T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
SOSYAL BİLİMLER ENSTİTÜSÜ

PROJELERİN VERİMLİLİĞİNDE TAKIM ÇALIŞMASI VE YAZILIMCININ ÖNEMİ

ELVAN ARSLAN

YÜKSEK LİSANS TEZİ
İŞLETME ANABİLİM DALI

GEBZE
2011

T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
SOSYAL BİLİMLER ENSTİTÜSÜ

PROJELERİN VERİMLİLİĞİNDE TAKIM ÇALIŞMASI VE YAZILIMCININ ÖNEMİ

ELVAN ARSLAN

YÜKSEK LİSANS TEZİ
İŞLETME ANABİLİM DALI

TEZ DANIŞMANI: DOÇ. DR. HALİM KAZAN

GEBZE
2011

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ	**SOSYAL BİLİMLER ENSTİTÜSÜ** **JÜRİ ONAY FORMU**

G.Y.T.E. Sosyal Bilimler Enstitüsü Yönetim Kurulu'nun ……. tarih ve …… sayılı kararıyla oluşturulan jüri tarafından …../…./2010 tarihinde tez savunma sınavı yapılan Projelerin Verimliliğinde Takım Çalışması ve Yazılımcının Önemi konulu tez çalışması İşletme Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE
(TEZ DANIŞMANI) : Doç. Dr. Halim KAZAN

ÜYE : Doç. Dr. Hakan KİTAPÇI

ÜYE : Doç. Dr. Metin ATEŞ

ONAY

G.Y.T.E. Sosyal Bilimler Enstitüsü Yönetim Kurulu'nun ……/……/2010 tarih ve ……/…… sayılı kararı.

İMZA/MÜHÜR

İMZA/MÜHÜR

ÖZET

TEZİN BAŞLIĞI: PROJELERİN VERİMLİLİĞİNDE TAKIM ÇALIŞMASI VE YAZILIMCININ ÖNEMİ

YAZAR ADI: Elvan ARSLAN

Bu tez çalışmasında Türkiye'de faaliyet gösteren özel bir kuruluşta takım çalışması ve yazılımcının proje verimliğine olan etkisi araştırılmıştır. Altmış üç soruluk bir anket çalışması ile Türkiye'de faaliyet gösteren özel bir bankanın bilgi teknolojileri bölümünden veriler elde edilmiştir. Anketimize 140 çalışan katışmış ve sonuçlar yazılım geliştirme sürecinin aşamalarını oluşturan talep iletimi, kapsam hazırlığı, analiz, tasarım, yazılım geliştirme, test, kullanıcı kabul testi ve bakım aşamaları ile proje verimliliği arasındaki ilişkiyi ortaya koymak maksadıyla analiz edilmiştir.

Yapılan analiz çalışmasında talep iletimi, kapsam hazırlığı, analiz, tasarım, yazılım geliştirme, test, kullanıcı kabul testi, takım çalışması ve yazılımcının, proje verimliliği ile pozitif yönlü bir ilişkisi olduğu saptanmıştır.

ANAHTAR KELİMELER: Yazılım Geliştirme, Takım Çalışması, Verimlilik, Yazılım Geliştirme Süreçleri

SUMMARY

THESIS : THE SIGNIFICANCE OF SOFTWARE DEVELOPER AND THE ROLE OF TEAMWORK ON PROJECT EFFICIENCY

AUTHOR :Elvan ARSLAN

In this study, the effect of software developer and the role of teamwork on project efficiency was investigated. Data were obtained from a private bank in Turkey. 140 people responded to the questionnaire which includes sixty-three questions and the data were analyzed to find out the relationship between the new inquiry, context planning, analysis, designing, software development, testing, user acceptance test, teamwork, software developer and project efficiency.

The analysis showed that new inquiry, context planning, analysis, designing, software development, testing, user acceptance test, teamwork and software developer had a positive effect on project efficiency.

Keywords: Software Development, Teamwork, Efficiency, Processes of Software Developing

TEŞEKKÜR

Bu tez çalışmasında bana büyük desteği ve katkısı olan danışmanım Sayın Doç. Dr. Halim KAZAN'a, çalışmamdaki yardımları ve manevi destekleri için Gebze Yüksek Teknoloji Enstitüsündeki çok değerli hocalarıma, araştırma görevlilerine ve çalışanlarına, kıymetli zamanlarını ayırarak araştırma anketlerimi dolduran tüm IBTECH Uluslararası Bilişim ve İletişim Teknolojileri çalışanlarına, çeviri çalışmalarında desteğini esirgemeyen sevgili kuzenim Feyza AKGÜN'e, zor günlerimde her zaman desteğiyle yanımda olan sevgili eşim Çağrı ARSLAN'a, eğitim hayatım boyunca en büyük destekçilerim olan sevgili annem Sultan İNCE'ye, sevgili babam Halil İbrahim İNCE'ye ve sevgili ağabeyim Adnan İNCE'ye sonsuz teşekkürlerimi sunarım.

Sevgi ve saygılarımla…

Elvan ARSLAN

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

TABLO DİZİNİ

GİRİŞ

Fiziksel kısıtlamalar olmamasına rağmen yazılım, karmaşık ve anlaşılması zordur. İnsanoğlunun işlerini kolaylaştırmak ve hızlandırmak her geçen gün biraz daha karmaşık yapılar haline gelmesine sebep olmuştur. Donanımdaki ve teknolojideki gelişmeler de yazılımlara yansımış, yazılımlara olan ihtiyaç arttıkça daha etkin, daha verimli yazılım geliştirme yollarına ihtiyaç duyulmuştur. Bu çalışmada, bu karmaşık yapının geliştirme süreçlerinin, takım çalışmasının ve yazılımcının proje verimliliğine olan etkisinin incelenmesi amaçlanmıştır.

İÇERİK

Araştırmanın 1. bölümünde yazılımın sistem içindeki rolü, 2. bölümünde yazılım geliştirme süreci modelleri (disiplinsiz, şelale, evrimsel, çevik modeller) birbirleriyle karşılaştırılarak incelenmiştir. 3. bölüm, yazılımın temel aktiviteleri olan ihtiyaç analizi, tasarım, kodlama, test, teslim ve bakım safhalarını içermektedir. Tüm safhalar içerdikleri aktiviteler, amaçlar, özellikler, dokümanlar, testler, kullanılan teknik ve araçlar, ölçümler bazında tek tek incelenmiştir. 4. bölümde takım kavramı ve yazılım geliştirme takımları incelenmiştir. 5. bölümde verimlilik konusu, verimliliğin önemi, yazılım dünyasında verimliliğin ölçülmesi konuları, 6. bölümde yazılım projelerinde başarısızlık ve nedenleri incelenmiştir. Araştırmanın son bölümü olan 7. Bölüm, takım çalışması ve yazılımcının projenin verimliliğine etkisini inceleyen bir uygulama ile uygulamanın analiz sonuçlarını içermektedir.

