13. Yeniden yapılandırma (Refactoring)

Tasarım hataları yazılım sisteminin daha ilerde tamir edilemeyecek bir hale dönüşmesine sebep verebilir. Bu yüzden bu hatalar hemen giderilir. Bu yeniden yapılandırma işlemine “refactoring” ismi verilir. Hazırlanan birim testleri ile yapılan

değişikliklerin yan etkileri kontrol edilir. Bu açıdan bakıldığında birim testi olmayan bir sistem üzerinde yeniden yapılandırma işlemi hemen hemen mümkün değildir, çünkü değişikliklerin doğurduğu yan etkileri tespit etme mekanizması bulunmamaktadır.

14. Eşli Programlama (Pair programming)

XP projelerinde iki yazılımcı aynı bilgisayarda çalışır. Bu sayede yazılımcıların kısa bir zaman içinde aynı seviyeye gelmesi sağlanır. Ayrıca bu kalitenin yükselmesini sağlar. [1,8,10,18]

1.3.4. XP Rolleri

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

Müşteri: Projenin var olma sebebi müşteridir. Müşteri ihtiyaç duyduğu ve gereksinimlerine cevap verebilecek bir yazılım sistemi için yatırım yapan kişidir.

Proje bünyesinde ne programlanması gerektiğini müşteri tayin eder. Müşteri yapılması gerekenleri kullanıcı hikâyeleri oluşturarak ifade eder. Yazılımcılar müşteriye bu süreçte yardımcı olurlar. Ama kullanıcı hikâyelerinin oluşturulma sorumluluğu büyük ölçüde müşteriye aittir. Her kullanıcı hikâyesi, yazılım sisteminin bir özelliğini tanımlar. Yazılımcıların gerçekleştirimi gerçekleştirebilmeleri için kullanıcı hikâyesini anlayabilmeleri gerekmektedir. Sadece bu durumda gerçekleştirim süreci için bir tahminde bulunabilirler. Ayrıca oluşturulan kullanıcı hikâyelerinin test edilebilir yapıda olması gerekmektedir.

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

Hangi kullanıcı hikâyelerinin uygulamaya alınacağına müşteri karar verir. Bu konuda müşteriye herhangi bir sınırlama getirilmez. Müşteri seçer ve yazılımcılar uygularlar.

Uygulanan kullanıcı hikâyelerini kontrol etmek amacıyla müşteri kabul testlerini tanımlar. Bu testler yazılımcı ya da testçi tarafından gerçekleştirilir. Kabul testleri kullanıcı hikâyesinin doğru gerçekleştirilip, gerçekleştirilmediğini kontrol edici bir mekanizmadır. [19]

Yazılımcı: Sistem analizi, tasarım, test ve gerçekleştirim yazılımcılar tarafından yapılır.

Müşteri tarafından hazırlanan kullanıcı hikâyelerinin gerçekleştirim süresi yazılımcılar tarafından tahmin edilir. Bu yazılımcıların XP projelerinde proje planlama sürecine dâhil edildikleri anlamına gelmektedir. Geleneksel projelerde bu tahminleri teknik bilgiye sahip olmayan yöneticiler yapmak zorundadır. Bu yüzden bu tahminler genelde gerçekleri yansıtmaz.

Her yazılımcı test güdümlü ve bir takım arkadaşıyla beraber çalışır. Eşli programlama olarak bilinen, iki yazılımcının birlikte yazılım yapması, kısa zamanda kod hakkındaki bilginin tüm yazılımcılar tarafından paylaşılmasını kolaylaştırır. Ayrıca eşli programlama yazılımcılar arasında iletişimi ve takım içinde çalışabilme özelliğini artırır. Test güdümlü çalışmak bakımı ve geliştirilmesi kolay kodun oluşmasını sağlar. XP projelerinde yazılımcılar herhangi bir satır kod yazmadan önce gerekli test sınıflarını oluşturarak gerçekleştirime başlarlar.

Yazılımcılar oluşturdukları testler yardımıyla her gün bir veya birden fazla şekilde modül entegrasyonu gerçekleştirirler. Sürekli entegrasyon yazılımcılar tarafından

oluşturulan modülleri entegre eden bir süreçtir. Her yazılımcı bu süreçten geri besleme sağlayarak, kendi yaptıklarının ne derecede sisteme entegre olduğunu ölçebilir. [15]

Proje Menajeri: Proje menajeri, müşteri ve yazılımcıları bir araya getirir. Onların beraber çalışabilecekleri ortamların oluşmasını sağlar. XP proje menajeri tek başına proje planlamasından sorumlu değildir. Yazılımcılara görev atamaz, onların kendi başlarına seçim yaparak, sorumluluk almalarını kolaylaştırır. Toplantı ve diğer buluşmaları koordine eder, takımın karşılaştığı sorunları ortadan kaldırmak için gerekli olanları yapar. [15]

Koç: Çevik süreci tanıyan ve nasıl uygulanması gerektiğini bilen uzmandır. Koçun görevi proje başlangıcında çevik takımı oluşturmak ya da bir araya getirmek ve onlara belirli bir süre rehberlik yapmaktır. Koç projede sorun çıktığı zaman ya da takım XP yöntemlerinin dışına çıktığında müdahale eder. Zaman zaman gerçekleştirimde aktif olarak rol alır. Örneğin birim testlerin nasıl doğru bir yapıda oluşturulabileceğini diğer yazılımcılara gösterebilir. [15]

Testçi: Müşteri tarafından oluşturulan kabul testlerini gerçekleştiren yazılımcıdır. Aynı zamanda birim ve tümleştirme testlerinin gerçekleştiriminde takım arkadaşlarına yardımcı olur.

1.3.5. XP Süreç İşleyişi

XP projelerinde projenin gidişatını genel olarak şu şekilde özetleyebiliriz.

- Müşteri gereksinimleri ihtiva eden kullanıcı hikâyelerini oluşturur. Bunlara öncelik sırası atar. Yazılımcılar her kullanıcı hikâyesi için uygulama zamanını tahmin ederler.
- Müşteri yazılımcılarla beraber iterasyon (1-2 hafta) ve sürüm (1-2 ay) planını hazırlar. Her sürüm birden fazla iterasyon ihtiva eder ve müşteri tarafından kullanılacak özellikte çalışır bir sistemdir.

- Müşteri ilk iterasyon (1 hafta) için gerekli kullanıcı hikâyelerini, tahminleri dikkate alarak seçer.
- Yazılımcılar iterasyon için seçilen kullanıcı hikâyelerini gerçekleştirirler ve varsa sorular müşteriye danışılarak cevaplandırılır.
- İterasyon sonunda yazılımcılar müşteriye çalışır bir sistem sunarlar. Müşteri sistemi değerlendirerek yazılımcılara geri besleme sağlar.

Edinilen tecrübeler ışığında bir sonraki iterasyon planlanır. Eğer müşteri yeni gereksinimlerin gerçekleştirilmesini isterse, bunlar için tekrar kullanıcı hikâyeleri oluşturulur ve tahminler yapılır. Eğer yeni kullanıcı hikâyeleri yoksa mevcut kullanıcı hikâye listesinden en yüksek öncelik sırasına sahip olanlar seçilir ve bir sonraki iterasyondan gerçekleştirime devam edilir. [12,15]

1.3.6. XP Proje Safhaları

Keşif Safhası (Exploration Phase): Projenin başlangıcında keşif safhasını oluşturan aktiviteler yer alır. Bu safhada müşteri kullanıcı hikâyelerini oluşturur. Yazılımcılar teknik altyapı için gerekli deney (spike) ve araştırmayı yaparlar.

Planlama Safhası (Planning Phase**):** Keşif safhasını planlama safhası takip eder. Bu safhada müşteri yazılımcılar yardımıyla iterasyon ve sürüm planlarını oluşturur. İterasyon planlaması için oluşturulan kullanıcı hikâyelerinin gerçekleştirim süresi yazılımcılar tarafından tahmin edilir. Müşteri kullanıcı hikâyelerine öncelik sırası vererek, iterasyonlarda hangi kullanıcı hikâyelerinin öncelikli olarak gerçekleştirimi gerektiğini tespit eder. Yazılımcılar tarafından herhangi bir kullanıcı hikâyesinin gerçekleştirim süresi tahmin edilemezse, yazılımcılar "spike solution" olarak bilinen basit bir çözüm gerçekleştirimiyle, kullanıcı hikâyesinin gerçek gerçekleştirimi için gerekli zamanı tahmin etmeye çalışırlar.

İterasyon ve Sürüm Safhası: Bir iterasyon bünyesinde gerçekleştirimi gereken kullanıcı hikâyeleri müşteri tarafından belirlenir. Gerçekleştirimin işlevini kontrol

etmek için müşteri tarafından kabul testleri belirlenir. Bu testler yazılımcılar ya da testçiler tarafından gerçekleştirilir. Her iterasyon sonunda müşteriye, çalışır bir yazılım sistemi sunulur. Bu şekilde müşterinin sistem hakkındaki görüşleri alınır. İterasyon son bulduktan sonra çalışma hızını tahmin etmek için bir önceki iterasyonda elde edilen tecrübeler kullanılır ve iterasyon planı bu değerler doğrultusunda gözden geçirilir. Bir önceki iterasyonda oluşan hatalar bir sonraki iterasyonda gözden geçirilmek ve giderilmek üzere planlanır. [15]

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

1.4. Scrum

Scrum, Kent Schwaber ve Jeff Sutherland tarafından 1990'ların ortalarında geliştirilmiş bir çevik süreçtir. Scrum Rugby oyununda kullanılan bir terimdir. Oyuncular kısa bir süre için bir araya gelerek, bir sonraki oyun hamlesi hakkında fikir alış verişinde bulunurlar, yani kısa bir toplantı yaparlar. Scrum daha çok proje yönetim metotlarına konsantre olmaktadır. Scrum, komplike yazılım ve ürünü basamak basamak geliştirerek kontrol ve yönetmeyi amaçlayan bir süreçtir. Yazılımın nasıl yapılması gerektiği hakkında ayrıntı içermez.

Scrum proje yönetimi, planlama ile ilgili yöntemlere odaklıdır. Mühendislik pratikleri ile ilgili detaylar içermez. Çevik kabiliyetleri tam anlamıyla kazanabilmek için XP geliştirme pratikleri gibi pratikler ile desteklenmesi şarttır.

Scrum takım içindeki iletişimi artırarak, yardımlaşmayı geliştirmeyi amaçlar. Takımlar iterasyon süresince sadece önceliği en yüksek olan iş gereksinimlerine odaklanarak, en fazla dört haftada bir bitmiş ürün kalitesindeki potansiyel ürünü

müşteriye geri besleme almak için sunarlar. Bu ürün yayımlanmaya hazır demek değildir sadece o iterasyondaki isteklerin doğrulanması için yapılan bir sunumdur.

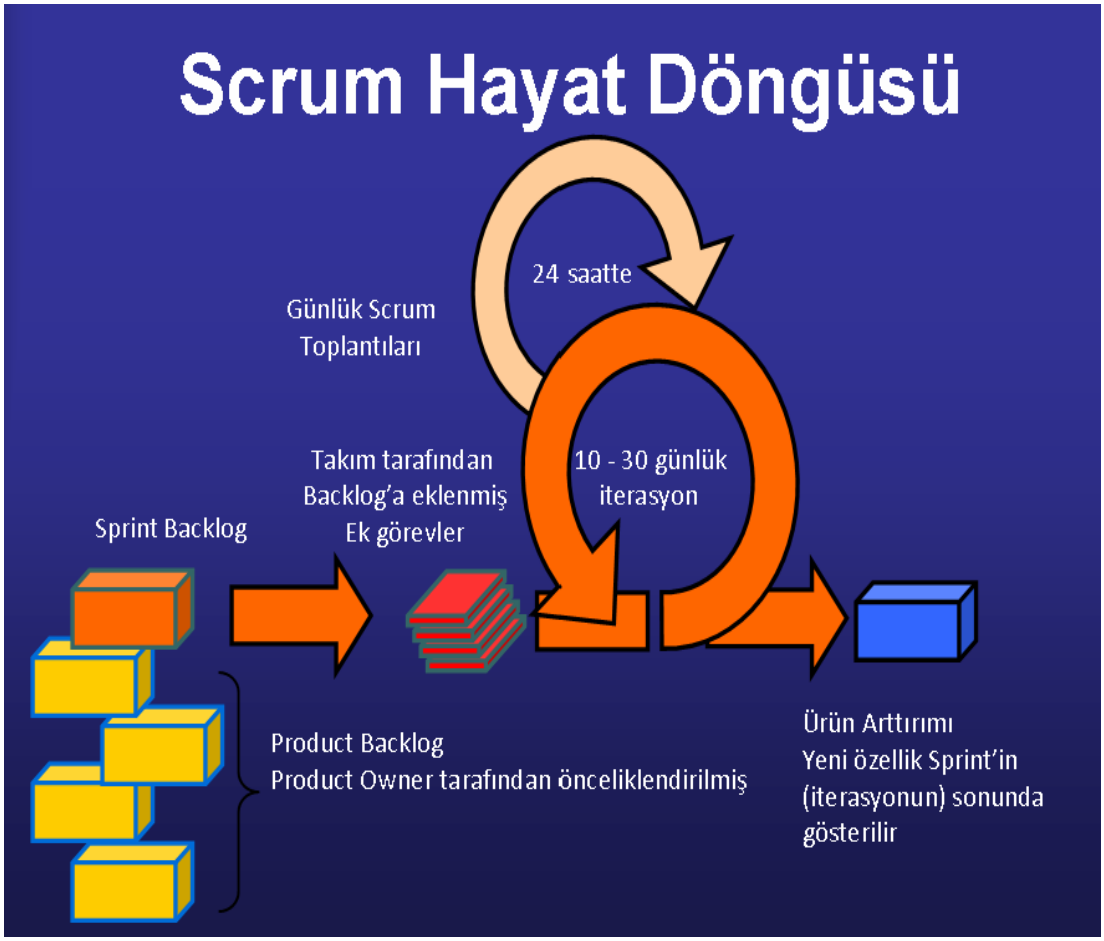
Takımlar karışık rollere sahip, kendi kendini organize eden kişilerden oluşur, müşteride takımın bir parçasıdır. Takımlar, her iterasyon süresince analiz, tasarım, geliştirme ve test yaparlar. Dokümantasyon, projedeki iletişimi sağlayabilecek en az seviyede tutulmaya çalışılır. [20]

Tablo 1.2'de Scrum'da ki hazırlık, geliştirme ve sürüm safhalarındaki amaç ve faaliyetler kısaca açıklanmıştır.

Tablo 1.2 SCRUM safhalarındaki amaç ve faaliyetler

HAZIRLIK		GELİŞTİRME	SÜRÜM
Planlama	**Hazırlama**		
Amaç Vizyonu oluştur, beklentileri tespit et, parasal desteği/ kaynakları güvence altına al/ elde et	Amaç Daha fazla gereksinimi belirle ve ilk iterasyon için gerekli olan miktarını önceliklendir	Amaç 30 günlük iterasyonlarla (Sprint'ler) sistemi kodla/gerçekleştir	Amaç Operasyonel uygulama
Faaliyetler Vizyonu,bütçeyi,ilk başlangıç *Product Backlog'u* oluştur/yaz, öğelere ait tahminleri oluştur. Keşfedici tasarım ve prototipler …	Faaliyetler Planlama Keşfedici tasarım ve prototipler …	Faaliyetler Her iterasyonda Sprint'in(iterasyonun) planlanması için toplantılar Sprint Backlog'unun tanımlanması, tahminler Günlük scrum toplantıları Sprint(iterasyon) gözden geçirmeleri(review) …	Faaliyetler Belgeleme Son Kullanıcı Eğitimi Pazarlama ve Satış …

Scrum temel bileşenleriyle Scrum'ın hayat döngüsü aşağıdaki şekilde verilmektedir.



Şekil 1.3 Scrum hayat döngüsü

Aşağıdaki bölümlerde scrum'ın temel bileşeni olan roller, çıktı ürünleri (artifacts) ve toplantılar anlatılacaktır.

1.4.1. Roller

Scrum'da roller ürün sahibi, scrum uzmanı ve takım olmak üzere üç tanedir.

- **Ürün Sahibi (Product Owner)**

    - Son ürünün sahibidir.
    - Ne yapılacağını söyler. Ürünün özelliklerini belirler, ürünün bitiş tarihini ve içeriğine karar verir.
    - Hangi sırada yapılacağını söyler. Ürün özelliklerinin önceliklerini değiştirebilir.
    - Genellikle ürün müdürüdür.
    - Projenin çıktılarını kabul veya reddeder.

- **Scrum Uzmanı (Scrum Master)**

    - Proje yöneticisi
    - Scrum doğru şekilde uygulanmasını sağlar.
    - Takımın karşılaştığı zorlukları ve sıkıntıları çözümler.
    - İletişimi koordine eder.

Scrum uygulamaları konusunda uzman kişi olarak adlandırılan scrum uzmanı tarafından sürekli tutulan istatistiksel raporlar ve iş takip sistemleri sayesinde takımdaki her bir üyeye verilen iş, işin durumu ve bunlara bağlı olarak elemanın performansı ölçülerek, daha sağlıklı öngörüler yapılabilir.

- **Takım Üyeleri / Geliştiriciler (Development Teams)**

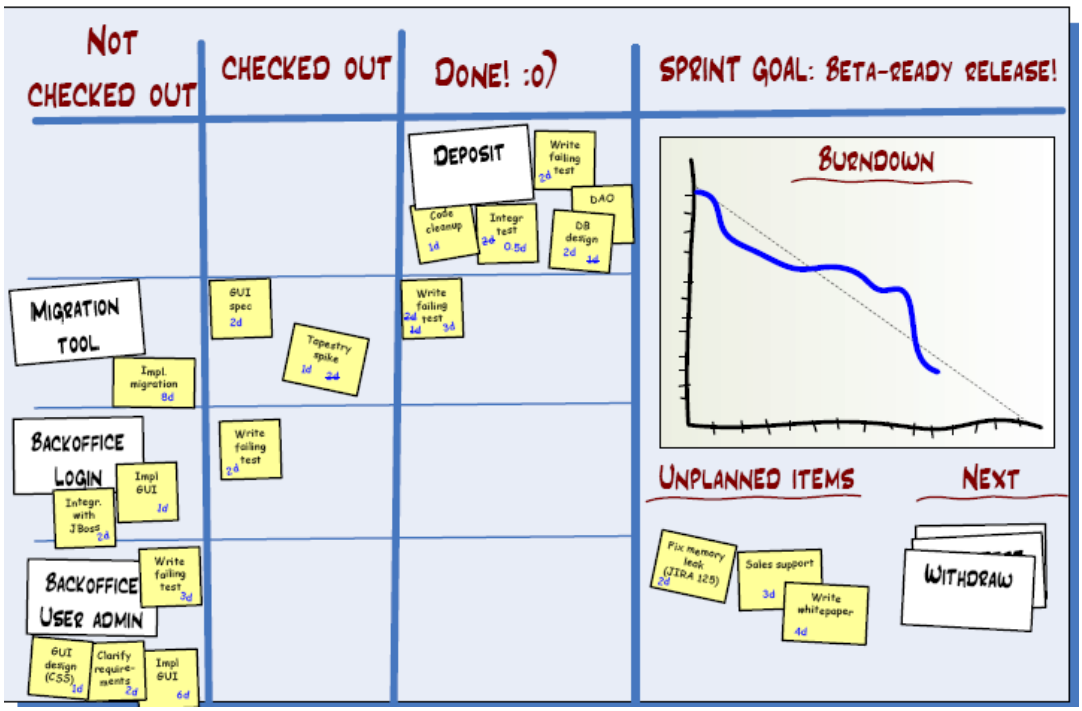
    - 5-10 kişiden oluşur.
    - Faklı rolleri bir arada üstlenen üyelerden (Analistler, Tasarımcılar, Geliştiriciler, Testçiler) oluşur.
    - Tam zamanlı çalışmalılardır.
    - Sprint süresince değişmemelilerdir.

1.4.2. Çıktı Ürünleri

Scrum'ın çıktı ürünleri; ürün gereksinim dokümanı, sprint dokümanı, sprint kalan zaman grafiği ve çalışan ürün'dür.

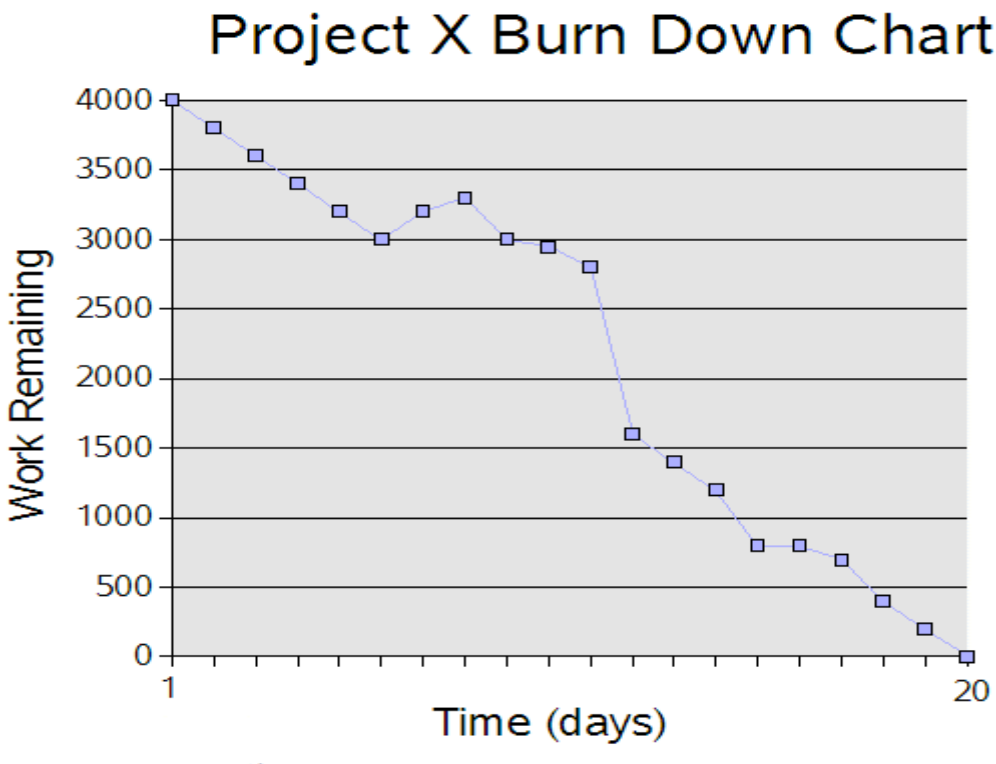
- **Ürün Gereksinim Dokümanı (Product Backlog)**: Projenin bütün gereksinimlerini içeren bir dokümandır. Scrum projelerinde ilk yapılan ürün sahibi ile bir masaya oturulup projenin kapsamını ifade eden "Product Backlog"u çıkartmaktır. "Product Backlog" proje sonunda üretilmesi beklenen sistemin özellik ve fonksiyonlarının önceliklerine göre yazıldığı bir liste dokümanıdır. Bu dokümanda, ürün sahibinin ihtiyaçları ve istekleri öncelik sırasına konarak listelenir. Proje süresince bu doküman ürün sahibi tarafından sürekli güncellenir; yeni fonksiyon ve özellikler listeye eklenir, mevcut fonksiyon ve özelliklerin öncelikleri değiştirilebilir.