Çalışmanın uygulama bölümünde Türkiye'de bankacılık yazılımı geliştiren özel bir şirketin yazılım geliştirme süreçlerinin proje verimliliğine olan etkisini anlamaya ilişkin bir anket çalışması yapılmıştır. Sonuç bölümünde, anket sonuçlarından elde edilen bulgular değerlendirilmiş, geliştirilmesi gereken alanlar saptanmıştır [GÜL, 2006].

BÖLÜM 1:YAZILIM VE SİSTEM İÇİNDEKİ ROLÜ

Yazılımın gelişimi 20. yüzyılın 2. yarısında yarı iletken (semiconductor) teknolojisinin gelişimi ile ortaya çıkan dijital hesaplamadaki gelişime denk gelmektedir. Yazılım, data sistemlerinin kontrol ve proses elemanıdır. Bu, bilgisayarın data kaynaklarını kullanılabilir bilgilere ve işlemlere çevirmesi anlamına gelmektedir. Birçok insan yazılımı bilgisayar programlarıyla eş tutar. Aslında, bu çok dar bir bakış açısıdır. Yazılım, sadece program değil, ayrıca programı doğru kullanmayı sağlayan dokümantasyon ve konfigürasyon datalarıyla da birleşmektedir.

Bilgisayarların ilk zamanlarında, yazılımlar 2. Dünya Savaşı'nın çabalarının "topçuluk" tablolarının hesaplanması için kullanılmıştır. Bu gün ise yazılım, bilgisayarları küçüklerden süperlerine kadar kontrol etmek ve sonsuz sayıda işlem gerçekleştirmek için kullanılmaktadır. Bu çok iş bilen potansiyel güç, yazılımı modern sistemlerde vazgeçilmez kılmaktadır.

Yazılım ve donanım içinden çıkılmaz derecede birbiriyle bağlantılı olsalar da gelişim tarihleri farklı olmuştur [Kossiakoff, 2003].

Donanımsal ilerleme bilgisayar dünyasında çok hızlı olmuştur. Mikroçip üreticileri için çok önemli olan ve yaygınca bilinen Moore's Law (Moore Yasası) (Her 18 ayda silikon çiplerin üstüne yerleştirilebilen transistor sayısının iki katına çıkacağını öngören yasa) , bu ilerlemeyi çok güzel göstermektedir. Donanım 1970'ler, 1980'ler 1990'lar boyunca 18 ayda bir performansını ikiye katlarken yazılım bocalamış, başarısızlıklar artmıştır. Fakat Bilişim Teknolojilerinde ilerleyen yıllarda, beklenenin aksine donanımların yazılımlara yön vermesi yerini sürpriz bir biçimde donanımların yazılımlara yetişme ve uyum sağlama ihtiyacını doğurmuştur. Öyle ki bir çok donanım üreticisi firma artık ürettikleri donanımsal parçalara "belirli bir yazılıma uyumludur" diye belirtme ihtiyacı duyar olmuşlardır. Yazılım sektörü donanım sektöründen daha önemli bir hal almaya

başlamıştır. Yine de yazılım sektöründeki ilerleme yeni yeni gelişim içerisine girmektedir [Kurnaz ve ark. , 2003, s.4].

1.1 Yazılım ve Donanım Arasındaki Farklar

Yapısal parçalar: Çoğu donanım parçaları standart parçalar (transistor, motor,..) dan yapılmaktadır. Diğer yandan yazılım yapısız parçaları, bilgisayar tarafından performansını gösteren form için sonsuz sayıda yoldan birleştirilmektedir. Donanım alt sistemleri ve parçaları gibi sınırlı sayıda ortak fonksiyonel bloklar yoktur.

Ara yüzler: Yazılımın yapısında daha fazla sınırlamalar olduğundan donanıma göre daha derin ve az görülebilir bağıntılarla daha fazla ara yüz vardır. Bu özellikler, iyi bir sistem modülerliği kurmayı ve değişikliklerin etkisini ölçmeyi zorlaştırmaktadır.

Fonksiyonalite: Donanımda fiziksel zorlamalardan olan sınırlar yazılımın fonksiyonalitesinde yoktur. Kritik, kompleks, standart dışı operasyonlar yazılıma uygulanabilmektedir.

Boyut: Donanımın boyutu hacim, ağırlık gibi zorlamalarla sınırlandırılırken yazılımda böyle bir sınır yoktur.

Değişebilirlik: Yazılımda yapılan değişikliklerin etkilerini tahmin etmek, etkileri karmaşıklık ve ara yüz problemlerine göre belirlemek daha zordur.

Hata biçimleri: Donanım yapı ve operasyonda sürekli iken yazılım dijital ve süreksizdir. Yazılım hataları genellikle ani ortaya çıkmakta, sıklıkla sistemin bozulmasıyla sonuçlanmaktadır.

Soyutluluk: Donanım parçaları fiziksel, yazılım ise soyuttur. Yazılım kodlarının formunu anlamak yazan dışında diğerleri için zordur. Soyutluluk, yazılım ile donanım arasındaki en önemli farktır [Kossiakoff, 2003].

1.2. Yazılım Sistem Tipleri

Genel kabul edilmiş yazılım sistemleri kategorileri olmamasına rağmen, yazılım sistemlerini 3 tipe ayırmak faydalıdır:

- Yazılım gömülü sistemler
- Yazılım yoğun sistemler
- Hesap yoğun sistemler

Yazılım gömülü sistemler: Yazılım, donanım ve insanların oluşturduğu hibrid bir kombinasyondur. Bu kategorideki sistemlerde donanım temel aksiyonları gerçekleştirir, fakat yazılımın burada büyük rolü vardır. Örnek olarak, taşıtlar, radar sistemleri, bilgisayar kontrollü üretim makineleri,..gibi. Yazılımın buradaki rolü; insan operatörlerinin desteğinde kontrol fonksiyonlarını gerçekleştirmek ve donanım parçalarını aktive etmektir. Bu rol, hane halkı aletlerinin kontrolünden askeri silah sistemlerindeki yüksek kompleks otomatik fonksiyonların kontrolüne kadar uzanabilmektedir.

Yazılım yoğun sistemler: Tüm bilgi sistemlerini içeren bu sistemler, insan operatörlerini desteklemek için çoğunlukla bilgisayar ve insanların gerçekten sistem fonksiyonalitesinde performans gösterdikleri bilgisayar ve kullanıcıların iş ağlarından oluşmuştur. Örnek olarak, otomatik bilgi işlem sistemleri, uçak rezervasyon sistemleri, finansal yönetim sistemleri,.. gibi.

Hesap yoğun sistemler: Diğerlerinden farklıdır. Kompleks hesaplamalar için büyük ölçekli hesaplama kaynakları içermektedir. Örneğin, hava analizleri ve tahminleme merkezleri, nükleer sonuçları tahminleme sistemleri ve diğer hesap yoğun sistemler [Kossiakoff, 2003].

1.3. Yazılım Üretimindeki Zorluklar

Her ne kadar, donanım her geçen gün biraz daha hızlanıp ucuzlasa da hızı ve küçülmeyi engelleyen kısıtlar göz ardı edilememektedir (ışığın hızı, atomun boyutu). Yazılımda ise fiziksel değil fakat farklı problemlerle uğraşılması gerekmektedir [Schach, 1993].

1986'da Frederics Brooks "Gümüş Kurşun Yoktur - Yazılım Mühendisliğinde Esas ve Rastlantı" adlı etkileyici makalesini yayınlamıştır. Brooks'un makalesinin temel savı, gelecek on yıl için üretkenlikte 1'e 10'luk bir gelişmenin habercisi olan bir araç ya da metodolojinin (gümüş kurşun) var olmadığıydı.

Brooks (1986), donanımdaki sorunlara (hız ve boyut) benzer olarak, yazılım üretim tekniği ile ilgili hiçbir zaman çözülemeyecek doğal problemler olduğunu belirtmiştir. Brooks'a göre yazılım her zaman zor olacaktır.

Brooks, yazılım zorluklarını 2'ye ayırmaktadır:

- Yazılımın doğasında olan, değiştirilemeyen zorluklar
- Rastlantısal, doğasında olmayan zorluklar

Brooks'a göre, yazılım geliştirmenin en zor kısmı, kavramları belli bir programlama dilinde düzgünce ifade etmek (kodlama) ya da bu ifadenin doğruluğunu kontrol etmek (test etme) değildir. Bunlar, yazılım mühendisliğinin rastlantısal kısmıdır. Brooks, yazılım mühendisliğinin özünü, oldukça kusursuz ve ayrıntılı bir iç içe geçmiş kavramlar kümesinin tanımı, tasarımı ve doğrulaması (verification) üzerine yapılan çalışmaların oluşturduğunu söylemektedir. Yazılım geliştirmeyi zor yapan şeyler, kendi özsel karmaşıklık, uygunluk, değişebilirlik ve görünmezliğidir [Brooks, 1986].

Bunlara kısaca değinmek gerekirse:

Karmaşıklık: Yazılım, insan yapımı şeylerden daha karmaşıktır. Gerçek-dünya ilişkilerini tanımlamak, istisna durumları tespit etmek, bütün durum değişimlerini öngörmek, kavramları geliştirmek, yanlışları ayıklamak v.b. işleri yapmak bir programlama diliyle çözülebilecek zorluklar değildir. Bu karmaşıklık, ürünü anlamayı zorlaştırır. Aslında büyük bir proje, bir kişi tarafından bütünüyle anlaşılmaz. Bu, takım üyeleri arasındaki iletişimi bozar, zaman ve maliyet kaybına, tanımlamalarda hatalara yol açar. Bu karmaşıklık, yönetimi de etkiler. Yönetimin doğru bilgi alamaması personel ihtiyacını belirlemede, bütçeyi belirlemede zorluklara neden olur.

Uyumluluk: Yazılımın içsel karmaşıklığının yanında bazı kısıtlara (eldeki donanım, yasal düzenlemeler, eski veri yapıları, vb. gibi kısıtlar) uygun olarak yaratılması gerekliliğinden doğan ek bir karmaşıklığı vardır. Yazılım, var olan sisteme uyumlu olmak zorundadır, sistem yazılıma değil.

Değişebilirlik: Yazılımı değiştirmek için sürekli bir baskı olacaktır. İşe yarar bir yazılımın değişikliğe uğramasının sebepleri vardır:

• Yazılım gerçeğin bir modelidir, bu nedenle ya gerçeğe adapte olur ya da yok olur.

• Yazılım faydalı ise, kullanıcılardan fonksiyonalitesini artırması için bir baskı olacaktır.

• Yazılımı değiştirmek donanımı değiştirmekten çok daha kolaydır.

• Başarılı yazılımlar, değişen donanım karşısında modifiye edilmelidir.

Bir program başarılı olduğu ölçüde yeni kullanım alanları bulacak ve ilk olarak amaçlandığı alanın ötesindeki alanlara adapte edilecektir. Tüm bu nedenlerden dolayı yazılım, değiştirilmek zorundadır ve bu sürekli değişikliklerin yazılımın kalitesi üzerinde zararlı etkileri vardır.

Görünmezlik: Bu özelliği yazılımı anlamayı ve üyeler arasındaki iletişimi zorlaştırır. Data akış diyagramları, modül bağlantı diyagramları programı gözünde canlandırmak için faydalıdır. Bu diyagramlar, müşteri ve mühendisler için iyi bir iletişim kaynağıdır. Problem ise; bu diyagramların hiçbir zaman ürünü tam olarak cisimlendirememeleri ve de neyin gözden kaçtığını tanımlayamamalarıdır. Problemin bir başka yönü de yazılımın durgun haldeyken bir anlam ifade etmemesidir; yalnızca çalışır durumdayken anlamlıdır. Yani anlamsızca karmaşık geometrik gösterimler bile, zaman boyutunu ihmal ettiklerinden, yazılımın yapısını olduğundan daha basit göstermektedirler [Brooks, 1986].

Yine de Brooks'un bu yapı üzerine birkaç önerisi vardır:

• Yapmak yerine al,

• Gereksinimlerin arındırılması ve hızlı prototip,

• Artarak geliştirmek - yeniden yapmak değil,
• Büyük tasarımcıların gelişimini desteklemek [Brooks, 1995].

Brooks'a göre yazılımın en zor safhaları gereksinimleri belirleme ve tasarım aşamalarıdır, kodlama değildir.

Brooks, yazılım mühendisliğinin özüne ait olmayan kavramlar üzerine yapılan sorgulamadan elde ettiğimiz kazanımların zaten elde edildiğini söylüyor: üst düzey (high-level) programlama dillerinin icadı, toplu (batch) işlemden interaktif (etkileşimli) işleme geçilmesi ve güçlü entegre ortamların geliştirilmesi. Daha öte gelişimlerin ise ancak yazılımın kendi özsel problemlerinin (karmaşıklık, uyumluluk, değişebilirlik ve görünmezlik) incelenmesiyle sağlanabileceğini belirtmiştir [McConnel, 1999].

BÖLÜM 2: YAZILIM GELİŞTİRME SÜRECİ

Yazılım süreci, bir yazılım ürününün üretilebilmesi için gerekli olan bir çok aktiviteler topluluğudur. İdeal bir proses yoktur ve her farklı organizasyon yazılım geliştirme için farklı yaklaşımlar geliştirmiştir. Prosesler, organizasyondaki insanların yeteneklerine, geliştirilecek olan projenin özelliklerine göre geliştirilmektedirler. Bu nedenle aynı şirkette farklı projeler için farklı prosesler olabilmektedir [Sommervile, 2000].