- **Sprint Dokümanı (Sprint Backlog):** Scrum yöntemi ile yönetilen projeler "sprint" denilen fazlara ayrılır. Her bir sprintin süresi genellikle (en fazla) 30 gündür. İdeal olan proje boyunca tüm sprintler aynı süreli olur. Proje takımı (scrum team), proje yöneticisi (scrum master) ile her sprint başlangıcında masaya oturarak yeni "sprint" dahilinde geliştirilecek özellik ve fonksiyonları belirler. Bu süreçte seçilen fonksiyon ve özellikler daha önceden oluşturulmuş ve sürekli güncellenen Product Backlog listesindeki sıra ile en öncelikli olanlardır. Hiçbir zaman önceliği düşük bir özellik veya fonksiyon önceliği yüksek bir özellik veya fonksiyondan önce geliştirilemez. Bu bağlamda bazı sprintler de proje takımı, Product Backlog'dan 4 eleman, bazı sprint'lerde 25 eleman seçebilir. Seçilen özellik ve fonksiyonlar Sprint Backlog denilen ikinci bir listeye aktarılır. Proje takımı bir sonraki sprint başlangıcına kadar bir daha Product Backlog'a bakmaz, o sprint dahilinde sadece ilgili Sprint Backlog listesine odaklanır. Genelde sprint backlog listesindeki işler "iş tahtasına" (task board) yapılması gerekenler, yapılması devam edenler ve bitirilenler şeklinde yazılır.

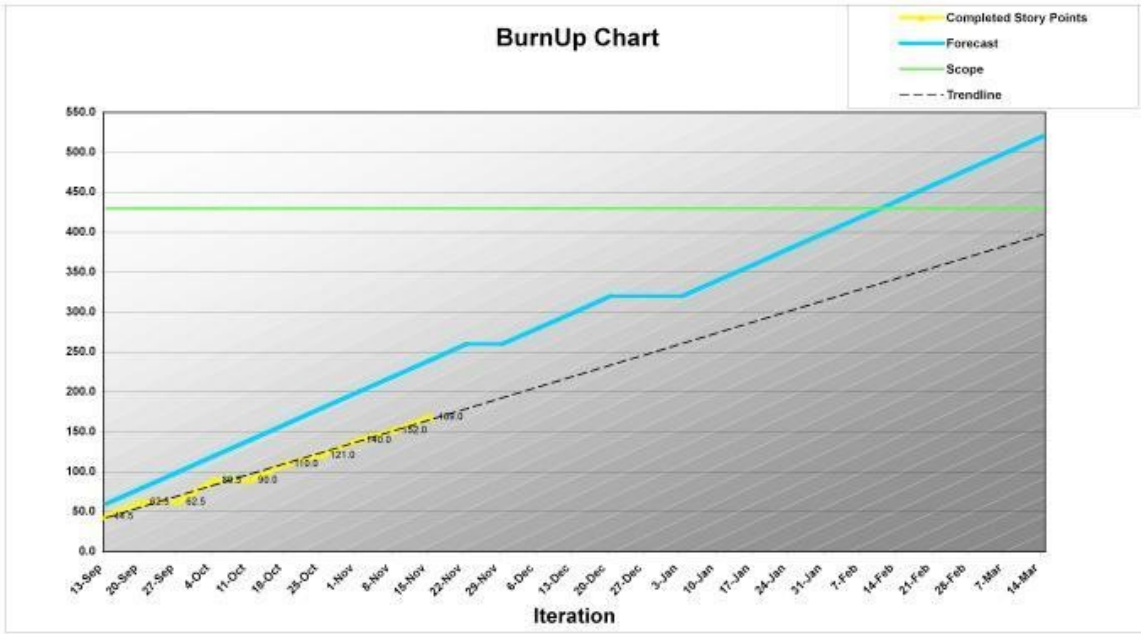


Şekil 1.4 İş tahtası (Task board) [22]

- **Sprint Kalan Zaman Grafiği (Burndown Chart):** Hedeflenen ve gerçekleşen işlerin grafik şeklinde gösterimidir. Burn Down ve Burn Up grafikleri proje kapsamındaki oynamaları, bunların zamana etkilerini, işlerin bitirilme hızını kısaca gidişatı özetler. Dikeyde kartların alınan puan toplamları, yatayda ise zaman yer alır. Projede meydana gelen değişikliklerin etkileri bu grafiklerde göze çarpar. Örneğin yavaşlamalar, işlerin ilerlemiyor olması gibi. [22]



Şekil 1.5 "BurnDown" Grafiği [23]



Şekil 1.6 "BurnUp" Grafiği [23]

- **Ürün (Product):** Test edilmiş, çalışır durumda ve isterleri karşılayan üründür.

1.4.3. Toplantılar

Scrum'da sprint planlama toplantısı, sprint planlaması, günlük scrum toplantısı, sprint gözden geçirme toplantısı ve hesaplama/ tahmin etme toplantısıdır.

- **Sprint Planlama Toplantısı (Sprint Planning Meeting):** Ürün sahibinin öncelik sırasına göre verdiği ürün taleplerinden bir tanesinin geliştirme sürecine alınmasıdır. Bu toplantıda, aşağıdaki sorular yer almaktadır.

    - Hangi backlog'lar yapılacak?
    - İterasyonun hedefi ne?
    - Takım elemanları kimler?
    - Günlük scrum nerede ve kaçta yapılacak?
    - Demo ne zaman yapılacak?

- **Sprint Planlaması (Sprint Planning):** Takım ve ürün sahibi sprint'in amacını belirler ve o sprint için yapılacak istekleri ayrıntılandırıp, her istek için süre belirler. Belirlenen süre hiçbir iş için bir günden fazla olamaz. Bu işler proje için listelenir.

- **Günlük Scrum Toplantısı (Daily Scrum Meeting):** Günlük scrum toplantısı takımın kendisini organize etmesi için çok önemlidir. Bu toplantılar takım üyelerinin bir biri ile eş zamanlı çalışmasını, takım arkadaşlarının güncel kalmasını, sorunlardan haberdar olmasını ve sorunların çok hızlı bir şekilde çözülmesini sağlar. Günlük scrum toplantıları her gün aynı saatte ve aynı yerde yapılır, bütün takım üyeleri katılır ve toplam 15 dakika sürer. Dün ne yaptın? Bu gün ne yapacaksın? Seni engelleyen bir sorun var mı? sorularına

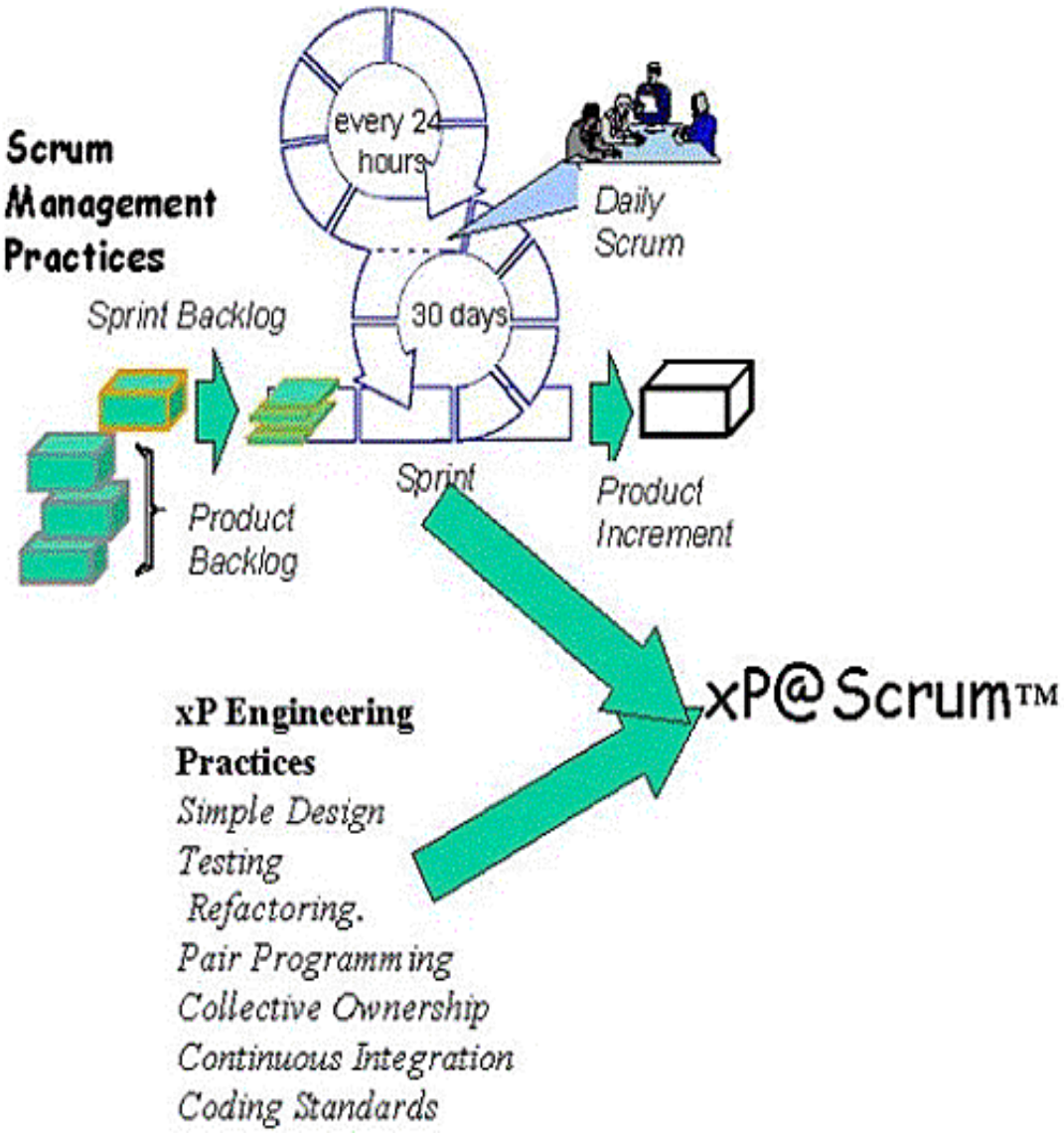
cevap verilir. Bu soruları cevaplarken verilen cevaplar, scrum ustası tarafından mümkün olduğunca kısa zamanda çözülmelidir.

- **Sprint Gözden Geçirme Toplantısı (Sprint Review Meeting):** Bir sprintin sonunda 1-2 saat arasında yapılan ve o iterasyonun sonunda ortaya çıkarılan ürünün değerlendirilmesi ve varsa son düzeltmelerin yapılmasıdır. Bu toplantıda neyi iyi yaptık? , daha iyi yapabiliriz, farklı yapmalıyız? sorularına cevap aranır.

- **Hesaplama/Tahmin Etme Toplantısı (Estimation Meeting):** Ürün sahibi ve takım bütün ürün istek listesinin üstünde çalışarak ürünün kaç sprintte çıkacağını ve her sprintte isteklerin nasıl önceliklendirileceğini, hangi işin ne kadar süreceğini hesaplamaya ve tahmin etmeye çalışırlar. [22]

1.5. XP Mühendislik Pratiklerinin Scrum ile Uygulanması

Yazılımcılar, geliştirme süresince sürekli "refactoring" yapmaya ve tasarımlarını geliştirmeye odaklanır. Bütün takımın amacı fonksiyonel ve fonksiyonel olmayan bütün istekleri en basit şekilde çözmektir. Yazılımcılar birim testleri kullanarak ürününü geliştirirler ve birim testlerin kapsama alanı sürekli ölçülüp raporlanır. Takım çok kısa aralıklarla kodu bütünleştirerek, stabil eder. Bu süre boyunca tamamlanan kullanım hikâyeleri hemen testçiler tarafından test edilir.



Şekil 1.7 Scrum süreçleriyle XP mühendislik pratikleri [24]

2. PROJEDE KULLANILAN YAZILIM GELİŞTİRME ARACI

İş dünyası ve teknolojideki hızlı gelişmeler çabuk karar almayı, plan ve programlarda hızlıca değişime uygun düzenlemeleri yapabilmeyi, gerektiğinde iş gücü kapasitesini artırmayı, azaltmayı yani her açıdan çevik olmayı gerektirmektedir. Bu kadar çeviklik gerektiren bir iş dünyasına yazılım üretirken, geleneksel yöntemlerle yazılımın başından sonuna müşteri gereksinimlerinin hiç değişmemesini bekleyerek yazılım geliştirebilmeyi ummak gerçeklik dışı olacaktır. İş dünyası için geliştirilen yazılım projelerinin amacı (özellikle de müşteriye özgü geliştirilen yazılımlar) iş dünyasına hizmet etmek olduğundan, proje geliştirme süreci en az müşterinin işi kadar çevik olmalıdır.

Çevik yazılım geliştirme metodolojilerinden ekibe, müşteriye uygun pratiklerin seçilmesi kadar kullanılacak aracın seçimi de önemlidir. Çevik proje yönetimleri için günümüzde ücretli ve ücretsiz birçok çevik yazılım geliştirme aracı bulunmaktadır. Thoughtworks Mingle, VersionOne, Rally Software gibi yazılımlar çevik proje yönetimi için özel tasarlanmış, kullanıcı hikayelerinin takibi, sürüm ve iterasyon planlaması, burn down grafikleri gibi kabiliyetleri içeren yazılımlardır.

Yazılımcıların kullandığı araçlarda özellikle testlerin kolayca yazılmasını ve hızlı şekilde çalıştırılmasını sağlayan araçlar kullanılır. xUnit test araçları ailesi, fonksiyonel testler için Selenium, Fit, Fitnesse, Watir bu tür araçlardır.

Ayrıca otomatik kurulumlar için Ant, Maven, MsBuild, Rake gibi araçlar kullanılır. Bunun yanında sürekli entegrasyon için CruiseControl , Hudson, Atlassian Bamboo gibi araçlar kullanılır. Bunlar Subversion, CVS gibi versiyon kontrol sistemlerine entegre çalışarak kısa aralıklarla otomatik kurulumlar gerçekleştirirler.

Hataların takibi için Jira, Trac gibi araçlar kullanılır. Proje kodlama standartları, potansiyel problemlerin analizi, test kapsam raporları gibi konularda da birçok araç mevcuttur. Bu araçların seçiminde ana kriterlerden biri, otomatik kurulum sürecine dâhil olabilme kabiliyetleridir. PMD, FindBugs, Clover, Cobertura, NDepend, FxCop bu tür kabiliyete sahip olan araçlardır. [23]

Bu tezin konusu olan ve gerçekleştirilen projede çevik proje yönetim aracı olarak Visual Studio Team System seçilmiştir. Bu bölümde VSTS'in özellikleri, mimari yapısı, Team Explorer ve raporlama alt yapısı incelenecektir.

2.1. Visual Studio Team System

Visual Studio Team System yazılım projelerinde proje ekibinde yer alacak kişilerin farklı rolleri üstlenmelerini ve bu rollere uygun olarak Visual Studio içerisinden görevlerini yapabilmelerini sağlayan yazılım geliştirme sistemidir. VSTS yazılım projelerinde yer alan yazılım mimarisinin tasarımı, kalitenin sağlanması, takım içerisinde etkin iletişimin sağlanması ve testlerin yapılabilmesi mümkündür.

VSTS, 4 temel hedef üzerine tasarlanmıştır :

- Servis temelli mimari (SOA) uygulamaları geliştirilmesi kolaylaştırmak.
- Tüm takım elemanları için, entegre araçlar ve daha iyi iletişim olanaklarının olduğu bir çalışma ortamı sağlamak.
- Dağıtık ekiplerin de uzaktan erişebileceği, ölçeklenebilir, güvenli bir yazılım geliştirme süreç yönetimi (SDLC) platformu sağlamak.
- Özelleştirilebilir süreç yönetimi sağlamak.

VSTS'nin en önemli avantajları aşağıdadır:

- Yaygın olarak kullanılan Visual Studio geliştirme ortamı entegrasyonu
- Diğer Microsoft ürünleri ile entegrasyon (MS Excel, MS Project, MS Sharepoint, MS SQL SERVER veri tabanı, veri ambarı)
- Süreç ile ilgili zengin araçlar
- Yazılım süreç yönetimi için entegre ve ihtiyaçlara uygun bir şekilde özelleştirilebilir ve genişletilebilir bir platform

Team Foundation Server (TFS), tüm VSTS için bir alt yapı (Framework) sunmaktadır. TFS ile yazılım geliştirme sürecinde sunulan araçlar ortak bir veritabanı üzerinden entegre edilmiş ve takım içinde birlikte çalışma alt yapısı sunulmuştur. Süreç yönetimi, proje yönetimini destekleyen araçlar, kaynak kod yönetimi, ürünleştirme servisi, merkezi raporlama, dağıtık ekipleri desteleyen alt yapısı, servis temelli mimarisi ve geliştirilebilir modeli en önemli özellikleridir.

TFS, kurumsal projeler için tasarlanmış, takım içerisindeki iletişimi ve buna bağlı olarak verimi arttırarak, iş durumunu izleme, takım rollerini yönetme, iş süreçlerini çıkartma ve sunduğu araçlarla proje yönetimini kolaylaştırarak ürünü ortaya çıkartmada etkin rol oynar. Team Foundation'ın ilk amacı, takım içerisindeki işbirliğini arttırarak ürün gelişimini kolaylaştırmak ve tamamlamaktır. Team Foundation Server, sunduğu teknoloji ile küçük projelerden, yüzlerce yazılım uzmanının yer aldığı çok geniş projelere kadar proje ihtiyaçlarının hemen hemen hepsini karşılayabilecek bir alt yapı sunmaktadır. Sunduğu özellikler şöyledir:

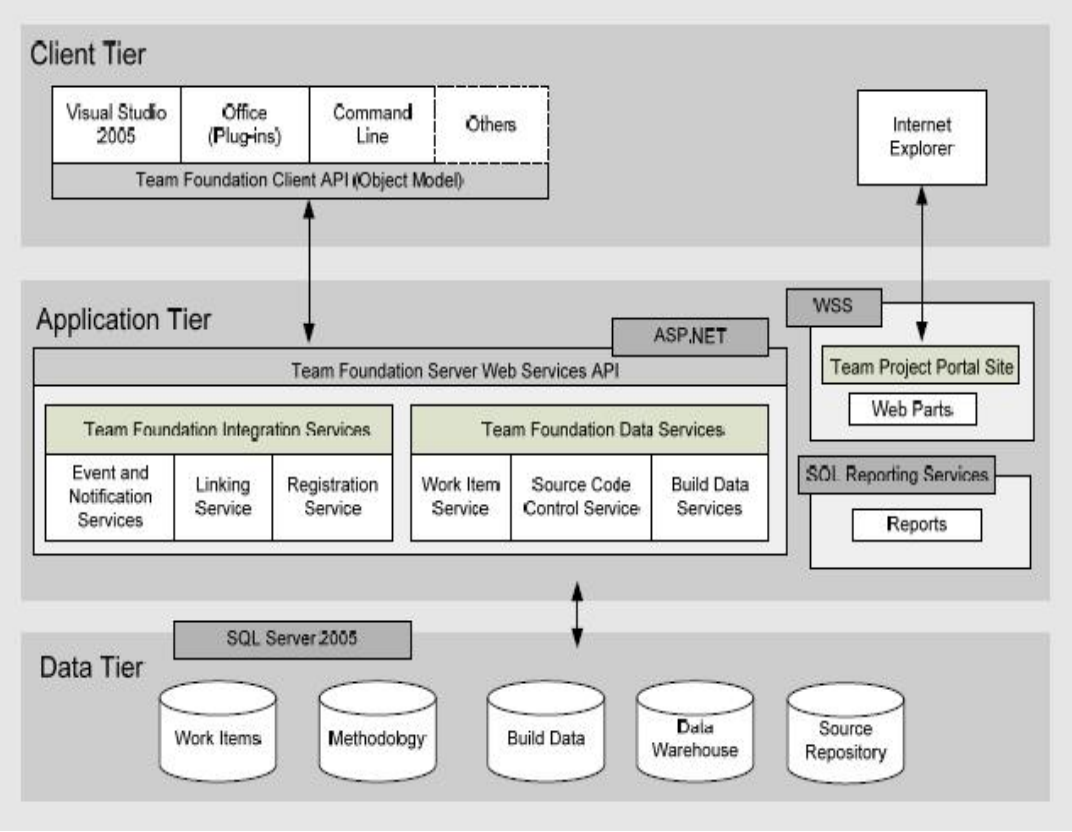
- Versiyon Kontrolü (Version Control)
- İş nesnelerinin izlenmesi (Work Item Tracking [Project Management])
- Takım Portalı (Team Portal)
- İş Zekası ve Raporlama (Business Intelligence and Reporting)
- Ürünleştirme Sunucusu (Build Server)
- Web Erişimi (Web Access)

TFS ile aşağıdaki imkânlar sağlanmıştır:

- Projelerin oluşturulması ve yönetimi (Team Projects)
- Süreç yönetimi (MSF, Scrum veya özel süreç şablonları)
- Projeler ile ilgili iş takip nesneleri ve iş akış yönetimi
- Kaynak kod ve versiyon yönetimi
- Proje ile ilgili yönetim ve durum raporlarının sağlanması
- Proje ürünleştirme sunucusu (Build Server)
- Doküman ve raporların paylaşıldığı, Sharepoint proje web sitesi

2.2. TFS'in Mimari Yapısı

Team Foundation Server; aşağıda yer alan Şekil 2.1'de de görülebileceği gibi Client Tier (istemci katmanı), Application Tier (uygulama katmanı) ve Data Tier (veri katmanı)' dan oluşan üç katmanlı bir mimariye sahiptir.



Şekil 2.1 Team foundation mimarisi [25]

2.2.1. İstemci Katmanı

İstemci katmanı, uygulama katmanının sunduğu web servislerini kullanarak sahip olduğu arayüz ile kullanıcıdan gelen isteklere cevap vermektedir.

İstemci katmanında kullanılan araçlardan bazıları aşağıdaki gibidir;

Visual Studio 2005 Team Suite : Visual Studio Team System'in bir parçası olan Team Foundation alt yapısını kullanan ve sunduğu araçlarla takım içerisinde yer alan üyelere hitap eden bir üründür. Sunduğu alt ürünler; Veri tabanı uzmanları için Visual Studio 2005 Team Edition for Database Professionals, yazılım mimarları için Visual Studio 2005 Team Edition for Software Architects, yazılım uzmanları için Visual Studio 2005 Team Edition for Software Developers ve son olarak da geliştirilen ürünleri test eden test uzmanları için Visual Studio 2005 Team Edition for Software Testers 'dır.

Team Explorer: Visual Studio'nun bir çok sürümü üzerine kurulabilen Team Explorer, Team Foundation Server'a bağlanarak proje içeriğini paylaşma ve projeleri düzenlemede etkin rol oynar. Team Explorer kullanarak, üzerinizdeki görevleri görebilir, düzenleyebilir, izin verilen proje kaynaklarına erişebilir, günlük takım faaliyetleri Team Foundation Server'a bağlanılarak yerine getirilmesi sağlanmaktadır.

Microsoft Office Uzantısı: Microsoft Office Excel ve Microsoft Project'i kapsayan uzantıları kullanarak Work item'lar Microsoft Office Excel içerisinden sorgulanmakta, Microsoft Project kullanarak Work item'ları düzenleyerek proje planları oluşturulmaktadır.

3. parti araçlar: Team Explorer'ın özelliklerinin bir kısmını barındıran ve farklı ortamlardan Team Foundation projelerine bağlanıp şirket içindeki bir terminal yerine şirket dışından da erişimi sağlayan 3. parti araçlar mevcuttur. Bunlardan bazıları, DevBiz tarafından geliştirilen ve DevBiz'in Microsoft tarafından satın alınmasıyla

artık Microsoft'un bir ürünü olan Team9Plain Web Access ve Teamprise diğer 3. parti araçlardır.

2.2.2. Uygulama Katmanı

Uygulama katmanı, istemci katmanının erişimi için ASP.NET Web servislerini sunmaktadır. Sunduğu web servisleri şöyledir:

- Team Foundation Data Services
- Team Foundation Integration Services

2.2.3. Veri Katmanı

TFS, istemci katmanının veri katmanına direkt erişimine izin vermemektedir. İstemci tarafından gelen tüm istekler web servisleri aracılığıyla uygulama katmanına iletilmektedir. Veri katmanı, SQL Server 2005 üzerine inşa edilmiştir.

Uygulama katmanı için sunduğu veri ambarları;

- İş nesnesi İzleme: İş nesneleri ile ilgili tüm veriler burada tutulur.
- Versiyon Kontrolü: Kaynak kontrolü ile ilgili tüm veriler burada tutulur.
- Team Foundation Build: Team Build özellikleri ile ilgili tüm bilgiler burada tutulur.
- Raporlama Ambarı: TFS araçları ve özellikleri ile ilgili tüm bilgiler burada tutulur.

2.3. VSTS'de Genişleyebilirlik ve Özelleştirme

VSTS ile yazılım geliştirme sürecinin entegrasyonu için ortak bir alt yapı kullanılmış, özelleştirilebilir ve genişleyebilir bir mimari de tasarlanmıştır. Kurumlar, yazılım geliştirme de takip ettikleri süreçlere göre, ürünü özelleştirebilirler. Çözüm ortakları tarafından geliştirilen ürünleri, yazılım geliştirme süreçlerine kolaylıkla

entegre edebilirler. Entegrasyon alt yapısı sayesinde, bu araçlar da sürecin bir parçası haline gelmektedirler.

VSTS, yazılım geliştirme sürecinin yönetilmesi ile ilgili gerekli şablonları ve araçları içermektedir. Kurumların ve ihtiyaçların çeşitliliği göz önüne alınırsa, özelleştirme ve genişleyebilirliğin önemi daha da anlaşılmaktadır. Gerek duyulduğu takdirde, çözüm ortakları tarafından geliştirilen, süreci daha da kuvvetlendirecek araçların kullanımı ile planlanan çalışma ortamına daha kolay ulaşabilmek imkân dâhilindedir.

VSTS özelleştirme seçenekleri genel olarak aşağıdaki tabloda sunulmuştur.

Tablo 2.1 VSTS'in özelleştirme seçenekleri

Özelleştirme yeri		**Açıklama**
Süreç şablonları genel özelleştirme		Süreç şablonu, Team System projesi oluşturulurken kullanılır. Proje ile ilgili rapor, iş takip nesneleri, kaynak kod yönetimi iş kuralları, Sharepoint sitesi yapısı ve içeriği, süreç hakkındaki bilgiler ve yetkilendirme bilgilerini ihtiva etmektedir. Süreç şablonlarının özelleştirilmesi için VSTS içinde araç sunulmamıştır. XML editör kullanılarak, özelleştirme yapılabilir.
Süreç şablonları	İş takip nesneleri (Work Items)	Yeni iş takip nesneleri tanımlanabilir ve mevcut olanlar özelleştirilebilir.
	Kaynak kod yönetimi	Her bir proje için, kaynak kodu yönetimi ile ilgili iş kuralları tanımlanabilir.
	Sharepoint sitesi	Sharepoint sitesindeki dokümanlar düzenlenebilir ve sitenin daha iyi bir

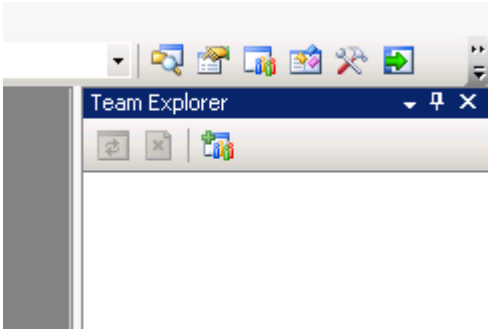
		birlikte çalışma ortamı olması için, Sharepoint servislerinin özelleştirme imkânları kullanılır.
	Raporlar	Raporlama servisleri kullanılarak, VSTS veri ambarında saklanan bilgilerden hazırlanan mevcut raporlar düzenlenebilir ve yeni rapor hazırlanabilir.
TFS temel servisleri üzerinden genişleyebilirlik (Team Foundation Core Services)		Bu servisler ile VSTS platformunda sunulan ürünlerin entegrasyonu gerçekleştirilebildiği gibi, çözüm ortakları tarafından geliştirilen ürünlerde sürece entegre edilebilmektedir. Entegre edilen araçlar üzerinde üretilen bilgilerde, veri ambarına aktarılabilmektedir.
Ürünleştirme süreç tipleri (Build Types)		Yeni ürünleştirme süreç tipleri tanımlanabilir.
Genişleyebilirlik araç seti		VSTS nesne modeli üzerinden daha geniş özelleştirme yapılabilir.

VSTS, yazılım geliştirme sürecinde ihtiyaç duyulan temel bileşenleri sunmaktadır.

2.4. TFS'de Team Explorer

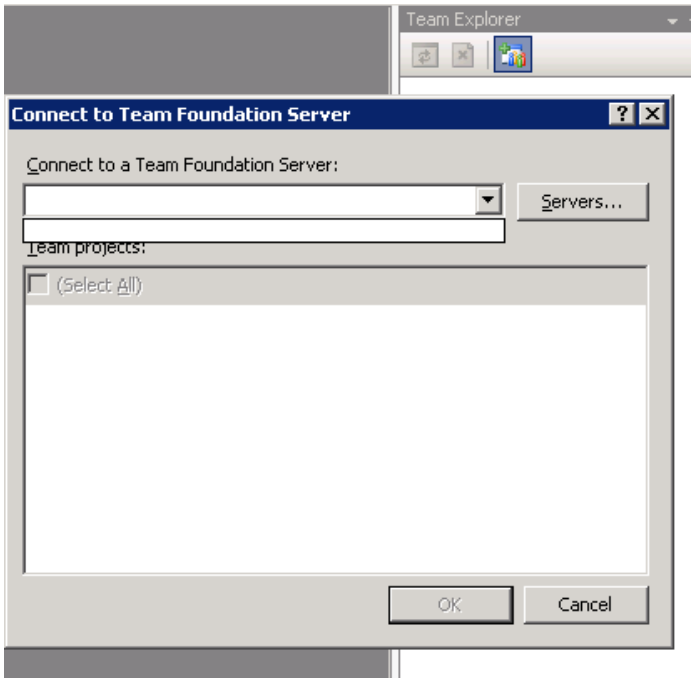
Team Foundation Server ile çalışabilmek için her kullanıcı profilinin kullanabileceği istemciler vardır. Örneğin bir proje yöneticisi Excel veya Project yazılımlarını kullanarak Team Foundation Server ile çalışabilmektedir. Yazılım geliştirici için Team Foundation Server istemci yazılımı Visual Studio 2005 içinde bir pencere olarak yer alan Team Explorer'dır. Team Explorer Visual Studio 2005 ile beraber gelmez. Visual Studio Team Foundation Server kurulum cd'si içinde bulunur ve Visual Studio 2005 yüklü olan bir bilgisayar üzerine kurulur. Team Explorer

kurulduktan sonra Visual Studio açıldığında View menüsü altından Team Explorer seçilerek Team Explorer penceresi açılır. Aşağıdaki resimde Team Explorer penceresi ilk açıldığındaki boş hali gözükmektedir.



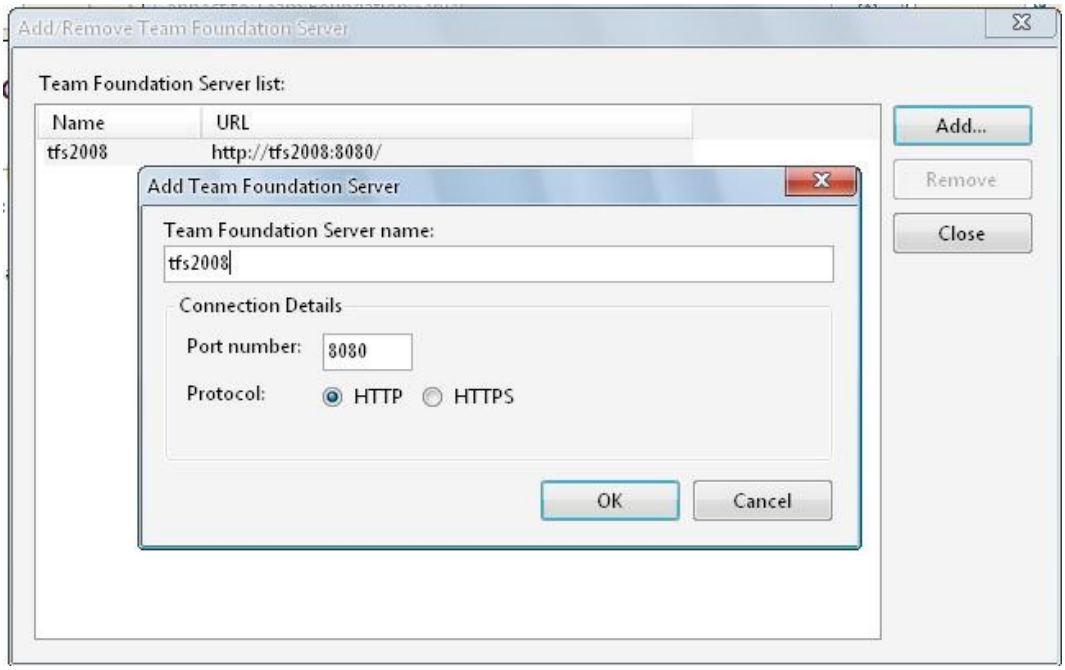
Şekil 2.2 Team Explorer ilk başlatıldığındaki ekran görüntüsü

Team Explorer penceresi ile hangi Team Foundation Server üzerinde çalışılacağını belirlemek için pencere dâhilinde yer alan üç düğmeden oluşan toolbar kullanılır. Bu toolbar üzerinde en sağda ve aktif olan tek düğme olan "Add Existing Team Project" düğmesine basarak aşağıdaki Team Foundation Server seçme ve seçilen sunucu üzerindeki erişilebilir projeleri seçme ekranına ulaşılabilmektedir.



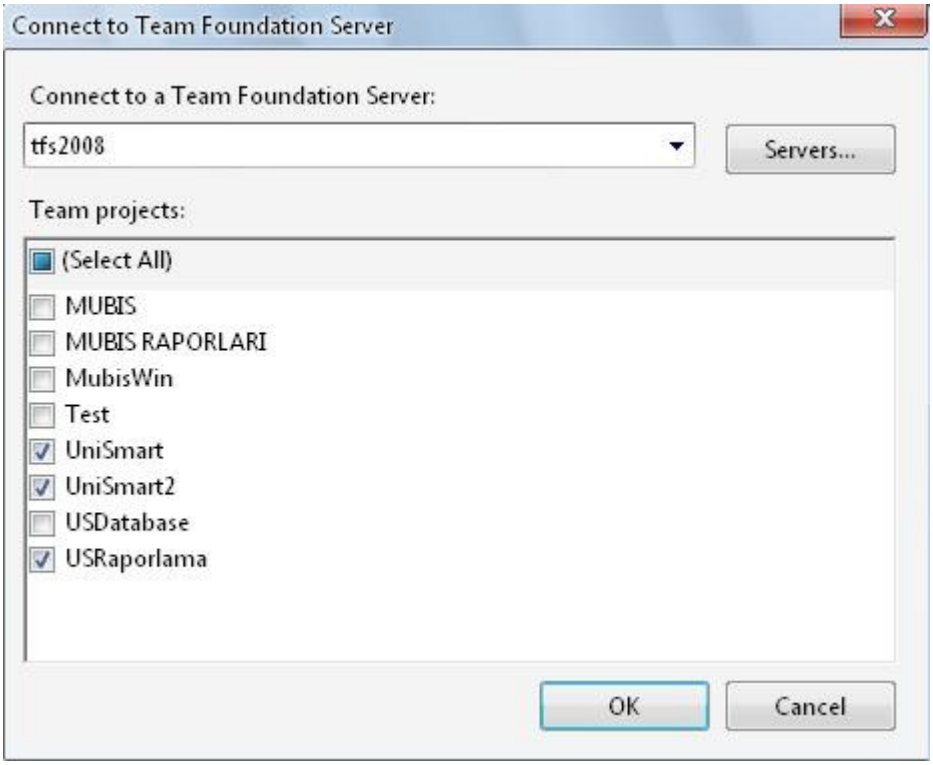
Şekil 2.3 Team Foundation Server seçme ekranı

Sunucu listesine yeni sunucu eklemek için “Servers” butonu kullanılır. Bu butona tıklandığı takdirde Team Foundation Server bilgilerini gireceğimiz aşağıdaki resimde altta kalan pencere açılır. Add butonu ile yeni bir Team Foundation Server bilgisi girilebilir.



Şekil 2.4 TFS ekleme ve çıkarma ekranı

Yukarıdaki ekran görüntüsünde "TFSERVER" şeklinde Team Foundation Server kurulu olan bilgisayarın network ismi girilmiştir. Buraya sunucunun ip adresi veya bir domain adresi de girilebilmektedir. Team Foundation Server internet veya yerel ağ üzerinden rahatça erişilebilir olması dolayısı ile HTTP protokolü üzerinden çalışan bir iletişim protokolüne sahiptir. Team Foundation Server yüklenen bir sunucu bilgisayarda Internet Informatin Services (IIS) üzerine yeni bir web site oluşturur ve web site içerisinde esas işi yapan ASP.NET web servislerini yerleştirir. Team Foundation Server'in istemcileri ile ara yüzünü oluşturan iletişim kontratı tamamen web servisleri ile oluşturulmuştur. Bu sayede Team Foundation Server için yeni bir istemci yazılması ya da mevcut herhangi bir yazılımın Team Foundation Server ile konuşması kolayca yapılabilmektedir.



Şekil 2.5 TFS üzerindeki takım projeleri

Girilen sunucu bilgisi onaylandıktan sonra Team Foundation Server üzerindeki "Team Project""ler listelenir. Team Project, Team Foundation üzerinde bulundurulan ve bir proje tipine (MSF for CMMI,MSF Agile ya da Scrum gibi) bağlı olarak oluşturulmuş projelerdir. Bir Team Project içinde sadece kaynak kod değil proje dokümanları, iş nesneleri gibi nesnelerde yer almaktadır.

Şekil 2.5'de oluşturmuş olunan projeler gözükmektedir. Team Explorer üzerinde listelenmesini istenilen ve üzerinde çalışılacak projeler bu listeden seçilerek işleme devam edilir.

Şekil 2.6 Team Explorer penceresi

Team Explorer penceresinde seçilen sunucu ve altında ise sunucuda yer alan ve listede yer almasını istediğimiz Team Project'ler listelenir. Seçilen Team Project'den work items, documents, reports, team builds ve source control bölümlerine erişilebilmektedir.

Work Items (İş nesneleri): Projede yer alan work itemlara erişimi sağlayan sorguları içeren team explorer bölümüdür.

Documents (Belgeler): Projenin işleyişi, analiz dokümanları, proje planları gibi proje ile ilgili dokümanların listelendiği bölümdür. Bu düğümde yer alan dokümanların kaynağı Team Foundation Server kurulumu sırasında zorunlu olarak kurulan Windows Sharepoint Services üzerinde barındırılmaktadır. Bu dokümanlara proje portalı aracılığı ile de erişmek mümkündür. Bunun için proje üzerine sağ tıklayıp "Show Project Portal" seçeneğini seçmek yeterli olmaktadır.

Reports (Raporlar): Sql Server Reporting Services üzerinde bulunan proje raporlarına kısayoldan erişim içindir.

Team Builds (Takım Derlemeleri): Team Foundation Server üzerinde oluşturulmuş olan Team Build nesnelerine erişimi sağlar. Team Build ile projelerin build loglarını tutmak, buildleri otomatize etmek gibi işlemler için kullanılmaktadır.

Source Control (Kaynak kontrolü): Kaynak kontrolü seçildiği takdirde Source Control Explorer penceresini açar ve bu pencere ile team project içerisinde yer alan kaynak kodlara erişim sağlar.

2.5. VSTS'de Süreç Yönetimi

VSTS ile birlikte sunulan MSF 4.0, yazılım geliştiricilerin değişen ihtiyaçları göz önünde bulundurularak, "MSF Çevik" ve "MSF CMMI" ve "Çevik Scrum" gibi süreç yaklaşımlarını içermektedir.

MSF Çevik ile çevik yazılım tekniklerini benimseyen profesyoneller ve küçük projeler için değişen ihtiyaçlara daha hızlı cevap verebilmek ve müşteri ile daha yakın bir çalışma ortamının sağlanması hedeflenmiştir.

MSF CMMI ile ise, uzun süreli planlamanın, iletişimin ve ayrıntılı raporlamanın daha önemli olduğu projelerdeki ihtiyaçlar ön plandadır. Sürecin devamlı iyileştirilmesi için bir model sunulmuştur. Sürekli iyileştirme ile yazılım geliştirme adımlarında gerçekleşen döngülerin azaltılması sayesinde, projenin süre ve maliyet hedeflerini yakalaması mümkün olmuştur. CMMI 'ın kullanılmasının en önemli avantajı, diğer yazılım geliştiren firmaların iş yapma kabiliyetlerini karşılaştıran bir standart olmasıdır.

Takımların çalışma biçimlerine uymayan süreçler, önemli bir dirençle karşılaşırlar. Üretkenliğin sürdürülebilmesi ve yeniliklerin uygulanabilmesi için bir denge şarttır. Denge sağlanamaz ise, süreç bir yük olarak gözükmeye başlayacaktır. Projelerde metodolojiyi seçerken proje takımının büyüklüğü, projenin kritikliği ve projedeki öncelikler gibi parametrelere dikkat edilmesi gerekmektedir. Her proje değerlendirildiğinde, projeye özel bazı parametreler (proje ekibindeki kişisel özellikler, kurum özellikleri gibi) ve özel ihtiyaçlar çıkacaktır. Projeler tüm ayrıntılarıyla değerlendirildiğinde, her projede özelleştirme imkânının bulunması ve

uyarlama yapılması çok önemlidir. Özelleştirme, VSTS'nin en önemli özelliklerindendir. VSTS ile sunulan ve yukarıda bahsi geçen süreç şablonlarının, aynen kullanılması mümkün olabileceği gibi, kurum ihtiyaçlarına göre gerekli özelleştirmeler de yapılabilir. Kurum içindeki alışkanlıklar ve tecrübeler yeni çalışma ortamının bir parçası olması sağlanabilir. Çözüm ortakları tarafından geliştirilen ve yaygın olarak kullanılan RUP (Rational Unified Process) gibi süreç şablonlarının da Team system ortamında kullanılması mümkündür.