Yazılım sürecince kullanılacak modelin seçimi, proje yönetiminin en önemli kararlarından biridir [Kettunen et al, 2005, s. 47, 587-608]. "Bir prosesin olması hiç olmamasından iyidir ve birçok durumda hangi prosesin kullanıldığı nasıl yerine getirildiğinden daha az önemlidir " [Natarajan, 2004]. Yine de, uygun bir modelin seçilmesi birçok problemden projeyi koruyabileceği gibi, yanlış bir seçim ek problemler çıkartabilecektir [Kettunen et al, 2005, s. 47, 587-608].

Farklı proseslere rağmen, tüm yazılım süreçlerinde ortak olan aktiviteler vardır.

Bunlar:

1. Yazılım özelliklerinin belirlenmesi; yazılım ihtiyaç ve kısıtlarının saptanması,
2. Yazılımın tasarımı ve gerçekleştirilmesi,
3. Yazılımın geçerliliğinin sağlanması ve teslim edilmesi; müşteri ihtiyaçlarının karşılanıp karşılanmadığının test edilmesi ve teslimi,
4. Yazılım bakımı; müşteri ihtiyaçlarının değişikliğine göre ya da oluşan herhangi bir yanlışlığa karşı yazılımın bakımı [Sommervile, 2000].

2.1 Yazılım Geliştirme Süreci Modelleri

Yukarıda belirtilen aktivitelerin izleniş sırası ve geri dönüşlerin yapılması yöntemlerine göre birkaç farklı yöntem geliştirilmiştir. Bunlara yaşam döngüsü modelleri adı verilir.

Bu modellerden bazıları şu şekildedir [Kettunen et al, 2005, s. 47, 587-608]:

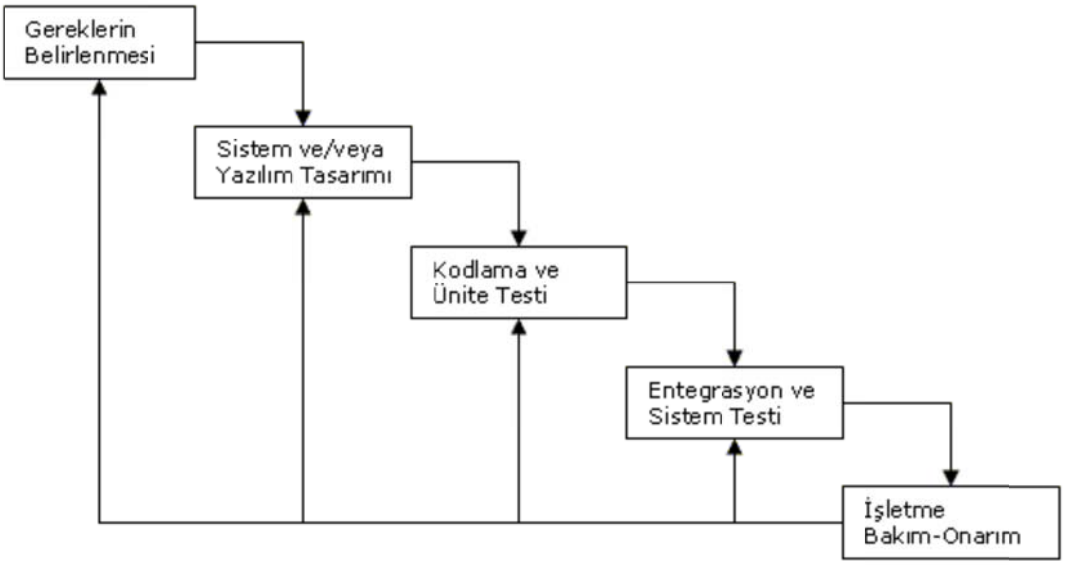
1. Disiplinsiz (Ad hoc)
2. Şelale(Waterfall) Modeli
3. Evrimsel Süreç Modelleri
    a. Artımsal (Incremantal) Model
    b. Spiral Modeller
    c. Rasyonel Bütünleştirme Süreci (RUP - Rational Unified Process)
4. Çevik (Agile) Metodolojiler
    a. Özelliğe Dayalı Geliştirme Modeli (FDD - Feature-Driven Development)
    b. XP (Extreme Programming)

2.1.1 Disiplinsiz (Ad hoc)

Bu yaklaşım genellikle kaotik, düzensiz ve plansızdır veya planlandıkları zaman kolayca bozulabilirler. Tahminler, zamanlamalar yapılmasa bile pratikte çok az gerçekleşirler [Karadağ, 2002].

2.1.2 Şelale (Waterfall) Modeli (Royce 1970)

İlk yayınlanan yazılım geliştirme süreç modelidir. Şekil 1'de gösterildiği gibi her evre, ihtiyaç analizi, tasarım, kodlama ve ünite testi, entegrasyon ve sistem testi, işletme ve bakım - onarım safhalarını içermektedir.



Şekil 1. Şelale Model İşleyişi [Gül, 2006]

İhtiyaçların analizi yapılır, gözden geçirilir ve onaylanır. Tasarım tanımlanır, gözden geçirilir ve onaylanır. Ve bu şekilde devam eder.Bir sonraki aşama, bir önceki bitmeden başlamamaktadır. Örnek olarak, ihtiyaçlar safhası tamamlanıp tasarım safhasına geçildiği zaman, yazılım gereksinimlerinin tamamen belirlendiği kabul edilmekte ve gereksinimler sabitlenmektedir. Bu da genelde gerçeği yansıtmamaktadır. Odaklandığı bir diğer nokta, dokümantasyondur [Pauca, 2003].

Bu modelde, dokümanları üretmek, iterasyonlar çok belirgin tekrar işler (rework) içerdiğinden çok maliyetli olmaktadır. Bu nedenle problemler son aşamaya bırakılmaktadır. Bu da müşterinin ihtiyaçlarına cevap vermeyi zorlaştırmaktadır.

Projeyi bu modelde olduğu gibi esnek olmayan safhalara ayırmak doğru değildir. Bu nedenle Şelale modelinin müşteri ihtiyaçları tam ve çok iyi anlaşıldığında kullanılması tavsiye edilmektedir. Sonuç olarak büyük sistemlerde, mühendislik projelerinde kullanılmaktadır [Sommervile, 2000].

2.1.3 Evrimsel Süreç Modelleri

Çoğu yazılım projelerinde evrimsel bir süreç işlemektedir. Safhalar tekrarlanarak yazılım ürünü sürekli geliştirilmektedir. Sistemin küçük bir parçası başlangıçta ortaya çıkarılmakta ve sonradan kısa aralıklarla yayınlanan versiyonlarla geliştirilmektedir.

3 problemi vardır:

• Yöneticiler ilerlemeyi görmek için rapor istemekte, fakat sistem çabuk geliştirildiğinden her versiyonu için doküman hazırlamak maliyetli olabilmektedir.

• Sürekli değişim yazılım mimarisini bozmakta, değişiklikleri birleştirmek zor ve maliyetli olabilmektedir.