2.6. VSTS'de Proje Yönetimi ve İş Takibi

VSTS ile takım içinde, iş takibinin sağlandığı birlikte çalışma ortamı sunulmuştur. Proje ile ilgili iş takip nesnelerinin (work item) takım içinde ilgili kişilere atanması ve tamamlanma süreci takip edilebilmektedir.

VSTS içinde, bir proje oluşturulurken, o projenin takip edeceği süreç şablonu (MSF Agile, MSF CMMI, Agile Scrum gibi) da seçilmektedir. Süreç şablonu içinde, iş takip nesneleri ile ilgili tüm bilgiler tanımlanmıştır. Her süreç şablonu, kendine özel iş takip nesnelerini ihtiva etmektedir. Hata, risk, gereksinim, servis kalitesi gibi örnek iş takip nesneleri bulunmaktadır. Bu iş takip nesneleri, o projede kullanılabilmektedir. Kurumlar ihtiyaçlarına göre süreçleri özelleştirdiklerinde veya yeni süreç şablonu geliştirdiklerinde, bu iş takip nesneleri de özelleştirilebilmekte, yeni iş takip nesneleri oluşturulabilmektedir.

İş takip nesnesi içinde aşağıdaki bilgiler yer almaktadır: Tüm iş takip nesnelerindeki ortak olan bilgiler:

- İş tanım nesnesi ile ilgili genel bilgiler (başlık, tarih, atanan kişi, durum)
- Bilgi giriş iş kuralları (o alan için desteklenen değerler gibi)
- İş tanım nesnesinin gösterileceği form tanımı
- Süreç tanımı ile ilgili basit bir model

Tüm iş takip nesnelerinde ortak olan özellikler:

- Ortak sistem alanlarını ihtiva etmeleri
- Yapılan değişikliklerin tarih bazlı tutulması
- Diğer iş tanım nesneleri, dosyalar gibi nesnelerle ilişkilendirme desteği

İş takip nesneleri, sistemdeki diğer bir iş takip nesnesi, kaynak kod veritabanındaki dosya gibi diğer nesneler ile ilişki tanımlanmasını desteklemektedir. "Bu değişiklik niçin yapıldı ?", " Bu değişikliği kim yaptı ?" gibi sorulara cevap bulunabilir. Bu ilişkilerin, sorgulanması ve raporlanması mümkündür. Proje ve süreç yönetimi açısından, izlenebilirliğe önemli katkı sağlanmıştır. Tüm iş takip nesneleri, TFS içinde proje bazlı olarak saklanmaktadır.

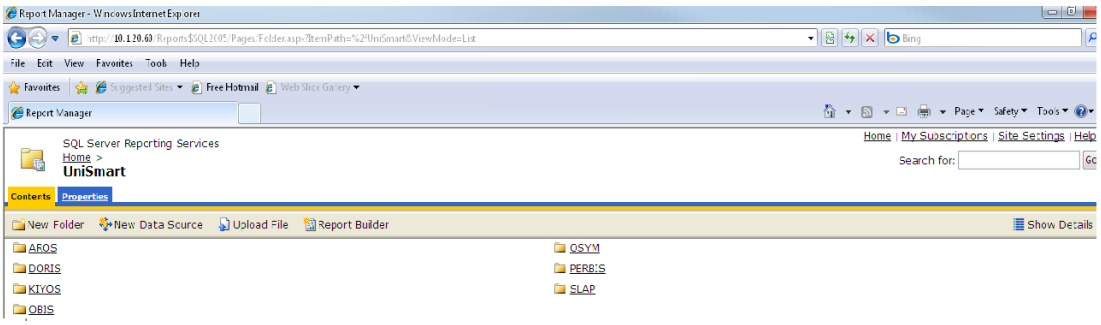
VSTS'nin en önemli özelliklerden birisi, takım elemanlarının kullanmaya alışık oldukları araçları kullanmalarını sağlamasıdır. MS Excel veya MS Project ürünlerini kullanarak iş takibi yapan proje yöneticileri, yine bu şekilde çalışmaya devam edebilirler. Proje yöneticileri için diğer önemli bir husus da, proje hakkında alabilecekleri raporlardır. VSTS ile süreç şablonları ile birlikte hazır raporlar sunulmuştur. MS Excel kullanılarak, VSTS veri ambarına bağlanılarak proje bilgileri analiz edilebilir. Sharepoint sitesi de kullanılarak, proje ile ilgili dokümanlar paylaşılarak takım için iletişim kolaylaştırılmıştır.

2.7. VSTS'de Konfigürasyon Yönetimi

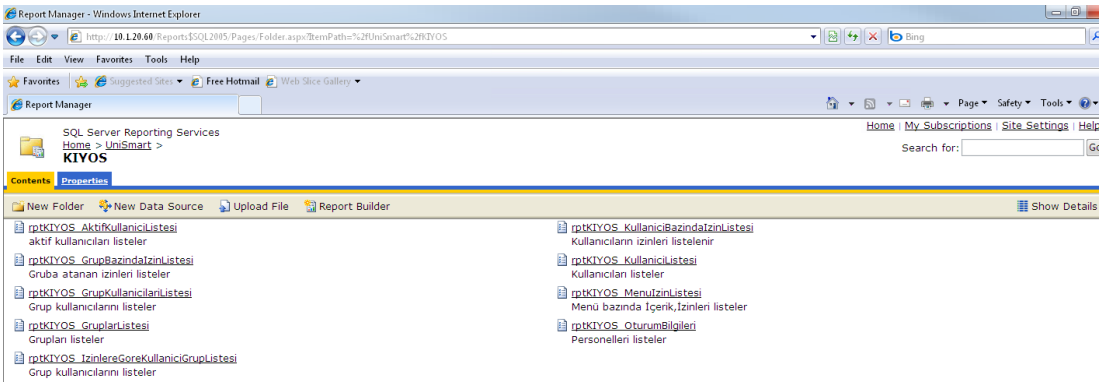
TFS kaynak kod yönetimi, SQL Server 2005 kullanılarak büyük proje ekiplerini destekleme hedefiyle, ölçeklenebilir bir yapıya kavuşturulmuştur. HTTP ve HTTPS desteği ile dağıtık proje ekiplerinin kaynak kod sistemini kullanması sağlanmış, dağıtık ekiplerde ortaya çıkan zaman farklılığı hususuna da çözüm sağlanmıştır. Check-in kurallarının tanımlanması, versiyon yönetimi ve raporlama özellikleri ile bir yazılım konfigürasyon yönetimi çözümünden beklenen özellikleri sağlamıştır.

2.8. VSTS'de Raporlama Altyapısı

VSTS'nin raporlama alt yapısı, SQL raporlama servisleri üzerine kurulmuştur. Süreç şablonları ile birlikte gelen 20'den fazla rapor hazır olarak kullanılabilmektedir. Raporlama servislerinin yönetim portalından veya rapor web uygulama parçası (Report Webpart) ile Sharepoint proje sitesinden izlenebilmektedir. Bu teknoloji alt yapısı sayesinde, yeni raporlar hazırlanabilir veya özelleştirmeler yapılabilir. Aşağıda proje raporlarının, raporlama sunucusundaki yönetim ekranı sunulmuştur :



Şekil 2.7 SQL Server rapor yönetim portalı

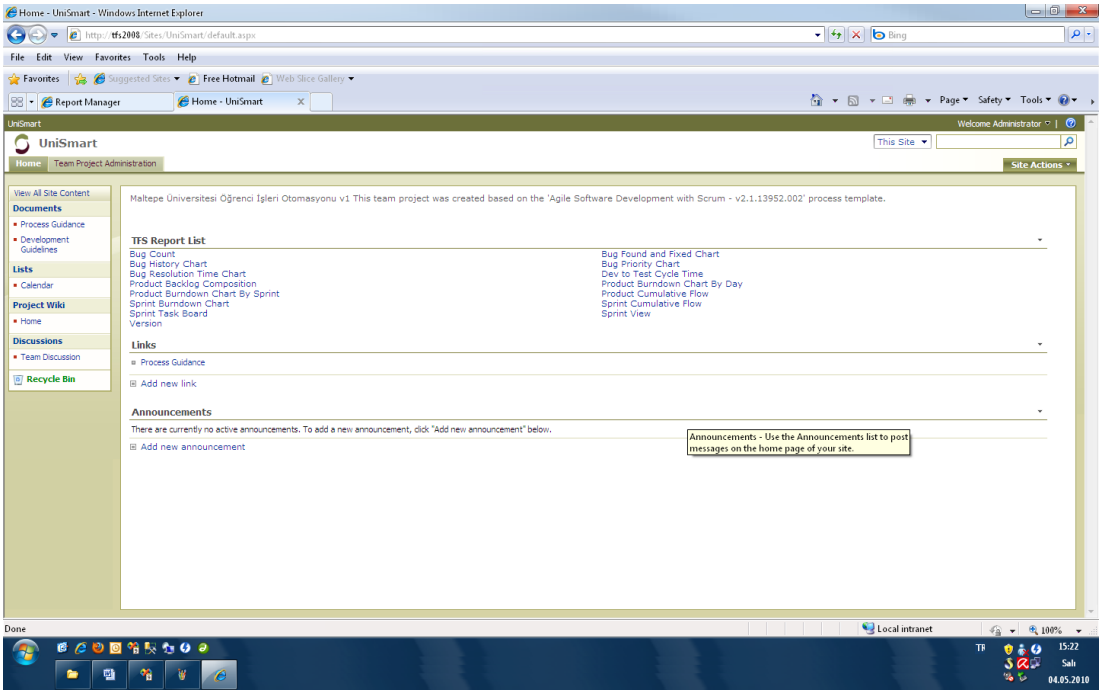


Şekil 2.8 Proje içinde yer alan raporlara örnekler

Uygulama katmanında, Web servisleri yer almaktadır. Windows Sharepoint servisleri (WSS), Visual Studio, MS Excel, MS Project, TFS Müşteri uygulaması gibi müşteri uygulamalarından bu web servisleri çağırılmaktadır. TFS Müşteri uygulaması, takım projelerinin yönetimi için geliştirilmiş yeni bir uygulamadır.

Visual Studio olmadan, bazı proje yönetimi fonksiyonlarını gerçekleştirmek için kullanılabilmektedir. Bu araç ile TFS üzerindeki proje ve süreç bilgilerine daha geniş bir kitle (mesela projeyi takip eden üst yöneticiler gibi) tarafından ulaşımı sağlanmıştır.

VSTS içinde proje oluşturulduğunda, proje sitesi de isteğe bağlı bir şekilde oluşturulmaktadır. VSTS süreç şablonları içinde, Sharepoint sitesinin yapısı ve içeriği de tanımlanmaktadır. Süreç şablonu kurum ihtiyaçlarına göre özelleştirilirken, Sharepoint sitesi de özelleştirilebilmektedir. Sharepoint sitesi de Windows Sharepoint Servisleri tarafından sağlanan portal alt yapısı, doküman yönetimi fonksiyonları ile bilgi paylaşım alt yapısı sunmaktadır. Yukarıda bahsedildiği gibi, Sharepoint sitesinden raporlar izlenebilmektedir. Visual Studio ortamını kullanmayan, proje ile ilgili diğer kullanıcılar (iş analistleri, proje yatırımcıları gibi) bu özelliği kullanarak proje durumu hakkında ayrıntılı bilgi alabilmektedirler. Şekil 2.9'da proje portalı ekran görüntüsü gösterilmektedir.



Şekil 2.9 Proje Portal Arayüzü

3. PROJE ARKA PLANI

Bu tezin konusu, kurum dâhilinde gerçekleştirilen üniversite öğrenci işleri otomasyon sistemidir. Bu sistem, standart öğrenci işleri otomasyonlarına ek olarak hazırlık bölümünün ve enstitülerin iş akışlarını ayrıntılı bir şekilde takip etmektedir.

Üniversite üst yönetimi tarafından sistemin web tabanlı olması, modüler olması ve ilişkisel veri tabanı üzerinde çalışması gerekliliği temel teknik özellikler olarak istenmiştir.

Bu proje, çevik yazılım geliştirme ve web uygulamalarında deneyimli bir proje yöneticisi ile bu konularda deneyimi olmayan iki yazılımcı ve bir analistle başlatıldı. Sonrasında projenin ikinci ayından itibaren ekibe bir yazılımcı daha ilave edildi.

Belirlenen temel gereksinimler üzerinden yazılım ekibinin deneyimleri de göz önünde bulundurularak en uygun mimari belirlendi. Mimari yapı belirlenirken, veritabanı, veri erişim katmanı, iş katmanı, XML web servisi ve web uygulaması olarak beş katmalı bir yapı modellenmiştir. Ayrıca katmanlar kendi içlerinde uygulamanın işlevlerine uygun olarak modüllere ayrılmıştır. Üniversitenin elinde bulunan mevcut lisanslar ve ekibin yetenekleri göz önünde bulundurularak; Microsoft Visual C#, programlama dili, Microsoft SQL Server 2005 ilişkisel veritabanı ve Microsoft Team Suite 2008 yazılım geliştirme ortamı kullanılmıştır.

Projenin ilk haftasında proje yöneticisi tarafından ekibe yoğunlaştırılmış bir SCRUM + XP eğitimi verilerek projeye başlandı. Bununla birlikte diğer çevik metodolojilerinden alınan pratikler de anlatıldı ve nasıl uygulanacağı belirlendi. Ardından ikinci bir haftada da ekibe kullanılacak yazılım geliştirme ortamıyla ilgili eğitim verildi ve örnek uygulamalar yapıldı.

Ekip kısmen hazır olduktan sonra müşteri (üniversite yönetimi) gereksinimleriyle ilgili proje hazırlık safhasına başlandı. Bu safhada müşteri gereksinimleri ana hatlarıyla tespit edildi ve müşteriyle birlikte önceliklendirildi. Ekip, yapılması gereken işler için genel bir zaman kestirimi çıkardı ve projenin birinci versiyonu için kabaca 6 aylık bir geliştirme süresi belirledi. Proje yöneticisi ekiple ve müşteriyle yaptığı toplantılar sonrasında 3'er haftalık iterasyonlarla ilerlemeye karar verdi ve sürüm planlamasının (3 sürüm planlanmıştır) müşteri tarafından onaylanmasının ardından ilk iterasyon planlama toplantısına oturuldu. İterasyonların ilk fazlarında her gün 1 saat ekibin eğitimine ayrıldı. Eğitime ayrılan süre ilerleyen iterasyonlarda haftada 2 saate kadar düşürüldü. 6 ay süreceği öngörülen geliştirme süresi, süreç içerisindeki eklenen ve çıkarılan işlevlerle 7 ay içinde tamamlandı.

4. PROJE SÜRESİNCE KARŞILAŞILAN ZORLUKLAR, ÇÖZÜMLEME GİRİŞİMLERİ VE ÖNERİLER

Projeye başlamadan önce değişik çevik metotlar gözden geçirilmiştir. Projenin büyüklüğü, gereksinimlerin karmaşıklığı, sonuçta ortaya çıkan yazılımın kapsamı ve ekibin yapısı değerlendirilerek uygun görünen pratiklerin belirli bir çerçeve içinde uyarlanmasına (tailoring) karar verilmiştir. Aşağıda seçilen her pratik dört paragrafta incelenmiştir. Buna göre her bir pratiğin, ilk paragrafında pratiğin karakteristiği ve seçilme nedeni, ikinci paragrafında proje içersinde nasıl uygulandığı, üçüncü paragrafında uygulama aşamasında karşılaşılan sıkıntılar ve bu sıkıntıları çözümleme çabaları, son paragrafında ise bu pratiği uygulayacaklar için öneriler anlatılmaktadır.

4.1. Müşteri Memnuniyeti

Müşteri memnuniyeti, çevik metodolojilerin en önemli pratiklerinden biridir. Bu pratikte, proje ekibi yazılım projesinin başlangıcından itibaren düzenli aralıklarla çalışan programlar oluşturarak bunları müşteriye sunar. Müşteri de bu çalışan programlarda gerekli duyduğu değişiklikleri talep eder. Böylelikle yüksek oranda müşteri gereksinimleriyle örtüşen bir yazılım geliştirilmiş olur. Bu pratiği seçerken müşteri gereksinimleri doğrultusunda müşteriyi tatmin eden bir yazılım projesinin oluşturulması amaçlanmıştır.

Ekip üyeleri, müşteri memnuniyetinin önceliğini kayıtsız şartsız kabul etmiş ve atılan her adımda bu önceliğe göre iş yapmıştır. Düzenli aralıklarla çalışan programlar üniversite yönetimine sunulmuş ve üniversite yönetiminden gelen geri beslemelere göre öncelikler belirlenerek, gerekli düzenlemeler yapılmıştır.

Ekip üyelerinin zaman zaman teknik yönleri baskın çıkmış ve müşteriye teknik olarak daha iyisini verme güdüsüyle hareket etme çabaları da olmuştur. Böyle durumlarda proje yöneticisi hem ilgili kişiyi hem de ekibin tamamını 5-10 dakikalık kısa toplantılarla müşteri önceliği konusunda yeniden bilgilendirmiştir.

Yazılım projelerinde esas olan müşteri gereksinimleridir. Bu noktada ekip üyeleri, projenin başında yeterince bilgilendirilmeli ve eğitilmelidir. Proje ekibinin görevi müşterinin ihtiyacını tam olarak karşılayan yazılımı geliştirmektir.

4.2. 5-10 Kişilik Takımlar

Bu pratik SCRUM metodolojisinden alınmıştır. Bu pratikte, proje süresince ekip üyelerinin sürekli olarak birbirleriyle iletişim halinde olmaları gerekmektedir. Bu iletişimde oluşabilecek problemleri ortadan kaldırmak amacıyla ekibin 5-10 kişiden oluşturulması öngörülür. [17] Bu pratiği seçerken ekip üyeleri arasında oluşabilecek iletişim problemlerinin en aza indirgenmesi amaçlanmıştır.

Gerçekleştirilen projede ekip, proje yöneticisi dâhil 4 kişiyle başlamış ardından üçüncü iterasyondan hemen önce ekibe 1 kişi daha ilave edilmiştir.

Ekibe yeni katılan kişi nispeten araçlar ve teknolojiler konusunda deneyimli olduğu için proje yöneticisiyle 1 haftalık bire-bir eğitim ve çalışmayla üçüncü iterasyona aktif olarak dâhil edilebilmiştir.

Proje süresince ekibe yeni bir üye katılacaksa iterasyon süreleri dikkate alınmalıdır. İterasyon geçişlerinde ekibe katılacak olan kişinin, araçlar ve teknolojiler konusundaki deneyimi göz önünde bulundurularak eğitim veya adaptasyon süresi belirlenmelidir.

4.3. İterasyonlu / Artırımlı Süreçlerle Geliştirme

Tüm çevik metodolojilerde yer alan bu pratikte, her bir iterasyon 2-4 hafta arasında icra edilir. [26] Bir iterasyon sonunda üzerine yeni özellikler eklenen ürün, müşteriye teslim edilir ve geri bildirimler alınır. Bu pratikle iterasyonlu süreçlerle yazılım geliştirerek müşteri gereksinimleriyle örtüşen bir yazılımın geliştirilmesi amaçlanmıştır.

Müşteri ve ekip üyeleri arasında yapılan görüşmeler sonucunda, proje için en uygun iterasyonun 3 hafta olacağına karar verilmiştir. İstisna olarak, ilk sürümün birinci sprinti ve son sürümün son sprinti için iterasyon süresinin 4 hafta olması kararlaştırılmıştır. Tüm tarafların onay ve taahhütleri alınmıştır.

Uygulama aşamasında bu pratikle ilgili yaşanan en önemli sorun, önceki iterasyonlarda çıkan yazılım hatalarını düzeltmek için aktif iterasyon içinde zaman ayırma gerekliliği olmuştur. Başlangıçta iterasyon planı içinde yer almayan bu unsur, yazılım geliştirme sürecinin ortalarına doğru iterasyona ayrı bir iş kalemi olarak eklenmiş ve gerekli zaman ayrılarak, iterasyon planının daha doğru şekilde oluşturulması sağlanmıştır.

Bir yazılım projesinde iterasyon süresi proje kapsamına, müşteri gereksinimlerine ve daha da önemlisi iş ürünlerinin büyüklüğüne bağlıdır. Bu faktörler göz önünde bulundurularak anlamlı bir artırım sağlayacak bir süre belirlenmelidir.

4.4. Küçük ve Kısa Aralıklı Sürümler (Release)

Bu pratik XP ve SCRUM metodolojilerinden alınmıştır. Bu pratikle proje, birbirinden ayrı zaman aralıklarına (2-4 hafta) bölünür. Her bir zaman aralığında yapılacak işin kendine ait son teslim tarihi vardır. Belirlenen son teslim tarihini aşmayarak iş bitirilir ve müşteriye teslim edilir. [12,17] Bu pratikle müşterinin,

projenin ilerleyişini her bir iterasyon sonrasında özellikleri artan, yaşayan bir uygulamayla takip edebilmesi amaçlanmıştır.

Proje sürümlerinden ilki 3, ikincisi ve üçüncüsü ise 2 sprintten oluşan üç sürüm planlanmıştır.

Bu pratik uygulanırken herhangi bir sorunla karşılaşılmamıştır.

Sürüm sayısı planlanırken müşteri gereksinimleri ve zaman kısıdı göz önünde bulundurulmalıdır.

4.5. Kısa Süreli Proje

Bu pratik SCRUM metodolojisinden alınmıştır. Bu pratikle proje süresinin 1 yılı geçmemesi hedeflenir. [17]

Projenin ilk tahminlerde 6 ay süreceği öngörülmüş, ancak süreç içerisindeki eklenen ve çıkarılan işlevlerin ardından 7 ay içinde tamamlanmıştır.

Üniversite yönetimi yazılım geliştirme sürecinde, yeni işlevler eklenmesine ve proje kapsamı dâhilindeki bazı işlevlerin çıkarılmasına karar vermiştir. Bu kararlar proje süresinin değişmesine neden olduğundan üniversite yönetiminin onayıyla proje süresi uzatılmıştır.

Projenin geliştirilmesi esnasında isterler değişmeyecek büyüklükte olmalıdır. Böylece başarıyla sonuçlandırılan kısa süreli projeler hem yönetimin, hem kullanıcıların hem de yazılım ekibinin motivasyonunu ve inancını artırır. Aynı zamanda ortaya çalışan bir uygulama çıkar ve hedeflenen büyük projenin bir kısmı bile olsa, hayata geçmiş olur.

4.6. Genel Bir Model/Mimari Geliştirme

Bu pratik DSDM metodolojisinden alınmıştır. İsterler ve kısıtlar ana hatlarıyla belli olduktan sonra projenin tamamı için genel bir mimari belirlenir. Bu mimari daha sonra her bir iterasyon safhasında ayrıntılandırılır. Bu pratiği seçmemizin nedeni, projenin genel yapı ve kapsamını önceden görebilmektir.

Uygulamanın tamamı için beş katmanlı ve her bir katman içinde işlevlere bağlı modüllere ayrılmış bir genel mimari oluşturulmuştur. Katmanlar sırasıyla en arka uçta ilişkisel veritabanı ve veritabanı programlama nesneleri, ardından veritabanı erişim katmanı, iş katmanı, XML Web Servisi ve Web Uygulaması şeklinde yapılandırılmıştır. Genel çözüm modeli ve mimari teknoloji seçimiyle birlikte müşteriye sunularak onay alınmıştır. Ardından mimari ve kullanılacak teknolojiler ekibe tanıtılmış ve tartışılmıştır.

Genel bir mimari model geliştirirken karşılaşılan en önemli sıkıntı, ekibin çok katmanlı mimari modelle ve servis temelli yaklaşımla çalışmamış olmasıydı. Ancak modülerliği ve müşteri isterlerini karşılamak amacıyla mimari model, ekibin yeteneklerinden ziyade projenin gereksinimlerine en uygun şekilde tasarlanmıştır. Ekibin mimariye uyum sağlaması için yoğun bir eğitim ve örnek uygulama programı hazırlanmış ve uygulanmıştır.

Müşteri gereksinimleri ve kısıtları ana hatlarıyla belli olduğunda, projenin geneli için bir mimari ve altyapı tasarımı oluşturmak projenin başarımına büyük oranda katkı sağlamaktadır. Bu noktada müşterinin sahip olduğu donanım, lisans vb. faktörlerin göz önünde bulundurulmasında fayda vardır.

4.7. Teknolojiyi Gereksinimler Belirler

Bu pratik Lean metodolojisinden alınmıştır. Öncelikli olarak uygulama sahasına yönelik gereksinimler belirlenir. Daha sonra bu gereksinimlere uygun teknolojiler

seçilir. [1] Bu pratik, müşterinin istediği projenin, bütçesi dâhilinde gerçekleştirilebilmesi için önemlidir.

Bir önceki pratikte belirtilen genel modeli geliştirebilmek için müşterinin sahip olduğu lisanslar, kullanıcı alışkanlıkları, ekip deneyimi gibi kısıtlar göz önünde bulundurularak teknoloji tercihleri belirlenmiştir. Mevcut durum gereği yazılım geliştirme için Microsoft Visual Studio Team Suite, ilişkisel veritabanı olarak Microsoft SQL Server 2005 kullanımı tercih edilmiştir. Uygulamanın Web sunucusu katmanı Asp.Net ile programlanmış, iş ve veri erişim katmanları SOA (XML Web Servisleri) ile servis edilmiştir.

Yazılımın analiz aşamasında uyguladığımız bu süreçte karşılaştığımız sorun, müşterinin uygulamanın tüm arayüzlerinin Web tabanlı olmasını istemekle birlikte, daha önceden alışık olunan masa üstü yazılım seviyesinde bir performans beklemesiydi. Sorunu çözümlemek için müşteri bu iki teknoloji hakkında ayrıntılı olarak bilgilendirildi ve her bir sayfa için en yüksek bekleme süresi olarak özel sayfalar ve raporlar hariç 4 saniyede karar kılındı.

Teknoloji belirlenmeden önce müşteri gereksinimleri ve kısıtları ana hatlarıyla belirlenmeli ve genel mimariye en uygun teknolojiler seçilerek müşterinin beklentilerinin bu şekilde ne kadar karşılanabileceği anlatılmalı, sonra da müşterinin onayı alınmalıdır. Teknoloji seçimi yaparken göz önünde bulundurulması gereken bir diğer faktör de ekibin bilgi düzeyi ve deneyimidir.

4.8. Sürekli Günlük Entegrasyon

Bu pratik XP ve MSF metodolojilerinden alınmıştır. Yazılım geliştirilirken yapılan sistem değişiklikleri ve yeni bileşenler hemen sisteme entegre edilerek günlük derlemelerle test edilir. Sürekli entegrasyon sayesinde yazılımcıların sistem üzerinde yapılan değişiklikleri görmeleri ve oluşabilecek hataları erken tespit edebilmeleri sağlanır. [12,14,27]

Uygulama geliştirme için tüm ekip üyelerine Micsoroft Visual Studio Team Suite kurulmuştur. Kodları ortak bir noktadan yönetmek ve eş zamanlı ekip çalışmasını kontrol edebilmek için de Microsoft Team Foundation Server kullanılmıştır.

Kullanılan yazılım geliştirme ortamı (Microsoft Visual Studio Team Suite ve Microsoft Team Foundation Server), kodları tek bir yerde tutma ve günlük otomatik olarak derleme özelliklerine sahip olduğu için bu pratiğin uygulanmasında teknik herhangi bir sıkıntıyla karşılaşılmamıştır. Ancak araçların kullanımına yabancı olan ekibin eğitilmesi gerekmiştir.

Teknoloji seçiminde bütçe kısıtlarının el verdiği ölçüde otomatik entegrasyon sağlayan ürünler tercih edilmelidir.

4.9. Kodlama Standartları

Bu pratik XP metodolojisinden alınmıştır. Ekip üyeleri önceden tanımlanmış kodlama standartlarına göre yazılımı geliştirirler. [12] Bu pratiği seçerken, yazılan kodun karmaşıklığını azaltarak, bütün ekip üyeleri tarafından kolaylıkla anlaşılabilmesi amaçlanmıştır.

Proje başlamadan önce ekiple beraber kodlama, isimlendirme ve kod yorumları yazımı vb. üzerine standartlar belirlenip, belgelenmiştir. Proje boyunca ekip üyelerinin bu standartlara uyması istenmiştir.

Ekibin kodlama standartlarına başlangıçta dikkatlice uymasına rağmen, zamanın daralmasından dolayı bazı noktalarda standartlara uymadığı gözlenmiştir. Bu noktada, projenin önceliği belirlenen işlevleri zamanında ve çalışır halde teslim etmek olduğu için bu sapmalar göz ardı edilmiştir.

Proje geliştirme süresince ekibin mutlaka bir kodlama standardına uyması gereklidir.

4.10. Planlama Oyunu

Bu pratik XP metodolojisinden alınmıştır. Planlama oyunu, müşterinin belirlediği her bir iterasyon için yazılım ekibinin müşterinin de içinde bulunduğu bir toplantıda, o işin ne kadar zamanda yapılacağıyla ilgili kestirimde bulunmasıdır. Böylece müşteri bir sonraki iterasyonda hangi işlerin yapılacağına karar verecek ve yazılım ekibi de belirttiği sürelere mümkün olduğunca bağlı kalarak, bu işleri ilgili iterasyonda yapacaktır. [12]

Uygulamada çoğunlukla proje yöneticisi önce geliştirilecek modülle ilgili yöneticiyle toplantı yaptı, sonra toplantıdan elde edilen bilgiler doğrultusunda ekiple birlikte ürün listesi oluşturuldu.

Müşteriyle birlikte bir sonraki iterasyonda yapılacak işlerin belirlendiği bu süreçte ekibin görevi en doğru kestirimleri yapmaktı. Planlama oyununda karşılaşılan en büyük iki sıkıntı ekibin doğru kestirimler konusunda yetersiz oluşu ve müşterinin gereksinimlerini aktarmada iş süreçleri yazılı ve net olmadığı için tam bilgi aktaramayışıydı. Kestirim hataları özellikle ilk iterasyonlarda planlanandan daha az product backlog tamamlanmasıyla sonuçlandı. Eksikliğin tamamlanması için ve daha sonradan bildirilen modül güncellemeleri için fazla mesai yapılmak zorunda kalındı.

Doğru kestirim tecrübeyle doğrudan ilişkili bir özelliktir. Planlama oyununda başarı için tecrübeli eleman faktörü çok önemlidir. Ekibin mümkün olduğunca, yapılacak işi ne kadar zamanda yapabileceğini bilen elemanlardan oluşturulması gerekmektedir. Ayrıca gereksinimlerin zamanında ve tam olarak alınabilmesi için, yazılım sürecinin başlamasından önce tam kapsamlı bir iş analizinin yapılması, yazılımın süresini kısaltacak ve kalitesini artıracaktır.

4.11. İterasyon İçerisinde Takım Dokunulmazdır

Bu pratik SCRUM metodolojisinden alınmıştır. Bu pratiği almamızın nedeni, belirli bir modüle konsantre olmuş takımın dikkatini dağıtmadan uygulama geliştirebilmesini sağlamak istememizdir. Bu özellik hem motivasyon hem de yazılımın kalitesi için önem arz etmektedir.

Pratiği uygulanma aşamasında proje yöneticisi müşteriyle gerekli görüşmeleri yapıp, sonraki sprint için gerekli bilgileri toplarken, gereksinimlerin devam eden sürece sızmasını engelleme yönünde çaba göstermiştir.

Müşteri tarafından genellikle bu pratiğe uyulmuştur. Ancak müşterinin tam olarak ne istediğini bilmemesinden dolayı, gereksinimleri hatalı belirtmesiyle ilgili olarak yaşanan bir istisna, yeni bir iterasyona başladıktan sonra, önceki iterasyonlarla ilgili kritik hata geri bildirimlerinin bir an önce çözümlenmesi gereksinimi olmuştur. Bu durumla ilgili alınan iki önlem; daha az hatalı kodlamaya teşvik ve iterasyonlarda %20'lik bir zaman dilimini hata ayıklamaya ayırmak şeklinde gerçekleştirilmiştir.

İterasyon gerçekleştirilirken ek gereksinimler talep edilirse, bu gereksinimler kesinlikle iterasyona dâhil edilmeyip bir sonraki iterasyona bırakılmalıdır. İterasyon gerçekleştirilirken istisnai durumlar dışında ekipten üye çıkarılmamalıdır ya da eklenmemelidir. Ayrıca bir önceki iterasyonda hata çıkma olasılığı göz önünde bulundurularak hata ayıklamaya makul bir zaman ayrılmalıdır. Bu zamanı en aza indirebilmek için kod gözden geçirmeleri ve mümkünse eşli programlama yapılmalıdır.

4.12. Müşteriyle Birlikte Geliştirme

Bu pratik XP, MSF ve DSDM metodolojilerinden alınmıştır. Çevik projelerde müşteri ile proje ekibinin beraber çalışması kaçınılmazdır. Programdan istenilenleri en iyi müşteri bileceği için, sürekli müşteriden geri bildirim alınması gerekmektedir. Bunun en kolay yolu müşteri ile proje ekibinin beraber çalışmasıdır. [17] Müşteri ile

proje ekibi arasındaki etkileşimi sağlayan ürün yöneticisidir. Ürün yöneticisi müşteri tarafında çalışan, müşterinin isteklerini anlayan ve bunu proje ekibine anlatan kişidir. Bu sayede projenin müşteriyle birlikte geliştirilmesi, gereksinimlerin doğru şekilde anlaşılması ve uygulanması sağlanmış olur.

Pratiğin uygulanması için sık sık geliştirilecek modüllerle ilgili kişiler ekiple toplantıya davet edilmiştir. Başlangıçta birkaç toplantı pratiğe uygun yapılabilse de, sonraki toplantılar da pratiği aslına uygun şekilde işletmek mümkün olmamıştır.

Proje boyunca yaşanan sıkıntılardan biri de müşterinin aktif olarak takımla birlikte olamayışıydı. Ancak müşteri iterasyon planlamada ekibe dâhil olmuş, sonrasında iterasyon boyunca ya süreçteki sorulara cevap vermiş ya da cevap verebilecek kişilere yönlendirmiştir.

Müşterinin ekiple birlikte çalışmaması bazı noktalarda yavaşlamaya neden olabilir. Bu durumun önüne geçebilmek için projenin başında müşterinin ikna edilmesi ve sürekli ekiple çalışabilecek yetkin birinin atanmasının sağlanması önemlidir.

4.13. Açık İletişim ve Ortak Dil

Bu pratik MSF ve DSDM metodolojilerinden alınmıştır. Bu pratikte projenin başarılı olabilmesi için ekip üyeleri arasında açık ve samimi bir iletişimin olması gerekmektedir. Bu pratiği seçerken, ekip üyelerinin projeyle ilgili endişelerini, yetersiz ve deneyimsiz olduğu alanlarla ilgili düşüncelerini diğer ekip üyeleriyle paylaşabilmesi amaçlanmıştır. Ekipteki iletişim eksikliği proje işlevlerinde eksikliklere yol açabilir. Bu problemin önüne geçilmesi için tüm ekibin aynı jargonu konuşması gerekir.

Ekip her toplantıda ve her fırsatta açık olmaya ve sürekli iletişime teşvik edilmiştir. Projenin başlarından ortalarına kadar ekip üyeleri arasında açık iletişim yeterince yapılamamıştır.

Ekipteki kişilerin karakteristik özellikleri ve çevik metodolojiyle ilk defa tanışmaları nedeniyle en çok sıkıntı yaşanan pratik bu olmuştur. Ekip üyeleri arasındaki iletişim problemlerinin büyük bir kısmını yanlış anlaşılmalar oluşturmuştur. Bu yanlış anlaşılmalar belirli bir oranda düzeltilene kadar yazılı iletişim yolu kullanılmak zorunda kalınmıştır. Bir süre, ekip üyelerinin hem iş yapış alışkanlığı kazanması hem de iletişimdeki yanlış anlaşılma ile proje işlevlerinde oluşabilecek eksikliklerin önlenmesi için yazılı iletişim yolu kullanılmıştır.

Projede ekip üyelerinin karakteristik özelliklerinden kaynaklanan yanlış anlaşılmalar varsa; ya bu projede uygulandığı gibi yazılı iletişim seçilebilir ya da ekip üyesine bir iş verildiğinde bu işin doğru anlaşıldığını kontrol etmek için proje yöneticisi geribildirim isteyebilir.

4.14. Ortak Vizyon

Bu pratik MSF metodolojisinden alınmıştır. Ortak vizyon, çözümü elde etmek için yapılması gerekenleri açık bir şekilde anlamaktır. [28] Bu pratikle hem ekip üyelerinin hem de müşterinin geliştirilecek olan yazılımın işleyişi hakkında hemfikir olması amaçlanmıştır.

Projeyle ilgili müşteri tarafından belirlenen vizyon ekip tarafından tamamıyla sahiplenilmiştir.

Bu pratik ile olarak proje süresince hiçbir sorun yaşanmamıştır.

Ekip, proje vizyonunu mutlaka benimsemelidir. Seçilen vizyon, proje sonucunda ne geliştirilecek, ne işe yarayacak ve nasıl geliştirilecek gibi üç soruya cevap verebilmelidir.

4.15. Ortak Kod Sahiplenme

Bu pratik XP metodolojisinden alınmıştır. Geliştirilen yazılım kodu, bütün ekip üyelerinin ortak malıdır. Bu pratikle bir ekip üyesinin belirli kurallar çerçevesinde herhangi bir ekip üyesinin ürettiği kodlara erişerek, mevcut yazılımı daha iyiye götürme adına değişiklikler yapabilme imkânına sahip olmaları amaçlanmıştır.

Ekip üyelerine katmanlı mimaride yazılan projenin her katmanında dönüşümlü olarak görevler verilerek projeye hakim olması sağlanmıştır.

Başlangıçta ekibin ortak kod sahiplenmeyi kabullenmesine rağmen süreç içinde çeşitli anlaşmazlıklar yaşanmış, proje yöneticisinin müdahaleleriyle bu anlaşmazlıklar giderilmiştir.

Ekip üyelerinin her biri, kodun herhangi bir yerini gerektiğinde değiştirebilmelidir. Ekip içinde tüm kararlar ve tasarımlar birlikte ele alındığı için, herkes birbirinin kodunu sahiplenmelidir.

4.16. Kendi-Kendini Düzenleyen, Yönlendiren, Motivasyonu Yüksek Takım

Bu pratik SCRUM metodolojisinden alınmıştır. Yazılım projesini gerçekleştiren ekip üyelerinin farklı kişilik ve iş yeteneklerine sahip kişilerden oluşması doğaldır. Ekip, motivasyonu ve iletişim becerisi yüksek, olumlu düşünce yapısına sahip, disiplinli ve kararlı kişilerden oluşturulmalıdır.

Ekip üyelerinin motivasyonunu yüksek tutmak ve kendi öz motivasyonları ile çalışmalarını sağlamak için, kodlama ayrıntılarına müdahale edilmemiş ve yapacakları işleri (alt modülleri) kendilerinin belirlemesi sağlanmıştır. Aynı zamanda ekip içinde yardımlaşmanın gerekliliği ve önemi sürekli olarak vurgulanmış ve cesaretlendirilmiştir.

Ekip daha önceki bireysel uygulama geliştirme tecrübelerinden dolayı motivasyonu yüksek elemanlardan oluşmaktaydı. Ancak ekip olarak çalışma konusunda zaman zaman sıkıntılar yaşandı. Bu tip sorunlar proje yöneticisinin motive edici ve yönlendirici yaklaşımlarıyla çözülmeye çalışıldı.

Ekip ruhu oluşturmak ve motivasyonu yüksek tutmak için sosyal faaliyetler düzenlenmeli, ekip içindeki uygunsuzluklar ve çatışmalar olabildiğince erken fark edilerek çözümlenmelidir.

4.17. Risk Yönetimi

Bu pratik MSF metodolojisinden alınmıştır. Projenin başından sonuna kadar karşımıza çıkabilecek, gerek projenin iptal olmasını gerekse de projenin yavaşlamasına yol açacak, bütün riskler belirlenir ve bu riskler önem sırasına göre sıralanır. Risk yönetim planı hazırlanarak, proje süresince oluşabilecek olağan dışı durumlara karşı önlemler alınmış olunur. [29]

Projenin hazırlık safhasında muhtemel riskler tespit edildi. Bu risklerden bazıları müşteri onayı alınarak proje kapsamı dışında tutuldu. Geri kalan riskler içinse önleyici tedbirler almak yerine risklerin gerçekleşmesi göze alındı. Bunun nedeni ise gerçekleşebilecek olan risk maliyetinin, önleyici tedbir maliyetinden daha düşük olacağının tespit edilmesiydi.

Bu pratikle ilgili karşılaşılan en önemli zorluk, projenin isterlerini her bir modül için ayrı biri hatta bazen birden fazla kişiyle çalışılması nedeniyle riskleri toplu olarak ön görmenin sorumluluğunun tamamen proje yöneticisi üzerinde olmasıydı. Bu noktada proje yöneticisi önceki yazılım projesi deneyimlerini de kullanarak olası tüm riskleri masaya yatırdı.

Proje hazırlık safhasında mutlaka riskler tespit edilmelidir. Tespit edilen risklerin büyüklükleri ve etkileri belirlenip, bu risklere müşterinin onayıyla uygun planlar hazırlanmalıdır. Risklerin önceliklerine karar verilip önleyici tedbirler alınmalıdır.

4.18. Günlük Toplantılar

Bu pratik SCRUM metodolojisinden alınmıştır. İterasyonun başında belirlenen yer ve saatte, günlük olarak 15 dakikalık ayaküstü toplantılar yapılır. Toplantılarda bir önceki gün neler yapıldığı, o gün içinde neler yapılacağı ve ekibin varsa ihtiyaçları ve karşılaştıkları güçlükler konuşulur. [17] Bu pratikle ekip üyelerinin birbirleriyle eş zamanlı çalışması ve sorunların çok hızlı bir şekilde çözülmesi sağlanır.

Proje boyunca her sabah saat 9.00-9.15 arasında günlük toplantılar yapıldı. Bu toplantılarda genel olarak bir önceki gün yapılanlar ve o gün yapılacaklar konuşuldu. Ayrıca ekibin varsa karşılaştığı zorluklar ve sıkıntılar tartışıldı. Proje yöneticisi bu sıkıntılarla ilgili olarak çözüm önerileri getirerek gerekli yönlendirmeleri ve takviyeleri yaptı.

Toplantılar başlangıçta SCRUM pratiklerine uygun olarak ayakta yapılsa da ekip buna pek alışamadı. Bir süre sonra ekibin yerleşiminin de uygunluğu nedeniyle toplantılar oturularak yapılmaya başlandı.

Günlük toplantıların düzenli yapılması ve belirli bir süreyi aşmaması önemle tavsiye edilir. Bizim tecrübelerimize göre, pratiğinde önerdiği gibi, bu toplantılar için 15 dakika uygun bir süredir. Bu toplantılarda ekip özellikle geliştirme süreciyle ilgili sıkıntılarını paylaşma yönünde cesaretlendirilmelidir. Ekibin tamamen toplantıya odaklanması için eğer imkân varsa toplantının çalışma ortamı dışında olması tavsiye edilir.

4.19. Kabul Testi

Kabul testi kullanıcıdan alınan gereksinimlerin “backlog” olarak tanımlanma safhasında, işin gerçekleşme kriterini ve bunun nasıl test edileceğinin tanımlanmasıyla başlar. Ardından yazılım ekibi tarafından geliştirilen işlev müşteri tarafından test edilir, varsa eksiklikler söylenir veya işlev gerçekleştirilmiş olarak

kabul edilir. Müşterinin istediğini elde edip etmediğini ölçmek için en etkili yollardan biridir.