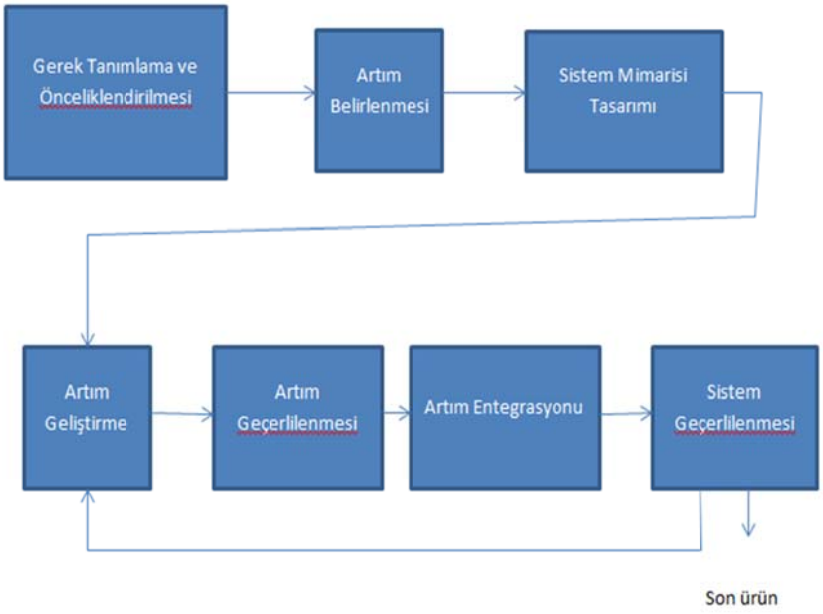
• Özel araçlar ve teknikler gerektirebilmektedir. Hızlı gelişim için gereken bu araçlar, diğerleriyle uyumlu olmadıkları için bunları kullanacak insan yeteneğine ihtiyaç duyulmaktadır.

Küçük (<=100.000 kod satırı) ve orta sistemler(<= 500.000 kod satırı) için önerilmektedir (Sommervile, 2000).

Bu yaklaşım ile ilgili farklı 3 model öne çıkmaktadır. Bunlar Artımsal Model, Spiral Model ve RUP'tur.

2.1.3.1 Artımsal (Incremantal) Model (Mills 1980)

Artımsal modelde, tek teslimatta tüm sistemi teslim etmektense, sistemi fonksiyonel birimlere ayırtıp, teslimatları artımsal fonksiyonel birimler halinde yapmak tercih edilir. Şekil 2 'de de bu modelin süreç şeması görülmektedir.



Şekil 2. Artımsal Model İşleyişi [Gül, 2006]

Avantajları arasında;

• Müşteri, değer almak için tüm sistemin bitmesini beklemez. Kritik ihtiyaçlarını karşılayan ilk geliştirme ile yazılım kullanılmaya başlanır.

• Müşteri, prototipleri kullanarak deneyim kazanır ve gelişimin devamı için ihtiyaçlarını bildirir. Müşteri, süreç içerisinde daha fazla yer alır.

• Daha az risklidir. Başarısızlıkların tüm projeye yansıması engellenmektedir.

• Öncelikli servisler ilk verilir ve diğerleri daha sonra entegre edilir.

Müşterinin sistemin önemli parçalarında hata ile karşılaşma olasılığı çok düşüktür. Çünkü önemli parçalar en çok teste tabi tutulan bölümlerdir [Sommervile, 2000].

2.1.3.2 Spiral Model (Boehm 1988)

Modeli oluşturan her döngü (sistemin olabilirliği, ihtiyaç tanımlama, tasarım,...)

• amaçların, sınırların tanımlanması,

• risklerin belirlenmesi,

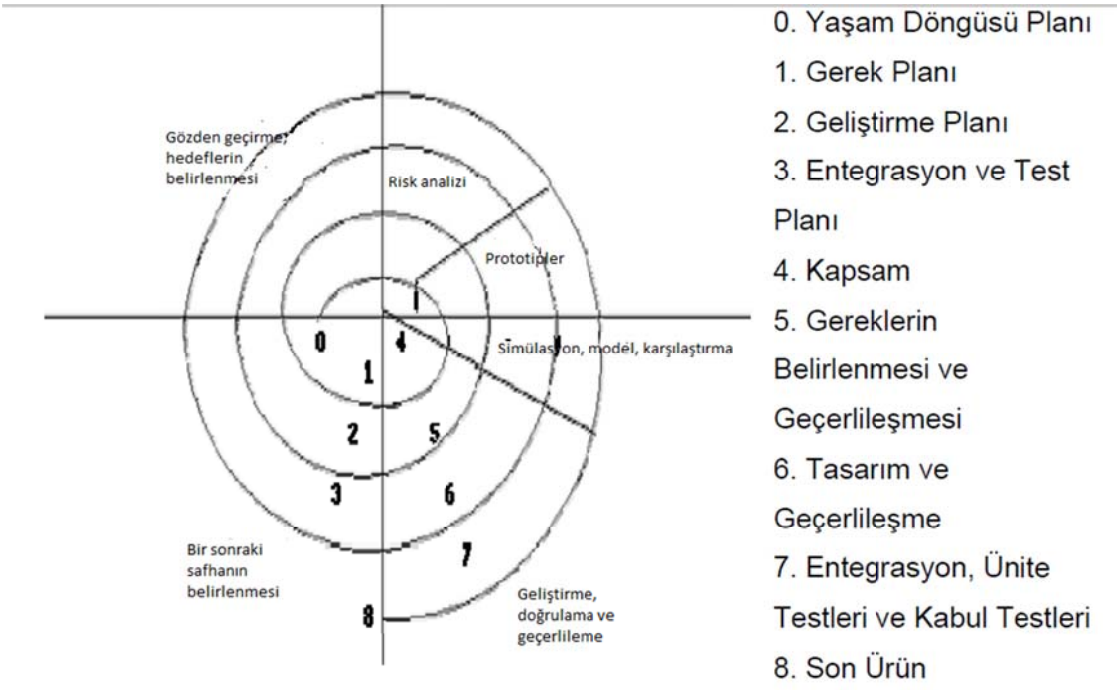
• gelişim ve geçerliliğin sağlanması,

• bir sonraki seviyenin planlanması,

aşamalarına tabi tutulur.

Odaklandığı nokta, bir sonraki riskleri azaltmaktır. Risk yönetimi kullanılması, farkını oluşturur [Sommervile, 2000].

Şekil 3'de yer alan spiral modelin her bir turu, süreçte yer alan herhangi bir safhayı temsil edebilmektedir.



Şekil 3. Spiral Model İşleyişi [Gül, 2006]

2.1.3.3 Rasyonel Bütünleştirme Süreci (RUP-Rational Unified Process)

RUP, çapraz- fonksiyonel projelerle çok iyi çalışır. RUP en iyi altı uygulama içerir:

• Yazılım değişikliklerinin kontrolü

• Tekrarlamalı yazılım geliştirme

• Bileşen temelli mimari kullanma

• Görsel modelleme yapma

• Kaliteyi test etme RUP, bir süreç üst yapısıdır ve hem geleneksel hem de hafif biçimde kullanılabilir; esnektir. Aslında yazılım projelerinin yönetimi için geliştirilmesine rağmen esnek tasarımı RUP modelini büyük e-İş dönüşüm projeleri için uygulanabilir kılar [Karadağ, 2002].