Proje boyunca kabul testleri oluşturulmuş, bu testler hem ekip içinde alfa testlerinde hem de müşteri tarafından kabul testlerinde kullanılmıştır.

Bu konuda yaşanan en büyük sıkıntı, müşterinin analiz aşamasında kabul testlerini yeterince tarif edememesinden kaynaklanmıştır. Bu nedenle test aşamasında geliştirilen işlevlerle ilgili bazı yetersizlikler olduğu ileri sürülmüş, ancak bunlar başlangıçta kabul testleri tanımında yer almadığı için ekip tarafından ek talep olarak değerlendirilmiştir.

Kabul testleri, proje ve ekip başarısının müşteri tarafından onaylandığı en önemli işlemdir. Bu yüzden projenin başında bu testler müşteri tarafından mümkün olduğunca ayrıntılı tanımlanmalı ve proje sürecindeki değişiklikler mutlaka kayıt altına alınmalıdır. Ayrıca kabul testi yapıldıktan sonra müşteri "tamamdır" dediğinde testi yapanın ve mümkünse bir yetkilinin imzası alınmalıdır.

4.20. Önce Test

Bu pratik XP metodolojisinden alınmıştır. Programın kodlanmasına başlamadan önce, birim testleri için test kodları yazılır. [17] Bu pratik geliştirilen programın kalitesini yüksek tutmak amacıyla alınmıştır.

Projede kullanılan yazılım geliştirme ortamı, birim testlerini, bütünleşik yazılım geliştirmeyi ve derlemeyi desteklediği için bu pratik önce testi kodla, sonra işlevi geliştir mantığıyla kodlanmıştır.

Ancak projenin sonlarına doğru zaman darlığı nedeniyle hata çıkma riski göze alınarak birim testinden çok, işlev geliştirmeye zaman ayrılmıştır. Kalitenin

korunması için yazılan kodlar gözden geçirmeyle doğrulanmaya çalışılmış ve kalitenin korunması için çaba gösterilmiştir.

Sonraki iterasyonlarda hatalarla uğraşmamak ve kaynakların daha verimli bir şekilde kullanılmasını sağlamak amacıyla birim testlerinin mümkün olduğunca yapılmasında fayda vardır.

4.21. Basit Tasarım

Bu pratik XP metodolojisinden alınmıştır. Bu pratikle, müşterilerin gereksinimlerini karşılayacak en basit tasarım gerçekleştirilir. Böylece kısa sürede anlaşılması, yönetilmesi ve değiştirilebilmesi kolay bir yazılım geliştirilmesi sağlanır. [12,17]

Genel mimari üzerine yapılacak olan her bir işlev için, müşteri beklentisini karşılayan minimum tasarımla ilerlenmeye çalışılmıştır.

Bu pratik çoğunlukla ekip tarafından uygulansa da bazen ileriye doğru müşterinin değişebilecek gereksinimleri göz önünde bulundurularak kodlama eğilimi gösterilmiştir. Böyle durumlarda proje yöneticisi devreye girerek sonraki versiyonlar için konuyla ilgili not alıp, daha basit bir tasarımla ilerlenmesini istemiştir.

Ekip içinde mümkün olduğunca basit tasarıma önem verilmeli ve müşterinin beklentisinden daha fazlası için kaynak harcanmamalıdır.

4.22. Zamanında Teslim

Bu pratik DSDM ve SCRUM metodolojilerinden alınmıştır. Ekip üyeleri her iki veya dört haftada bir müşteriye çalışan bir program teslim eder. Ürünün teslim zamanı değiştirilemez fakat zamanında teslim etmek için teslim edilecek işlev/iş sayısı azaltılabilir. [17]

Proje boyunca tüm çalışma zaman kutusuyla (time boxing) sınırlandırılmıştır. Böylece her bir kabul testi belirlenen zamanlarda gerçekleştirilmiştir.

Proje ekibinin kestirim hataları ya da önceki iterasyonlardan gelen hataların çözümlemeleri nedeniyle tamamlanamayan işlevler olmuştur. Bu işlevler sonraki iterasyonlara devredilmiştir.

İterasyonlar kesinlikle uzatılmamalı ve tamamlanmayan işler bir sonraki iterasyona bırakılmalıdır. Zamanlama kutusuna mutlaka sağdık kalınmalıdır.

4.23. Sürekli Gelişim

Bu pratik MSF metodolojisinden alınmıştır. Ekip üyeleri her zaman değişime ve yeniliğe açık olarak, ortaya çıkan hatalardan ders alırlar ve aynı zamanda aralarındaki bilgi paylaşımıyla da kendilerini sürekli geliştirirler. Böylece her geçen gün daha da çok şey öğrenen ve kendini geliştiren birey ekibe ve dolayısıyla da projeye daha fazla katkıda bulunur.

Proje süresince hem proje yöneticisinin belirlediği konulardaki eğitimler hem de günlük toplantılarda paylaşılan güçlü ve zayıf yönler üzerine sürekli iyileştirmeler yapılmıştır. Bir taraftan ekip üyeleri bireysel olarak programlamada ve çevik metodolojilerde ilerleme sağlarken diğer taraftan ekip olarak çalışma konusunda da gelişme sağlanmıştır.

Bu konuda yaşanan tek sıkıntı ekip üyelerinin bilmedikleri konularda bilgi eksikliklerini kabul etmemeleri ve gizlemeleri olmuştur. Proje yöneticisi böyle bir durumu fark ettiğinde ilgili kişiye eksiğini fark etmesini sağlayacak bir görev vererek, az da olsa zaman kaybını göze alarak ekip üyesinin eksiğini fark etmesini ve tamamlayıcı aksiyonlar almasını sağlamaya çalışmıştır.

Ekip üyeleri sürekli öğrenme konusunda açık olmalı, eksik ya da hatalarını yeni bir şey öğrenebilme konusunda fırsat olarak görmelidir. Ekip kurulurken bu faktörlere dikkat edilmelidir.

4.24. Az da Olsa Belgele

Bu pratik Lean metodolojisinden alınmıştır. Çevik programlamada az da olsa belgeleme vardır. Kapsamlı belgelemeyle uğraşıp zaman kaybetmek yerine amaca uygun olarak sadece gerekli görülen yerlerde belgeleme yapılır. [30] Gerektiğinde raporlama amacıyla kullanmak ve ekip içinde isterler ve projenin akışıyla ilgili gerekli bilgiyi sağlamak amacıyla belge oluşturulmuştur.

Çevik metodolojilerin doğası gereği ayrıntılı belgelendirme yapılmamış, ancak hem mimari hem kodlama tarafında mutlaka yeterince belgeleme yapılmıştır. Özellikle kodlama standartlarıyla getirilen kurallardan kod açıklama satırları, yazılan kodla ilgili gerekli belgelemeyi sağlamaktadır. Aynı şekilde veritabanı tarafında da hem yerleşik yordamlarda hem de tablolarda gerekli açıklamalara yer verilmiştir. Ayrıca kullanılan çevik proje yönetim aracı (VSTS) ile gereksinim dokümanları, backloglar ayrıntılı bir şekilde belgelendirilmiştir.

Belgelemeyle ilgili karşılaşılan zorluk, üniversitedeki birimleri yazılı iletişimden çok işleri sözlü iletişimle yapma yönündeki ısrarı olmuştur. Bu zorluğu çözmek için gerekli bilgiler ve bildirimler kısa e-postalar aracılığıyla yapılmıştır.

Projenin başından itibaren tanımlanmış mimari, kodlama standartları ve kabul testleriyle ilgili mutlaka belgeleme yapılması gerekmektedir.

4.25. İş İhtiyaçlarına Odaklanma

DSDM metodolojisinden alınmış bu pratik, daha geç teslim edilecek mükemmel bir yazılım yerine önceden teslim edilmiş iyi bir iş ihtiyacı çözümüne odaklanmayı

öngörür. Bu da kritik iş ihtiyaçlarına odaklanmayı gerektirir. Bu noktada her iterasyon öncesi müşterinin yapılacak işler için önceliklerini belirlemesi gerekir. [31]

Müşterinin belirlediği önceliklendirmeye uyularak kritik gereksinimlere odaklanılmıştır. Genel ihtiyaçlarda da gelecekte yapılma ihtimali olan işleyiş ve işlemlerden çok, şu an gerçekleşen iş süreçlerine ve bu ihtiyaçlara cevap verebilecek işlevler geliştirmeye odaklanılmıştır.

Bu pratikte karşılaşılan zorluk, her projede olduğu gibi müşterinin önceliklerinin beklenmedik durumlar nedeniyle değişmesi yönünde olmuştur. Müşterinin öncelik değişikliği sprintler arasında ekibe yansıtılmamış, yeni bir sprinte başlarken değişen önceliğin de uygulanan metodoloji nedeniyle çok doğal karşılandığı için ekibin ilerleyişine olumsuz bir etkisi olmamıştır.

Öncelikleri belirlemek müşterinin yetkisi dâhilindedir. Bu konudaki sorumluluğu müşterinin üstlenmesi sağlanmalıdır.

5. ÇEVİK METODOLOJİLERİN PROJEDE UYGULANMASI

Bu bölümde, bu tezin konusu olan projenin başlatma ve yürütme safhalarının Team System ile yönetilmesi anlatılmaktadır.

5.1. Projenin Team System ile Başlatılması

Projenin başlangıç safhası, Team System'in kurulumu, grupların ve izinlerin konfigürasyonu, check-in kurallarının tanımlanması, product backlogların oluşturulması ve release planının oluşturulmasını içermektedir.

Ekip üyeleri Microsoft Visual Studio Team Suite kurulumunu gerçekleştirdikten sonra proje yöneticisi tarafından team suite projesinin okunabilirliği kolay ve anlamlı bir proje adı olarak "UNISMART" tanımlanmıştır. Team Explorer aracılığıyla proje metodolojisi ve kaynak kontrol dosyalarının ayarlamaları yapılarak takım projesi oluşturulmuştur.

Bir proje yöneticisi, bir analist ve üç yazılım geliştiriciden oluşan ekip üyeleri için kullanıcı hesapları oluşturulmuştur. Project Administrators, Contributors, Build Server, Readers adlı gruplar tanımlanmış ve izinleri atanmıştır.

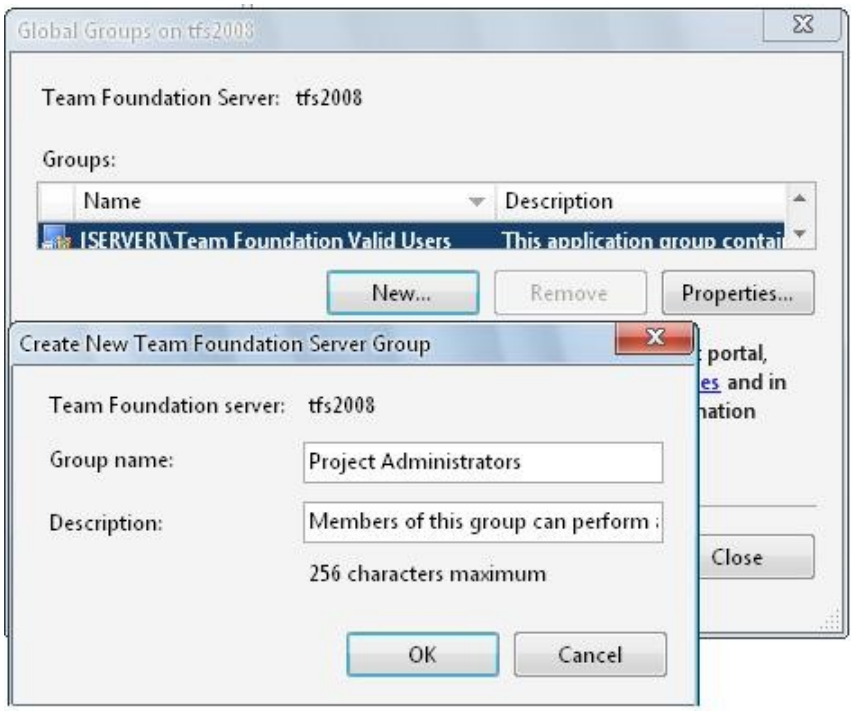
- Project Administrators: Proje Yöneticisi proje içerisinde tüm yetkilere sahiptir. Proje Build Service kullanma hakkı, check in, branch, security gibi tüm proje ayarları yapabilirler ama server ayarlarını değiştirme hakları yoktur. Yetkileri sadece bulundukları Proje için geçerlidir.
- Contributors: Proje kaynak kodlarına erişmek, düzenlemek, check in yapmak, iş nesneleri düzenleme haklarına sahiptirler.

- Build Services: Build Service de kuyruk yönetimi, projenin derlenmesi ve Build Agent ayarlarını değiştirme haklarına sahiptir.
- Readers: Sadece kaynak kodlarını inceleyebilirler düzenleme hakları yoktur. Aksi belirtilmediği sürece iş nesneleri oluşturamazlar. Örneğin, beta testçileri.

Team suite'deki izin isimleri şöyledir:

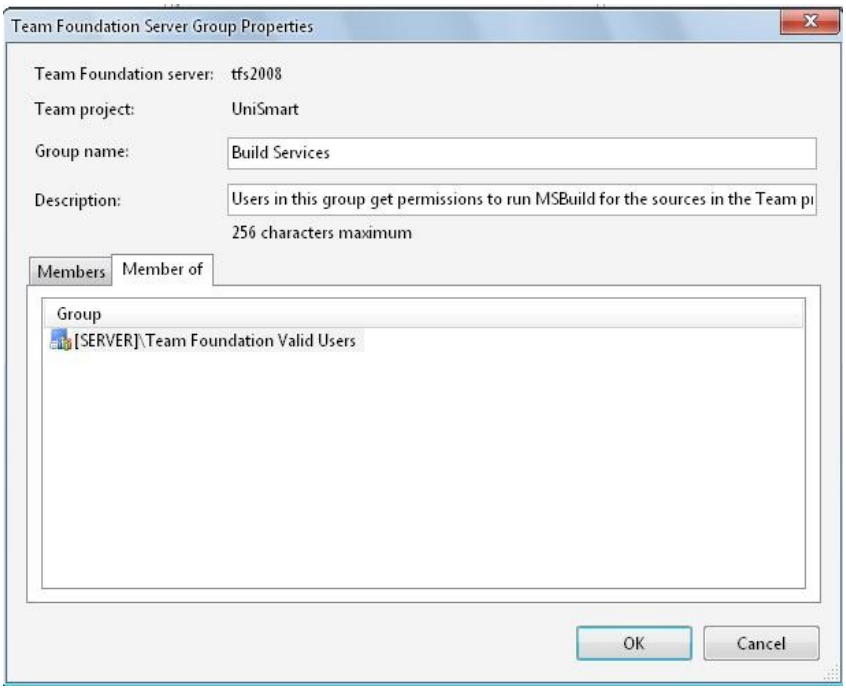
- Admin workspaces (workspacelerin yönetimi)
- Alter trace settings (izleme ayarlarını değiştirmek)
- Create a workspace (workspace oluşturmak)
- Create new projects (yeni projeler oluşturmak)
- Edit server-level information (server seviyesindeki bilgilerin değişimi)
- Manage process template (süreç şablonunun yönetimi)
- View server-level information (server seviyesindeki bilgileri görüntüleme)

TFS'de yeni bir grup, grup adı ve kısa bir açıklama (maksimum 256 karakter uzunluğunda) girilerek tanımlanır.



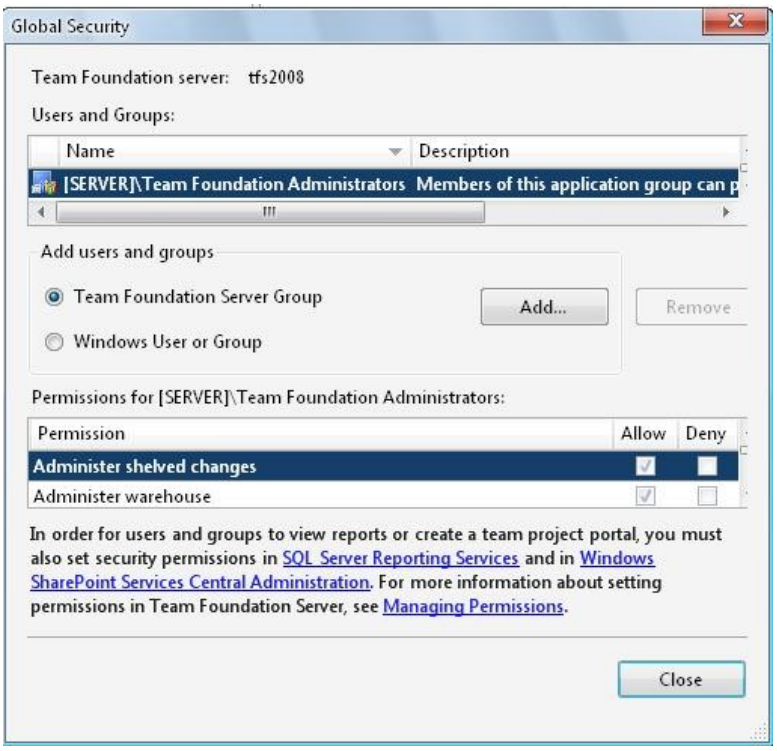
Şekil 5.1 TFS üzerinde grup tanımlama ekran görüntüsü

Tanımlanmış grubun özelliklerinden gruba atanacak üye/üyeler tanımlanır.



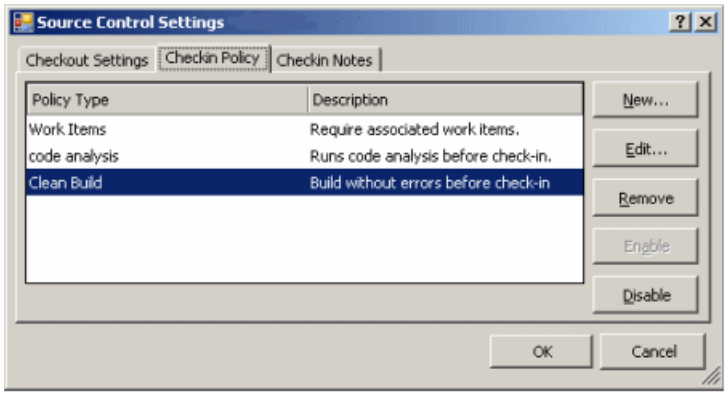
Şekil 5.2 Tanımlanan grubun özellikleri

Kullanıcı ya da gruba izin atama işleminde, kullanıcı ya da grup seçildikten sonra listelenen izinler allow/deny (izin ver/izin verme) olarak seçilir.



Şekil 5.3 Kullanıcılara ve gruplara izin atama ekranı

Değişikliklerin veritabanına yüklenmeden önce belirlenen iş kurallarını (iş nesneleriyle ilişkilendirilmesi, Kodun temiz derlenmesi, Kod analizi çalıştırılması, Birim testlerinin çalıştırılması) sağlaması gibi özelliklerle, yazılım geliştirme süreci içinde önemli bir role sahip olan check-in kuralları TFS Explorer'dan tanımlanmıştır. Aşağıda, proje yöneticisi tarafından, iş kurallarının belirlendiği örnek bir ekran sunulmuştur.



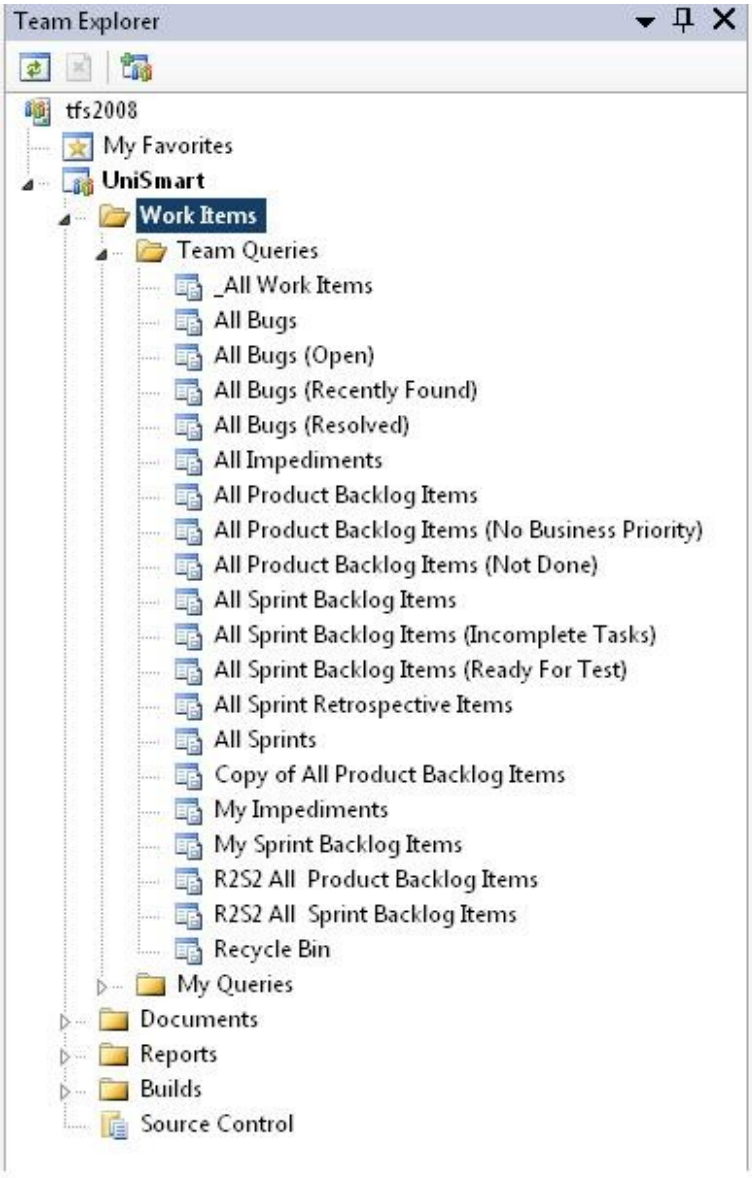
Şekil 5.4 Kaynak veritabanına yükleme işlemi sırasında çalıştırılacak iş kurallarının ekran görüntüsü

5.1.1. Projedeki Product Backlogların Oluşturulması

VSTS ile takım içinde, iş takibinin sağlandığı birlikte çalışma ortamı sunulmuştur. Proje ile ilgili iş takip nesnelerinin (work item) takım içinde ilgili kişilere atanması ve tamamlanma süreci takip edilebilmektedir.

İş takip sistemi, Visual studio ortamı ile entegre çalışmaktadır. Diğer takım üyeleri (mimar, yazılım geliştirici, test uzmanı) iş takip sürecine, çalıştıkları ortam içinden katılabilmektedirler.

TFS üzerindeki bir projede çalışırken, yerine getirmemiz gereken görevler takım lideri ve sistem analizcileri tarafından Work Item Type'ler kullanılarak takım üyelerine atanır. Work Item Type'ler yüklü olan metodolojiye göre farklılık göstermektedir. Scrum süreç şablonu için TFS ile birlikte varsayılan olarak gelen iş takip nesnelerinde product backlog item, sprint backlog item ve hata tipleri mevcuttur.

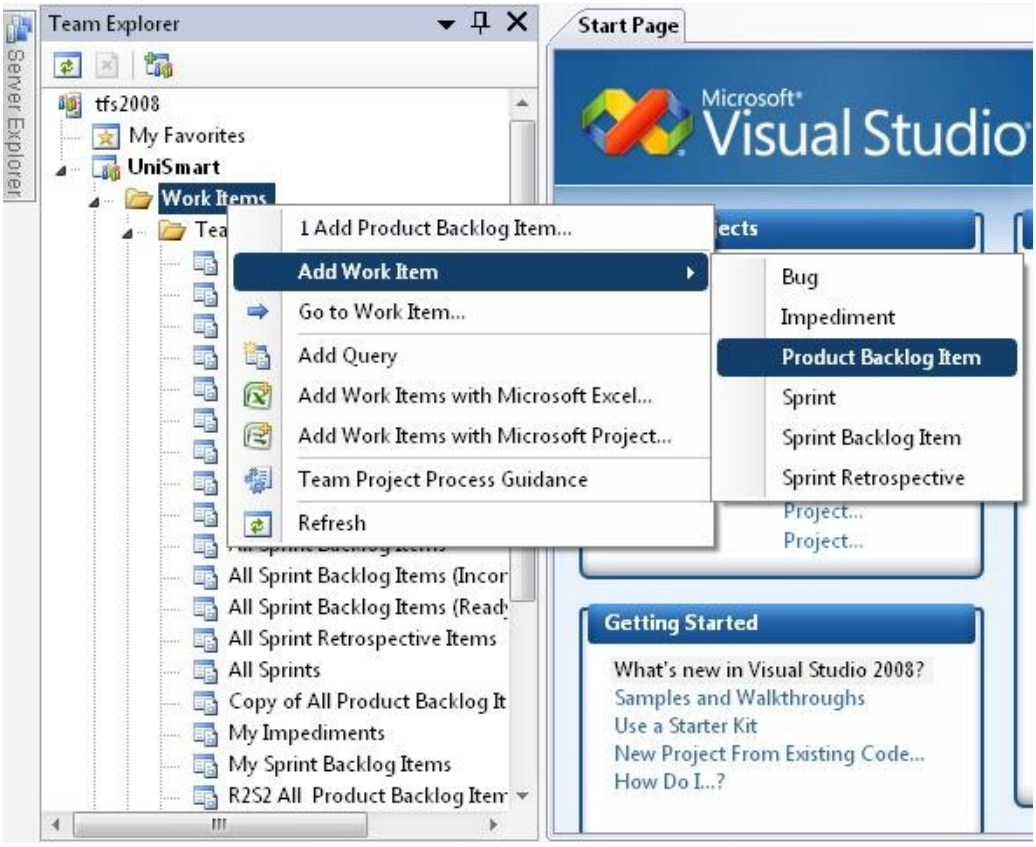


Şekil 5.5 Team Explorer'daki iş nesneleri

İş nesnelerine erişmek için Team Explorer aracılığı ile hazır gelen sorgular kullanılabilir. Örneğin iş nesnelerindeki My Sprint Backlog Items'dan yalnızca yazılımcıya atanmış işler listelenmektedir. Ayrıca sorgulama arabirimi ile iş nesnesi havuzu istenildiği şekilde sorgulanabilmektedir.

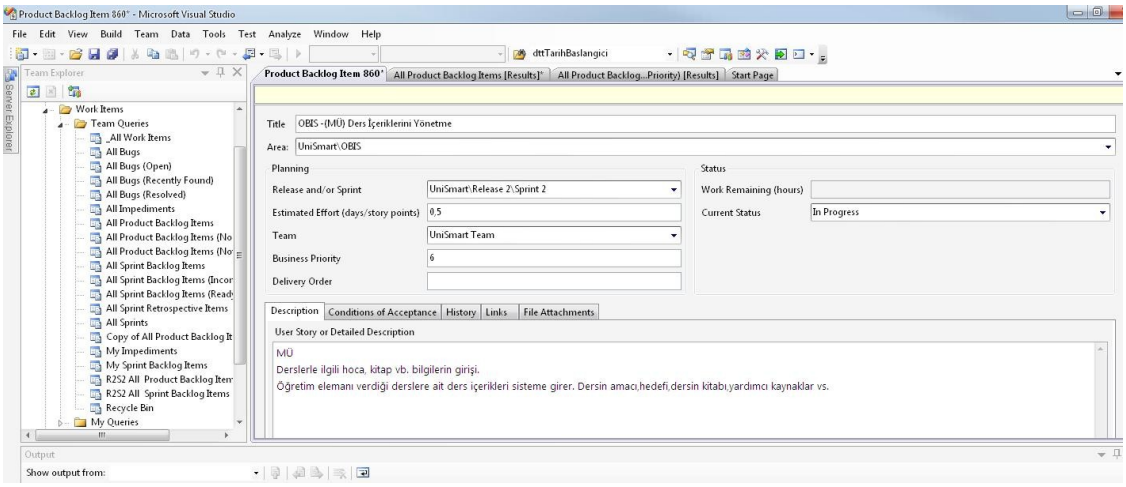
Proje başlangıç safhasında müşteri tarafından, 1-10 arasında verilen öncelik numaralarıyla önceliklendirilmiş product backloglar tanımlanmıştır. Gerçekleştirilen

proje, roller ve işlevler yönünden kapsamlı olduğundan backloglar TFS üzerinden tanımlanırken AROS, DORIS, KIYOS, OBIS, OSYM, PERBIS, SLAP gibi alt alan adlarıyla gruplandırılarak tanımlanmışlardır. "KIYOS- Kullanıcı Hesabı Tanımlama" adlı backlog başlığında önce alan adı daha sonra backlog'un adı şeklinde takip edilerek projenin takibi ve yönetilebilirliği kolaylaştırılmıştır. TFS üzerinden product backlog tanımlama ekranına erişim Şekil 5.6'da gösterilmektedir.



Şekil 5.6 TFS'den product backlog item tanımlama ekranına erişim

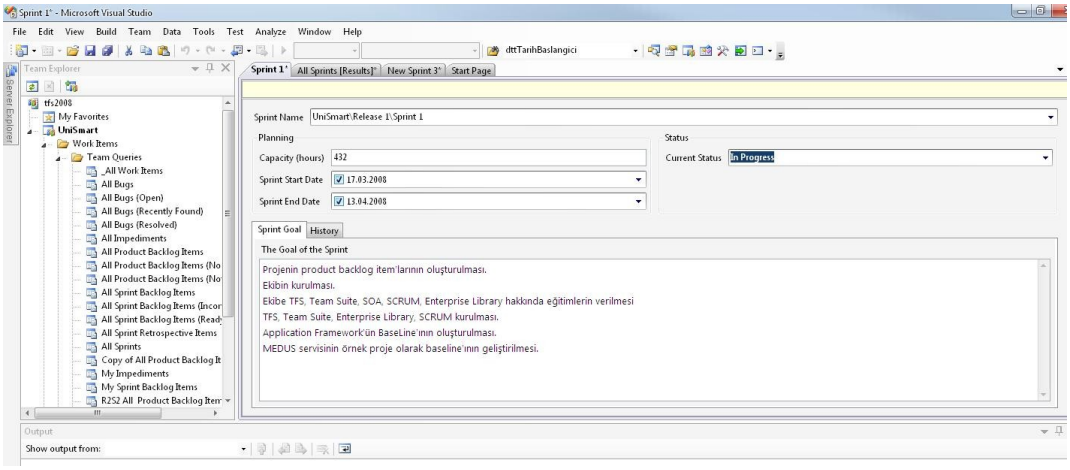
Product backlog item tanımlama ekranından backlog adı (başlık), alanı (Unismart/OBIS), sürüm ve/ veya sprint numarası, bu işlev için kaç gün gerektiği, takımın adı, öncelik puanı (1-10), ayrıntılı açıklama, varsa ilgili dosyalar eklenerek ve durum bilgisi "In Progress" seçilerek backlog item tanımlanmış olur. Yapılmadı (not done), tamamlandı (done), devam ediyor (in progress) gibi durum bilgileri mevcuttur. Product backlog tanımlayla ilgili ekran görüntüsü Şekil 5.7'de gösterilmektedir.



Şekil 5.7 TFS üzerinden product backlog item tanımlama ekranı

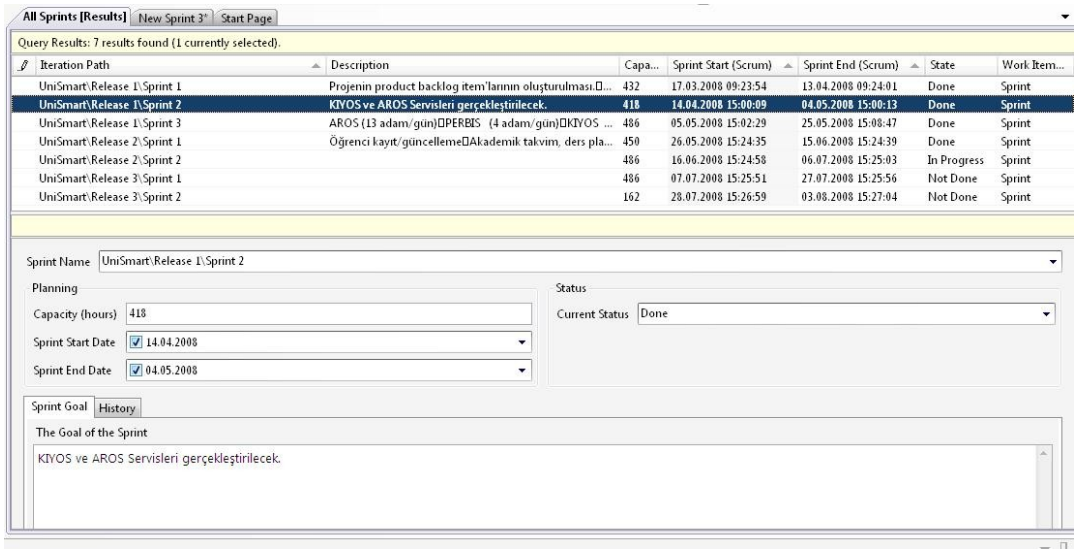
5.1.2. Proje'nin Sürüm Planının Oluşturulması

Product backlogdaki önceliklendirmelere göre, proje sürümlerinden ilki üç, ikincisi ve üçüncüsü ise iki iterasyondan oluşan üç sürüm planlanmıştır. Her bir sprint tanımlanırken sprintin adı (Projenin adı / Sürüm sayısı / Sprint numarası olarak tanımlanmıştır), o sprintte alınan işlevlerin tamamlanması için gerekli olan süre (saat bazında), sprint'in başlangıç ve bitiş tarihi, sprint'in amacı tanımlanır. Sprint tanımlama ekranı Şekil 5.8'de gösterilmektedir.



Şekil 5.8 Sprint tanımlama ekran görüntüsü

Gerçekleştirilen projede tanımlanmış bütün sprintler Şekil 5.9'da gösterilmektedir.



Şekil 5.9 Projedeki sprint'lerin listesi

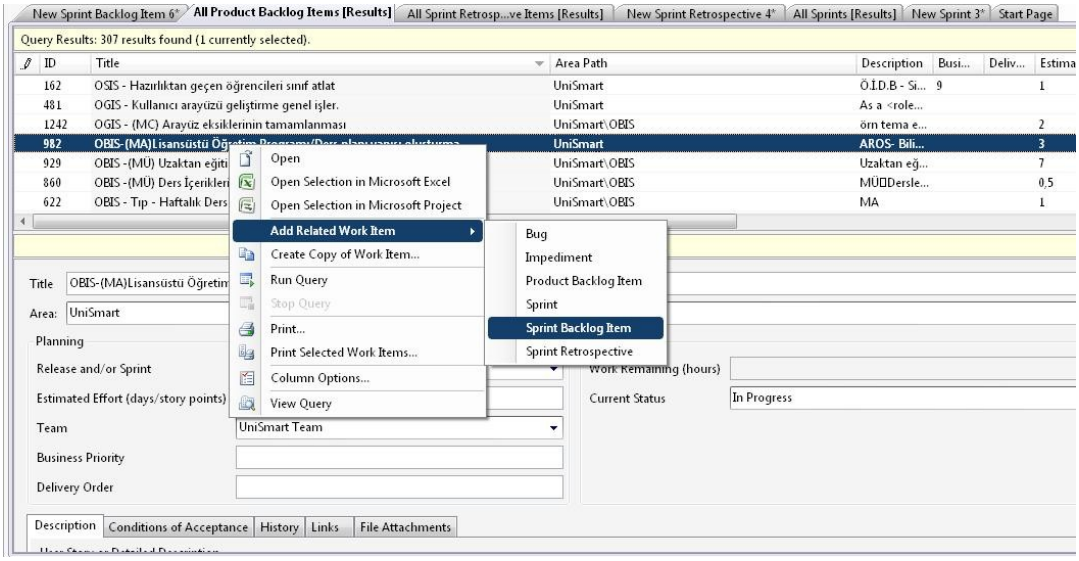
5.2. Projenin Team System ile Yürütülmesi

Bu bölümde work item tanımlaması, versiyon kontrolü ve veritabanı proje yönetiminin Team System ile yürütülmesi anlatılmaktadır.

5.2.1. Work Item Tanımlanması

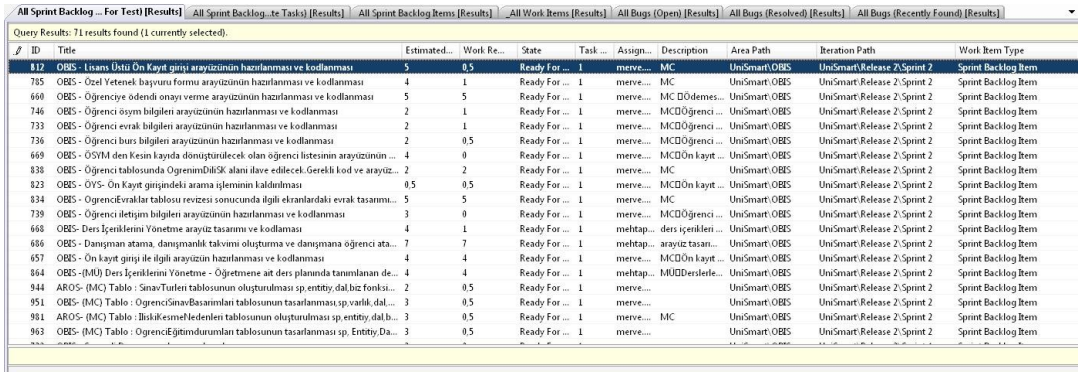
Work Item Tracking, TFS içindeki iş takibi bileşenidir. Ekip üyelerinin görevlerinin atanması, izlenmesi, yazılımdaki hataların takibi ve izlenmesi, yazılım üzerindeki senaryoların takibi ve benzeri işler için kullanılmaktadır.

TFS'de tanımlanmış product backlog itemlardan biri sağ tıklanıp, seçilen product backlog item için sprint backlog item tanımlanması Şekil 5.10'da gösterilmektedir.



Şekil 5.10 Sprint backlog item tanımlama ekranına erişim

Sprint backlog item tanımlama ekranından sprint adı, alanı (Unismart/OBIS), sürüm ve/veya sprint numarası, öncelik numarası, bu işlev için tahmini kaç gün gerektiği, işin kalan süresi, görev atanan kişinin adı, ayrıntılı açıklama ve durum bilgisi "In Progress" seçilerek backlog item tanımlanmış olur. Yapılmadı (not done), tamamlandı (done), devam ediyor (in progress), teste hazır (ready for test), iptal gibi durum bilgileri mevcuttur. Tanımlanmış bütün Sprint backlog listesi Şekil 5.11'de gösterilmektedir.



All Sprint Backlog ... For Test] [Results] | All Sprint Backlog...te Tasks] [Results] | All Sprint Backlog Items [Results] | _All Work Items [Results] | All Bugs (Open) [Results] | All Bugs (Resolved) [Results] | All Bugs (Recently Found) [Results]

Query Results: 71 results found (1 currently selected).

ID	Title	Estimated...	Work Re...	State	Task ...	Assign...	Description	Area Path	Iteration Path	Work Item Type
812	OBIS - Lisans Üstü Ön Kayıt girişi arayüzünün hazırlanması ve kodlanması	5	0,5	Ready For ...	1	merve....	MC	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
785	OBIS - Özel Yetenek başvuru formu arayüzünün hazırlanması ve kodlanması	4	1	Ready For ...	1	merve....	MC	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
660	OBIS - Öğrenciye ödendi onayı verme arayüzünün hazırlanması ve kodlanması	5	5	Ready For ...	1	merve....	MC DÖdemes...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
746	OBIS - Öğrenci ösym bilgileri arayüzünün hazırlanması ve kodlanması	2	1	Ready For ...	1	merve....	MCDÖğrenci ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
733	OBIS - Öğrenci evrak bilgileri arayüzünün hazırlanması ve kodlanması	2	1	Ready For ...	1	merve....	MCDÖğrenci ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
736	OBIS - Öğrenci burs bilgileri arayüzünün hazırlanması ve kodlanması	2	0,5	Ready For ...	1	merve....	MCDÖğrenci ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
669	OBIS - ÖSYM den Kesin kayıda dönüştürülecek olan öğrenci listesinin arayüzünün ...	4	0	Ready For ...	1	merve....	MCDÖn kayıt ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
838	OBIS - Öğrenci tablosunda OgrenimDiliSK alanı ilave edilecek.Gerekli kod ve arayüz...	2	2	Ready For ...	1	merve....	MC	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
823	OBIS - ÖYS- Ön Kayıt girişindeki aranma işleminin kaldırılması	0,5	0,5	Ready For ...	1	merve....	MCDÖn kayıt ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
834	OBIS - OgrenciEvraklar tablosu revizesi sonucunda ilgili ekranlardaki evrak tasarımı...	5	5	Ready For ...	1	merve....	MC	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
739	OBIS - Öğrenci iletişim bilgileri arayüzünün hazırlanması ve kodlanması	3	0	Ready For ...	1	merve....	MCDÖğrenci ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
668	OBIS- Ders İçeriklerini Yönetme arayüz tasarımı ve kodlaması	4	1	Ready For ...	1	mehtap...	ders içerikleri ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
686	OBIS - Danışman atama, danışmanlık takvimi oluşturma ve danışmana öğrenci ata...	7	7	Ready For ...	1	mehtap...	arayüz tasarı...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
657	OBIS - Ön kayıt girişi ile ilgili arayüzün hazırlanması ve kodlanması	4	4	Ready For ...	1	merve....	MCDÖn kayıt ...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
864	OBIS -(MÜ) Ders İçeriklerini Yönetme - Öğretmene ait ders planında tanımlanan de...	4	4	Ready For ...	1	mehtap...	MÜDDerslerle...	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
944	AROS- (MC) Tablo : SinavTurleri tablosunun oluşturulması sp,entitiy,dal,biz fonksi...	2	0,5	Ready For ...	1	merve....		UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
951	OBIS- (MC) Tablo : OgrenciSinavBasarimlari tablosunun tasarlanması,sp,varlık,dal,...	3	0,5	Ready For ...	1	merve....		UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
981	AROS- (MC) Tablo : IliskiKesmeNedenleri tablosunun oluşturulması sp,entitiy,dal,b...	3	0,5	Ready For ...	1	merve....	MC	UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item
963	OBIS- (MC) Tablo : OgrenciEğitimdurumları tablosunun tasarlanması sp, Entitiy,Da...	3	0,5	Ready For ...	1	merve....		UniSmart\OBIS	UniSmart\Release 2\Sprint 2	Sprint Backlog Item

Şekil 5.11 Tanımlanmış sprint backlog itemlar

Teste hazır olan sprint backlog item'lar, testle görevli olan ekip üyesi tarafından test edilir. Testi başarıyla geçen sprint backlog'un durum bilgisi tamamlandı olarak işaretlenir ve sprint backlog item tamamlanmış olur. Testi geçemeyen sprint backlog için ise hata tanımlaması yapılır. Hata tanımlaması; hatanın adı, hata seviyesi (blok, kritik, yüksek, orta, düşük), hata tespit tarihi, hata kapanış tarihi, hatanın durumu (yapılmadı, yapılıyor, yapıldı, teste hazır), tahmini yapılış süresi, kalan süre, sprint numarası girilerek gerçekleştirilir.