2.1.4. Çevik Metotlar

Yazılım süreç modellerinde günümüzdeki tercih edilen durum daha adapte edilebilir ve daha esnek çalışma yollarıdır. Kesin tanımlanmış büyük organizasyonel proseslerden taslağı çizilebilir, esnek, çevik proseslere hareket başlamıştır [Kettunen et al, 2005, s. 47, 587-608].

Çevik, projelerde insan odaklı bir yaklaşımı içerir. İnsanların değişime etkin bir şekilde karşılık vermesini sağlar. Bu da bu projeden fayda sağlayacak olanların ihtiyaçlarını karşılayabilen çalışma ortamlarının yaratılmasına imkan verir. Bu kategoride süreçler üst düzeyde tanımlanır. Genellikle sinerjik çözümler veya felsefeler olarak sunulur. Özelliğe Dayalı Geliştirme Modeli (Feature Driven Development) (Palmer and Felsing 2002) ve XP Metodolojisi bu kategorideki süreçlere birer örnektir [Karadağ, 2002].

2.1.4.1 Özelliğe Dayalı Geliştirme Modeli (FDD - Feature Driven Development)

FDD, modelce yönlendirilen kısa tekrarlamalı bir yazılım geliştirme sürecidir. FDD, 5 süreci içerir:

- Bütünüyle bir modeli geliştirme
- Özellik listesini oluşturma
- Özelliklerin planı
- Özellikleri tasarlama
- Özellikleri gerçekleştirme

FDD işlevlerinin basitliğinin bir yararlı özelliği, yeni elemanların kolaylıkla ekibe katılabilmesidir. Bu yöntem, sık ve dikkate değer sonuçlar üretir. İlave olarak, FDD planlama stratejilerini içerir ve hatasız bir ilerleme izleme sağlar [Karadağ, 2002].

2.1.4.2 XP (Beck 1999)

XP Metodoloji; "Hafif" metodolojilerden en iyi bilinendir. XP, küçük, aynı yerde yerleşik takımlarda çok iyi çalışır. Küçük, birlikte çalışan proje takımları (4-10 kişilik) için uygundur (Kettunen ve Laanti, 2005, s. 47, 587-608).

XP'nin temel uygulaması, tekrarlama içeren hızlı uygulama geliştirme (Rapid Application Development) metotlara benzer.

Bu iki haftalık dilimleri, sık güncellemeleri, teknik özellikleri ve işin bölünmesini içerir.

Dolayısı ile, müşteri ile güçlü bir ilişki önemli bir gerektir.

XP metodolojinin 4 anahtarı vardır:

• İletişim
• Geri Bildirim
• Basitleştirme
• Cesaret

XP programcıları müşterileri ve takım üyeleri ile devamlı iletişim halindedir. Tasarımlarını basit tutarlar. Başlanılan ilk gün yazılımlarını test ederek geri bildirim (feedback) alırlar. En kısa zamanda sistemi müşteriye teslim ederler ve öneriler doğrultusunda değişiklikleri uygularlar. Bu temel düşünce ile XP programcıları cesurca değişen ihtiyaçlara ve teknolojiye cevap verebilirler.

XP ile diğer metodolojiler arasında gözlenen bir diğer fark onun testlerdeki gücüdür. Test bütün geliştirmelerde temel noktadır. XP Programcıları yazılım kodu geliştirirken test programları da yazmak zorundadır.

XP, küçük geliştirme ekiplerinin, çabuk ürün verme ve değişimleri için tasarlanmıştır. XP küçük ve aynı yerdeki geliştirme ekibinin günümüzün hızlı geliştirme ortamında etkin biçimde çalışmasını sağlayacak minimum uygulama kümesi sunar [Karadağ, 2002].

Özetlersek, her bir metodoloji farklı bir düşünce kümesine karşılık gelir. *Ad hoc metodu*, kovboylar gibi az bir rehberlik ile bağımsız çalışmayı tercih eden geliştiricilere hitap eder. Bazen bu yö ntemle iyi çalışmalar yapılsa bile gerçekte çoğunlukla ekip çalışmasına uygun değildir, başarı oranı düşüktür. *Şelale modeli*, anlaşılması ve yönetimi basit bir yaklaşım arayan yöneticiler için uygundur. Özellikle sıkı düzenlemelerin ve bürokrasinin yoğun olduğu sektörlerde kullanılır. *Evrimsel metot* ise bürokrasiye dayanıksız fakat risklere de açık süreçlerde

kullanılır. Tekrarlamalı süreç geliştirme riskleri azaltır, fakat yönetimi zordur. *Çevik yöntem* ise nispeten daha yeni bir yaklaşımdır [Karadağ, 2002].

Kompleks projelerde ise tek bir model seçmek en iyi yol değildir. Hibrid modeller, farklı modellerin özelliklerini dengelemek daha iyi bir seçenektir. Ürünün farklı parçalarının değişen karakteristiklerine bağlı olarak farklı modeller kullanılabilmektedir. Örneğin; kullanıcı ara yüzü parçası çevik modellemedenfaydalanırken, daha sabit öz parçaları ise şelale modelini izleyebilir [Kettunen et al, 2005, s. 47, 587-608].

BÖLÜM 3: YAZILIMIN TEMEL AKTİVİTELERİ

Farklı organizasyonlar yazılım üretiminde farklı yolları kullanırlar. Bazı organizasyonlar yazılımın kaynak kodunun okunarak anlaşılabileceğini düşünürlerken, bazıları da dokümantasyona çok önem verirler. Tasarımdan önce planlar yapar, kodlamadan önce tasarımı tekrar tekrar kontrol ederler. Bakım aşamasında herhangi bir değişiklik isteği için geçerli nedenler ararlar ve yapılan değişikliği dokümana aktarmadan ürüne entegre etmezler.

Test safhası ise diğer karşılaştırma ölçüsüdür. Bazı organizasyonlar, bütçelerinin yarısını yazılımın testine ayırırken, bazı organizasyonlar ise ürünü sadece kullanıcının test edebileceğini düşünürler. Teste ayrılacak zamanı ve eforu minimal etmeye çalışsalar da, kullanıcıların gösterdiği hataları tespit etmek için bir hayli zaman harcamak zorunda kalırlar.

Yazılım süreçleri genel olarak gereksinim analizi, tasarım, kodlama, test, teslim, bakım aktivitelerinden oluşur. Test ayrı bir safha değil, tüm süreç boyunca yer alan bir aktivite olarak bakılmalıdır [Schach, 1993].

Tüm aktiviteler, kullanılan araçlar, ölçümler ve yapılan testler çerçevesinde incelenmiştir.

3.1 İhtiyaç Analizi

Sistemden ne istenildiği ve kısıtların belirlendiği aşamadır. Bu aktivite "Gereksinim Mühendisliği (Requierements Engineering)" olarak da adlandırılmaya başlanmıştır. Kritik bir aşamadır. Yazılım geliştirme sürecinin başında olması nedeniyle bundan sonraki tüm süreçleri de doğrudan etkilemektedir [Sommervile, 2000].

Sistem gereksinimleri tam ve doğru olarak ortaya konulamaz ise, daha sonraki süreçlerde bu aşamadan dolayı ortaya çıkacak problemlerin maliyeti göreceli olarak artacak ve ortaya çıkan yazılım kaynaklar açısından verimli

olmayacaktır. Bu sürecin mühendislik şeklinde ifade edilmesinin nedeni ise gereksinimleri elde etme esnasında ve analiz işlemlerinde sistematik yaklaşımların yer almasıdır.

Bu önemine rağmen hala yazılım geliştiriciler bu sürece daha az zaman ve emek harcayarak doğrudan geliştirme sürecine geçmekte, müşteri gereksinimleri dokümanlarını yapısal ve düzenli biçimde hazırlamamaktadırlar [Şen, 2005].

3.1.1 Gereksinimler

İhtiyaçları toplamak ve üzerinde birliğin sağlanması başarılı bir projenin temelidir. Bütün ihtiyaçların tasarım, kodlama aşamasından önce sabit olması gerekli değildir, fakat yazılım geliştiren takım için hangi ihtiyaçların inşa edileceği önemlidir [Natarajan, 2004].

Müşteri ile uzlaşılmış tüm istekler ve ortaya çıkabilecek tüm kısıtlar gereksinim olarak düşünülmelidir. Geniş bir aralıkta değerlendirilen gereksinimler, üst seviyedeki genel isteklerden yazılımın fonksiyonel detaydaki belirtimlerine kadar dağılım gösterir. Sonuçta hepsinin gereksinim olarak ele alınıp analizinin yapılması gerekir. Gereksinimler çok değişik şekilde sınıflandırılıp, farklı tiplere ayrılabilir.

3.1.2 Gereksinim Tipleri

a) Kullanıcı gereksinimleri: Doğal diller, tablolar, diyagramlar kullanılarak tanımlanabilen, müşteriler için hazırlanan ihtiyaçlardır. Kullanıcı senaryoları (use-case) için temel oluşturur. Kullanıcıdan alındıkları için teknik anlamda detaylar içermeyebilir. Bir takım problemler çıkabilir (gereksinimlerin yeterince açık ifade edilememesi, fonksiyonel ve fonksiyonel olmayan olarak ayırt edilememesi, birçok ihtiyacın tek bir ihtiyaç gibi ifade edilmesi,..gibi).

b) Sistem gereksinimleri: Sistemin servislerinin detaylı tanımlamaları ile donanımsal yapılara özgü gereksinimlerdir. Belirsizlikleri ortadan kaldırmak için

yapısal dillerle tanımlanmalıdır. Bu dil, yüksek seviyeli bir programlama dili ya da gereksinimlerin belirlenmesi için özel bir dil olabilir.

Gereksinimler etki alanlarına göre de 2 tipe ayrılırlar:

İşlevsel (Fonksiyonel) gereksinimler: Sistemin ne yapacağını yapısal ve işlevsel olarak ortaya koyarlar. Geliştirmeden bağımsız çoğunlukla giriş, çıkış arabirimleri, süreçler ile hata yönetimine yönelik gereksinimlerdir. Sistem girişindeki izin verme ve yetkilendirme gereksinimleri de bu tiptedir. Sistemin neler yapacağını soyut olarak değil de detaylandırılmış biçimde belirler.

İşlevsel olmayan (fonksiyonel olmayan) gereksinimler: Sistemin daha çok kısıtları ile fiziksel ortam, ara yüzler, kullanıcı odaklı olma, güvenlik, güvenilirlik, kalite güvence gibi soyut niteliklerini belirleyen gereksinimlerdir. Yazılımlara işlevsellik katmamasına rağmen bu tip gereksinimler özellikle yazılım kalitesi açısından kritik rol oynarlar. Bu gereksinimler yazılımda karşılanmadığı sürece yazılımın kullanılabilirliği yetersiz kalacaktır.

Örneğin, güvenilirlik işlevsel olmayan gereksinimi için yazılıma birkaç işlev ile hata raporlama yapılarının kurulması gerekecektir ve ancak bu şekilde yazılımların güvenilirliği sağlanacaktır. Gerçekte, bu gereksinim tipleri arasında kesin bir fark çizgisi yoktur [Sommervile, 2000].

3.1.3 Gereksinim Zorlukları

Gereksinimleri elde etmenin çeşitli zorlukları mevcuttur. Bu zorluklar şöyle sıralanabilir:

Uygulama alanına yabancı olmak: Yazılım geliştiriciler çok çeşitli alanlara hizmet sunmaktadırlar ve geliştirilecek alanda bilgi birikimi sahibi olunmaması durumunda, doğru ve eksiksiz gereksinim elde etmek zordur. Bu zorluğun üstesinden gelmek için piyasa araştırması yapılmalı veya mümkün olduğunca bileşen tabanlı çözümler üreterek eldeki çözümler yeni sisteme uyarlanmalıdır.

Uygulama alanının kompleks olması: Uygulama geliştirme alanı ne kadar kompleks ve büyük olursa, ortaya o kadar fazla gereksinim ve bu gereksinimler arasındaki ilişkiler çıkar. Kompleks sistemleri anlamanın yanı sıra bu sistemler arasında ilişkileri kurmak da oldukça zordur. Bu durumda önce sistemleri modüllere ayırmak sonra da bu modüller arasındaki ara yüzleri çıkartmak, gereksinimleri elde etmeyi kolaylaştıracaktır.

Tüm sistemi bir kişinin bilmesi: Organizasyon altyapısı oturmamış bir kuruluşa yazılım geliştirileceği zaman çoğunlukla sistemi bütünüyle bilen bir kişiyle karşılaşılır. Bu kişi genelde o kuruluşta yıllardır çalışan ve eski sistemin mimarlarından birisidir. En doğru bilgi bu kişilerden alınabileceği için, bu kişilerden gereksinimleri elde etmek için planlı toplantılar yapmak ve zamanı verimli kullanmak gerekir.

Müşterilerin bilgilerini ortaya koy(a)maması, gereksinimlerin tutarsız ve eksik olması, yanlış anlamalar: Kullanıcılar, kimi zaman yeni sisteme karşı bir tutum izleyerek bilgi sağlamayarak zorluklar çıkarabilirler. Bazen de dar bir bakış açısından sistemin bütününe bakmak durumunda olan kişiler, gerçek gereksinimleri göz ardı ederek detaylarda boğulurlar ve yeterince verimli olamazlar. Bu durumda sistemle ilgili tüm kişi ve grupları gereksinim sağlayıcı olarak seçmek ve her kişiye uygun sorular sormak gerekir. Farklı bakış noktaları, sistemin tüm paydaşlar tarafından nasıl algılandığı konusunda ipucu verir ve sistemin analizini kolaylaştırır.