5.2.2. Versiyon Kontrolü

Versiyon kontrolü, uygulamadaki kaynak kodun yönetimini sağlayan uygulamadır. Yazılan kodların güvenilir bir şekilde saklanması, versiyonların takip edilmesi, kodun gelişimine katkıda bulunanların ve kodun kontrol altında tutulması gibi ihtiyaçları karşılamaktadır. Kodu takımlar halinde geliştirirken kod paylaşımı kolaylıkla yapabilir ve revizyonları kontrol edebilmektedir. Kodun geçmiş sürümleri de saklandığı için istenilen bir sürümüne geri dönülebilmektedir.

TFS yönetimi altındaki bir projede çalışırken sürekli çevrimiçi olunması gerekir. Bu da kopuk çalışmayı önler. Sunucu üzerindeki bir projeyi yazılım geliştiriciler üzerlerine alıp (check-out), düzenleyip gönderirler (check-in). Yani TFS lock-modify-unlock (üzerine al-değiştir-bırak) mantığıyla çalışır. Eğer TFS üzerinde merge'e (iki kişinin aynı dosya üzerinde çalışması durumunda yapılan

değişiklikleri birleştirme özelliği) izin verilmemişse dosya birisinin üzerinde ise onun dosyayı bırakması gerekir. TFS'de diğer sürüm kontrol sistemlerine göre check-in, check-out ve get latest version (son sürümü yüklemek) işlemlerinin süreleri oldukça kısadır. Bunun sebebi dosyalar ve bilgilerin MSSQL veritabanında tutulmasıdır.

TFS'de shelving (rafa kaldırma) özelliği bulunur. Bu özellik geliştiricilerin denemek istediği herhangi bir kodu ana projeye zarar vermeden denemesi için ideal bir yöntemdir. TFS'de kod göndermede belirli şartlar koyulabilir. Örneğin kodun en az bir kere derlenmesi gibi bir kural belirlenebilir.

5.2.3. Veritabanı Proje Yönetimi

Yazılım projelerinde yazılımcıların güncel veritabanı versiyonu üzerinde çalışmaları gerekiyor. Çoğu zaman, güncel versiyon gerçek ortamdaki veritabanı olmaktadır. Değişiklikler genelde, gerçek veritabanında geçekleştirilmektedir. Bu yüzden test veritabanlarına bu değişiklikler yansıtılmıyor veya eksik yansıtılabiliyor. Test ortamlarında, veritabanı değişiklik yönetiminin takip edilememesi sebebiyle, hem yazılım geliştirme hem de gerçek ortama geçiş sürecinde ciddi problemlerle karşılaşılmaktadır. Bu problemler büyük kayıplara sebep olabilmektedir. Veritabanı uzmanları için VSTS sürümü, veritabanı geliştirme ve gerçek ortama geçiş sürecini, tüm yazılım sürecine entegre etmesi ile önemli bir ihtiyaca cevap vermiş olmaktadır.

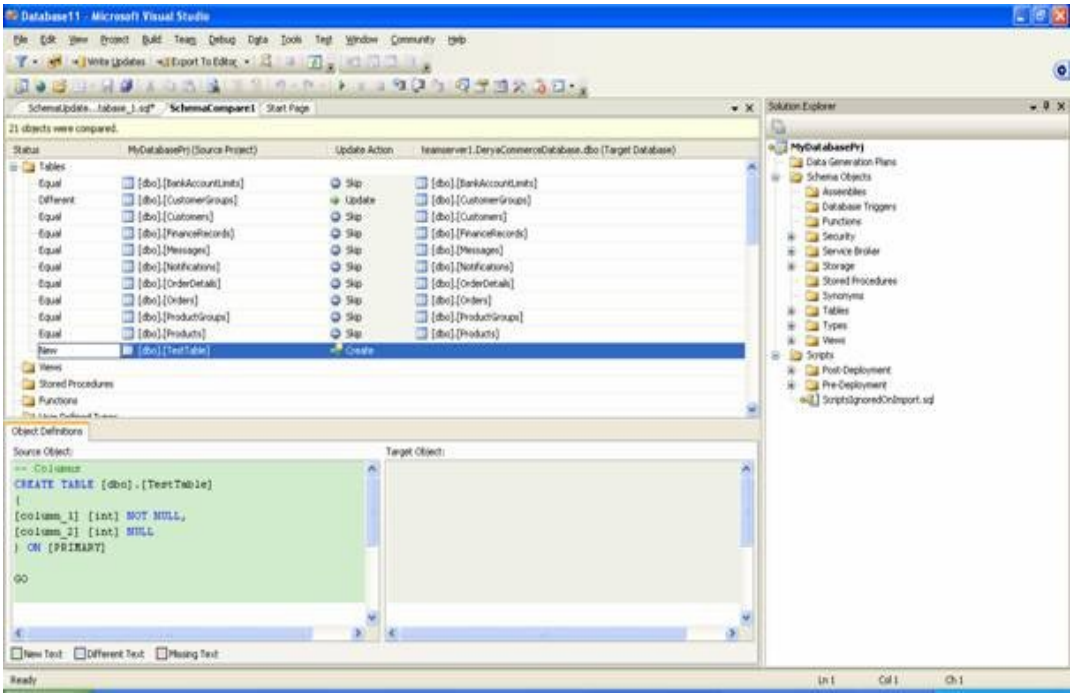
Veritabanı şeması, tüm nesneleri ile birlikte bir proje olarak takip edilebilmektedir. Sağlanan şablonlar sayesinde, veritabanı projesi oluşturulması ve var olan bir şemanın taşınması (import) kolaylıkla gerçekleştirilmektedir. Kaynak kod ve XML dosyaları gibi, tablo tanımları, veritabanı prosedürleri gibi veritabanı nesneleri de projenin bir parçası haline gelmiştir. Proje yapısının oluşturulması ile kaynak kod ve versiyon yönetimi imkanı da sağlanmış olmaktadır. Veritabanı proje yapısı olarak, SQL Server 2000, 2005 ve 2008 proje şablonları bulunmaktadır. Bu tezin konusu olan ve gerçekleştirilen proje, SQL Server 2005 veritabanını kullanmaktadır.

Test veritabanı projesi ile gerçekleştirilen veritabanı projesini karşılaştırma işleminde; öncelikli olarak, karşılaştırma yapılacak şemaların seçimi gerçekleştirilmelidir. Aşağıda, şema karşılaştırması seçim ekranı gösterilmiştir.

Şekil 5.12 Şema karşılaştırması seçimi

Seçilen kaynak ve hedef ortamlar karşılaştırılarak, farklılıklar listelenmektedir. Bu farklılıkların tanım (script) dosyası oluşturularak, sürecin devamında kullanmak için saklanabilir. Bu işlem içerisinden de, hedef şema güncellenebilmektedir.

Test veritabanı projesinde 2 değişiklik gerçekleştirildi. Bunlardan bir tanesi, bir tabloya yeni bir alan eklenmesi, diğeri ise yeni bir tablo eklenmesidir. Test veritabanı projesi ile veritabanı karşılaştırıldığında elde edilen ekran görüntüsü Şekil 5.13'de gösterilmektedir.



Şekil 5.13 Şema karşılaştırması sonuç ekranı

Tüm nesneler karşılıklı listelenmiş, farklı olanlar ve yeni eklenenler listede belirlenmiştir. Listenin altında, listede seçilen nesnenin kaynak ve hedef ortamlarında ki tanımları gösterilmektedir.

6. SONUÇ ve ÖNERİLER

Gerçekleştirilen projenin sprint bazında adam/gün hesapları aşağıdaki tabloda sunulmaktadır.

Tablo 6.1 Projenin sprint bazında adam/gün hesapları

Sürüm No	Sprint No			Toplam Kapasite							
		Başlangıç Tarihi	Bitiş Tarihi	Saat	Gün	Harcanan Emek	Proje Yöneticisi	Ekip Üyesi 1	Ekip Üyesi 2	Ekip Üyesi 3	Ekip Üyesi 4
Release 1	**Sprint 1**	17.Mar	23.Mar	**432**	**54**	110	34	16	16	44	0
		24.Mar	30.Mar			110	34	16	16	44	0
		31.Mar	06.Nis			94	34	16	0	44	0
		07.Nis	13.Nis			118	34	16	24	44	0
	Sprint 2	14.Nis	20.Nis	**418**	**52,25**	118	34	16	24	44	0
		21.Nis	27.Nis			138	34	16	24	44	20
		28.Nis	04.May			162	34	16	24	44	44
	Sprint 3	05.May	11.May	**486**	**60,75**	162	34	16	24	44	44
		12.May	18.May			162	34	16	24	44	44
		19.May	25.May			162	34	16	24	44	44
Release 2	**Sprint 1**	26.May	01.Haz	**530**	**66,25**	166	34	0	44	44	44
		02.Haz	08.Haz			182	34	16	44	44	44
		09.Haz	15.Haz			182	34	16	44	44	44
	Sprint 2	16.Haz	22.Haz	**546**	**68,25**	182	34	16	44	44	44
		23.Haz	29.Haz			182	34	16	44	44	44
		30.Haz	06.Tem			182	34	16	44	44	44
Release 3	**Sprint 1**	07.Tem	13.Tem	**546**	**68,25**	182	34	16	44	44	44
		14.Tem	20.Tem			182	34	16	44	44	44
		21.Tem	27.Tem			182	34	16	44	44	44
	Sprint 2	28.Tem	03.Ağu	728	91	182	34	16	44	44	44
		04.Ağu	10.Ağu			182	34	16	44	44	44
		11.Ağu	17.Ağu			182	34	16	44	44	44
		18.Ağu	24.Ağu			182	34	16	44	44	44
TOPLAM:					**461**	**3686**	**782**	**352**	**772**	**1012**	**768**

Bir proje yöneticisi, iki yazılımcı ve bir analistle başlatılan projenin ikinci ayından itibaren, ikinci sprintin başlangıcında, ekibe bir yazılımcı daha ilave edildi. Tablo 6.1'de sprint süreleri ve ekip üyelerinin sprint bazında çalışma saatleri verilmiştir. 4. Ekip üyesi 2. Sprint'in başında ekibe dahil edilmiştir. Yazılım geliştirme ortamında deneyimli olduğu için bir haftalık eğitim, adaptasyon süreci için yeterli olmuştur.

6 ay olarak öngörülen geliştirme süresi, müşterinin onayıyla süreç içerisinde eklenen ve çıkarılan işlevlerle 7 ay içinde 5 adam/461 günde 3686 saat iş gücüyle tamamlanmıştır. 5 adam/461 günde 114 product backlog item, 487 sprint backlog item üretilmiş ve 111 hata tespit edilmiştir. Sprint bazındaki backlog ve hata sayıları Tablo 6.2'de verilmiştir.

Tablo 6.2 Projeye ait backlog ve hata sayıları

Sürüm No	**Sprint No**	**Tamamlanan Product Backlog Item Sayısı**	**Tamamlanan Sprint Backlog Item Sayısı**	**Hata Sayısı**	**Hata / Sprint Item Oranı**
Sürüm 1	Sprint 1	7	44	15	%34
	Sprint 2	4	69	20	%29
	Sprint 3	18	104	21	%20
Sürüm 2	Sprint 1	25	90	15	%17
	Sprint 2	15	77	18	%23
Sürüm 3	Sprint 1	23	59	13	%22
	Sprint 2	22	44	9	%20
Toplam		114	487	111	%23

Tablo 6.2 sürüm ve sprintler bazında yatay olarak dilimlere ayrılmıştır. Tamamlanan product backlog sayısı, yapılacak işlevlerin genel tanımlarını içeren analiz birimlerini ifade eder. Sprint backloglar ise product backlogların detaylandırılmış hali olarak her bir product backlog için bir ya da daha fazla sprint backlog oluşturulmuştur.

Hata sayısı, ilgili sprintte tam olarak çalışmayan sprint backlogların sayısıdır ve sprint içerisinde test aşamasında tespit edilerek çözümlenmiştir. Burada yazılımcıların hata/kod oranlarını hesap etmek amaçlanmış, sprint backlog için verilen sürenin %10'undan daha fazla sürede çözülebilen sorunlar hata olarak nitelendirilmiştir. Zamanla metodoloji oturdukça ve teknolojiye alışıldıkça hata oranını azaldığı gözlemlenmiştir.

Yazılım doğru yapılan bir çevik metodoloji uyarlamasıyla geliştirildiğinde, müşteri ve ekip deneyimsiz olsa bile çevik metodolojileri kullanarak başarılı olmanın mümkün olduğu görülmüştür. Belirlenen süre içerisinde versiyon başarıyla tamamlanmış ve kullanıma geçmiştir.

Bu çalışmada elde edilen bulgulardan en önemlisi; gerçekleştirilen projede tek bir çevik metodolojiye bağlı kalınmadan ekibe ve müşteri gereksinimlerine özgün pratikler seçilerek oluşturulmuş olmasıdır. Uyarlamada tercih edilen pratikler, projenin büyüklüğü, gereksinimlerin karmaşıklığı, yazılımın kapsamı ve ekibin yapısı değerlendirilerek belirlenmiştir. Çalışmada bu pratiklerin, seçilme nedenleri, nasıl uyarlandığı, uyarlanmasındaki sıkıntılar ve bu sıkıntıları çözümleme çabaları paylaşılmış olup, edinilen deneyimler doğrultusunda uyarlama yapacaklar için önerilerde bulunulmuştur.

Diğer bir önemli bulgu ise, ekibin teknoloji ve araçları kullanma konusundaki becerilerindeki iyileşmenin, metodolojiyi uygulama konusundaki ilerlemeden çok daha yüksek oluşudur. Bunun nedeniyse genel olarak yeni teknolojileri uygulamaya açık olan yazılımcıların, insanın doğası gereği iş yapış alışkanlıklarını değiştirmeye aynı derecede açık olmayışlarıdır. Ekip, kodla ve düzelt (code & fix) yöntemlerle yazılım geliştirmeye aşina olduğundan, çevik metodoloji pratiklerini uygulamada iş yapış alışkanlıklarına uymadığından zorlanmışlardır.

Kullanılan VSTS aracı ile sağlananlar:

- Proje yönetimi
- Scrum süreç şablonu ile süreç yönetimi
- Proje ile ilgili iş takip nesneleri ve iş akış yönetimi
- Entegre ve çok kullanıcılı proje geliştirme ortamı sağlaması
- Kodun merkezi olarak yönetilmesi, saklanması, korunması
- Otomatik derleme
- Entegrasyonun sağlanması
- Versiyon yönetimi
- Erişim yetkisinin yönetimiyle, güvenliğin sağlanması
- Raporlar aracılığıyla performans değerlendirme

Bu aracı kullanma konusunda deneyimsiz olan ekip üyelerinin, proje başlangıcında TFS konusunda eğitimlerinden sonra iş çıkarma sürelerinin azaldığı ve proje yönetiminin kolaylıkla yürütüldüğü görülmüştür. Çevik yazılım geliştirmeyi destekleyen araç seçimi, proje takvimine sadık kalmak için çok önemli bir seçimdir. Gerçekleştirilen projede VSTS seçimiyle doğru bir karar verildiği görülmüştür.

Proje sonunda elde edilen bulgulara dayanarak şu sonuçlar ve öneriler ortaya konmaktadır:

1. Projede uygulanacak mimari modele ve kullanılacak teknolojilere hakim, çevik metodoloji deneyimi olan kişilerden oluşturulan bir ekip, proje için en ideal olanıdır. Eğitim ve adaptasyon süresi olmadan iş yapış alışkanlığı kazanmış bir ekiple yazılım geliştirmek projelerin başarısını artıracaktır.

2. Ekip oluşturulurken hem teknolojiye hem de metodolojiye hâkim olan elemanlar bulunamıyorsa, metodolojiye hâkim olanlar öncelikle tercih edilmelidir. Ekibin teknoloji ve mimari model eksikliği proje başında eğitimle tamamlandığında, ideal bir ekip kurulmuş olacaktır.

3. Müşteri ya da müşterinin atayacağı yetkin biri projeye aktif olarak dâhil edilmelidir. İterasyonlar arasında gerçekleştirilen fikir alış verişleri, çevik yöntemlerin değişikliğe olan yatkınlığı ve müşteri odaklı yaklaşımları sayesinde, projeler, müşteriler ile birlikte değişerek gelişmekte ve proje sonunda da müşteri ihtiyacını en iyi derecede karşılayabilecek programlar ortaya çıkacaktır.

4. Uygun çevik pratikleri belirlerken ekip değil, proje baz alınmalıdır. Pratikler, projenin büyüklüğü, gereksinimlerin karmaşıklığı, yazılımın kapsamı değerlendirilerek belirlenmelidir.

5. Müşteri, proje yöneticisi ve ekip tam bir takım olmalı ve herkesin yürekten inandığı ortak bir proje vizyonu oluşturulmalıdır.

6. Çevik süreçleri destekleyen ve ortak bir çalışma ortamı sağlayan araçlar kullanılmalıdır.

Çevik metodolojilerde başarının anahtarı müşteri gereksinimleri, kısıtlar, ekip ve çevik pratiklerin en uygun şekilde bir araya getirilmesidir. Çevik metodoloji kullanılarak gerçekleştirilecek bir yazılım projesinde esas alınacak nokta, müşteri gereksinimleri ve kısıtlarıdır. Gereksinim ve kısıtları belirleyip odak noktasına aldıktan sonra ilk olarak mevcut bir ekiple mi ilerlenecek yoksa yeni bir ekip mi kurulacak sorusu cevaplanmalıdır? Eğer ekip önceden belirlenmişse ve ekibe müdahale şansınız yoksa, çevik pratiklerini seçerken proje ihtiyaçları ve kısıtlarının ötesinde ekibin deneyimi, bilgi ve beceri durumları da göz önünde bulundurulmalıdır. Özellikle teknoloji ve mimari seçimi, proje için verilen süre kısıtlıysa, ekibin yetenek ve deneyimlerine göre belirlenmelidir. Yeni bir teknoloji ve/veya mimari seçimi, ancak bu konularla ilgili gerekli eğitim ve eğitimde öğrenileni tecrübe etmek için yeterli süre varsa tercih edilmelidir. Eğer hazırda bir ekibiniz yoksa, öncelikle projeye uygun çevik metodoloji pratiklerini, projede kullanacağınız teknoloji ve mimariyi belirleyin. Ardın da tüm bu proje bileşenlerine (çevik pratikler, mimari, teknoloji) en uygun ekibi kurun. Böylece projenin başarı şansını olası en üst seviyeye çıkartmış olursunuz.

7. KAYNAKLAR

1. Highsmith J., “Agile Software Development Ecosystems”, Addison Wesley, 2002.
2. Boehm B., Turner R., “Observation on Balancing Discipline and Agility”, Proceedings of the Agile Development Conference, IEEE Computer Society, 2003.
3. Cockburn A., “Agile Software Development”, Addison-Wesley Longman, 2001.
4. http://agilemanifesto.org, (01.03.2010)
5. http://agilemanifesto.org/principles.html, (01.03.2010)
6. Martin C. R., “Agile Software Development”, Prentice Hall, 2002.
7. Shore J., “The Art of Agile Development”, O’Reilly, 2007.
8. Acar Ö. , “Extreme Programming”, Pusula, 2009.
9. Larman C., “Agile&Iterative Development”, Addison-Wesley, 2007.
10. Beck K. with Andres C., “Extreme Programming Explained: Embrace Change”, Addison–Wesley, 2007.
11. Hayes S., Andrews M., “An Introduction to Agile Methods”, Mart 2009 http://www.scribd.com/doc/4465171/Intro-To-Agile-Methods.
12. Beck K., “Extreme Programming”, Addison Wesley, 2002.
13. http://en.wikipedia.org/wiki/Microsoft_Solutions_Framework, (11.03.2010)
14. Turner M., “Microsoft Solutions Framework Essentials”, Microsoft Press, 2006.
15. Beck K., Fowler M., “Planning Extreme Programming”, Addison Wesley, 2001.
16. Coram M., Bohner S., “The Impact of Agile Methods on Software Project Management”, Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS’05), IEEE Computer Society, pp. 363-370, 2005.
17. Abrahamsson P., Salo O., Ronkainen J., Warsta J., “Agile Software Development Methods, Review and Analysis”, Espoo 2002, VTT Publications 478. 107 p.
18. Newkirk J., Martin R. C. , “Extreme Programming in Practice”, Addison Wesley, 2001.
19. S. Fraser, A. Martin, D. Hussman, C. Matts, M. Poppendieck, L. Rising, “The XP Customer Role”, Proceedings of Extreme Programming and Agile Processes in Software Engineering, Springer Verlag, Berlin, Germany, pp. 342–346, 2004.

20. Schwaber K., “Agile Project Management with Scrum”, Microsoft Press, 2004.
21. http://www.ambysoft.com/unifiedprocess/agileUP.html, (12.03.2010)
22. Kniberg H., “Scrum and XP from the Trenches”, InfoQ, 2007.
23. http://www.prime.com.tr/agile/agile_primeteknoloji.pdf, (01.04.2010)
24. http://ptgmedia.pearsoncmg.com/images/art_schwaber5_scrum_xp/elementLinks/schwaber5_fig1.gif, (04.04.2010)
25. www.csharpnedir.com/Mimages/Mimari.jpg, (03.05.2010)
26. Augustine S., “Managing Agile Projects”, Prentice Hall, 2005.
27. http://martinfowler.com/articles/continuousIntegration.html, (21.03.2009)
28. http://www.stromboli.co.uk/msf.aspx, (14.03.2009)
29. “MSF Risk Management Discipline v.1.1”, Mart 2009.
30. http://www.itabhi.com/ld.htm, (21.03.2009)
31. http://www.dsdm.org/atern/introduction/principles/, (21.03.2009)

ÖZGEÇMİŞ

1984 yılında İstanbul'da doğmuştur. 2007 yılı Maltepe Üniversitesi Bilgisayar Mühendisliği Bölümü mezunudur. Yazılım Mühendisliği, Veri Madenciliği konularıyla ilgilenmektedir. Ayrıca ikinci üniversite olarak Anadolu Üniversitesi İşletme Bölümünde 4. sınıfta okumaktadır. 2007 yılından beri Maltepe Üniversitesi'nde Arş. Gör. olarak akademik çalışmalarına devam etmektedir.