Gereksinimlerin farklı bakış açılarından elde edilmesi durumunda benzer gereksinimler arasında uyuşmazlık ve tutarsızlık ortaya çıkabilmektedir. Bunu önlemek için gereksinimleri, kontrol listelerini kullanarak sınama ve doğrulama yapmak gerekir. Bu şekilde gereksinimler arası tutarsızlıklar ortaya çıkarılabilir.

Kimi zaman gerek analist-müşteri ve gerekse analist-geliştirici arasında yanlış anlamalar olmakta ve ortaya beklenmeyen sonuçlar çıkabilmektedir. Bunu önlemek için sık sık müşteriye geri bildirimlerde bulunmak ve hatta yapılabiliyorsa müşteri temsilcisinin de yazılım ekibinin bir parçası olmasını

sağlamak gereklidir. Müşteriden erken geri dönen uyarılar ve bildirimler, ileride oluşabilecek yüksek maliyetlerin önüne geçilmesini sağlayacaktır [Şen, 2005].

3.1.4 Gereksinim Amaçları

Gereksinimler, 3 amaca hizmet ederler:

Geliştiricilere, müşterinin sistemi nasıl çalışmasını görmek istediklerini anlamalarını sağlar.

Tasarımcılara, sonucun hangi fonksiyonlarının ve özelliklerinin olması gerektiğini anlatır.

Test takımına, müşteriye istediklerinin gerçekleştiğini göstermeye yarar [Pfleger, 1991].

3.1.5 Gereksinim Özellikleri

İhtiyaçlar;

• Doğru,

• Çelişkisiz, birbirini tutan,

• Bütün,

• Gerçekçi (mümkün olup olmadığına bakılmalı) Örneğin; donanımın yeteri kadar hızlı olması veya kullanılan disk ambarının yeni ürünü kaldıracak kapasitede olması gerektiği gibi [Schach, 1993].

• Müşterinin isteği ve

• Doğrulanabilir,

olmalıdır [Pfleger, 1991].

3.1.6 Gereksinim Süreçleri

4 ana süreçten oluşmaktadır:

1. Olurluk (fizibilite) çalışması: Müşteri gereksinimlerinin, uygun bütçe ve teknoloji sınırları içinde olup olmadığının tespit edildiği süreçtir. Müşterilere, fazladan istenilen hizmetlerin bir ek maliyeti olduğu ve bunun hem zaman hem de maddi açıdan yeterli olması gerektiği belirtilmelidir. Bazen de eldeki teknolojinin de yetersiz kalması ortaya çıkabilir. Tüm gereksinimler için olurluk çalışmasının

yapılması gereklidir. Bunun sonucunda fizibilite raporu hazırlanmalıdır. Değişik biçimlerde olurluk çalışması yapılabilir: teknik olurluk (kaynak, teknoloji vb.), ekonomik olurluk, yasal olurluk ,alternatifler.

Olurluk çalışmasında ayrıca karlılık-maliyet analizinin de çıkarılması ve belli kriterlere göre karar verilmesi gereklidir. Her bir hizmet ve araç için maliyetlerin ortaya konulması ve beklenti, risk ve öncelik değerleri ışığında karar verilmesi gerekir.

2. Gereksinim analizi: Müşterinin ve tüm paydaşların sistemden olan beklentilerini ortaya çıkarmak ve bunları uygun metodolojilerle ayrıştırıp arındırmaktır. Özelikle çatışma yönetiminin bu süreçte yapılması ve varsa tutarsızlıkların çözümlenmesi gereklidir. Gereksinimlerin önceliklendirilmesi de bu süreçte yapılır. Müşteri bakış açısına göre tüm gereksinimler aynı öncelik derecesine sahip görülebilir. Ortada bir proje planı ve bütçe olması nedeniyle bazı gereksinimlerin daha fazla önceliğe sahip olması gerekecektir.

Gereksinim tanımlama: Gereksinimleri müşterinin anlayabileceği formlara dönüştürerek gerekli tanımlamaları yapmaktır. Bu süreçte diyagram veya formlar kullanılır ve görsel anlamda yapısal doküman hazırlanır. Müşteriye yönelik süreçtir.

Ayrıntılı gereksinim belirtimi: Gereksinimleri detaya inerek tanımlama ve ortak olan ve olmayan noktaları ortaya koyma sürecidir. Yazılım tasarımına giriş yapılır. Bu süreç daha çok uygulama geliştiriciler ile müşteri ortamındaki teknik paydaşlar içindir. Gereksinim tanımlama sürecinden ortaya çıkan formların her biri için detaylı formlar ve diyagramlar oluşturulur. Yazılımın tasarımında tanımlanacak nesneler için soyut tanımlamalar yapılır ve üst düzeyde veri akışı diyagramı oluşturulur.

Gereksinimleri geliştirmenin bir yöntemi ise prototip oluşturma ve bu prototipi müşteriye sunarak daha önceden alınan gereksinimlerin eksiklerini tamamlamak ve geliştirmektir [Şen, 2005].

Prototip: Programın ne yapabileceğini görmek için küçük çalışan bir modelinin geliştirilmesi. Özellikle kullanıcıların sistemden ne istediklerini tam bilmedikleri durum için uygundur.

Keen'e göre kullanıcılar sistemin ne faydalar ne özellikler barındıracağını operasyonda görmeden bilemeyecekleri için prototip onların denemelerini sağlar. Lantz'a göre daha az performanslı, Alavi'e göre büyük projelerde uygulaması zor, Ivari ve Karjalar'a göre kullanıcıların bir bölümünde gerçek olmayan beklentiler yaratmaktadır [Olson, 2001].

Gordon ve Bieman (1995), 39 farklı prototip proje üzerinde yapılan çalışmalarında, yazılım sürecinde prototip kullanmanın faydalarını şu şekilde bulmuşlardır:

- Sistem kullanılabilirliğinde iyileşme,
- Sistemle kullanıcı ihtiyaçlarının daha çok örtüşmesi,
- Tasarım kalitesinde iyileşme,
- Bakımda iyileşme,
- Gelişmeye harcanan eforda azalma.

Prototip genellikle, erken safhalarda maliyeti arttırır, fakat daha sonraki safhalardaki maliyeti azaltmaya yardım eder. Bunun temel nedeni, müşterinin daha az değişiklik istemesinden dolayı geliştirme süresince tekrar işleri (rework) azaltmasıdır. Bu olumlu etkilerine karşı, prototipin negatif sonuçları arasında, tüm sistemin performansının etkisiz prototip kodunun tekrar kullanılmaya zorlanması gibi işlerden dolayı bazen aşağı çekilebileceğini bulmuşlardır.

Prototipin amaçları sürecin başlangıcından itibaren açıkça ifade edilmelidir. Amaçlar açık değilse, yönetim veya son kullanıcı prototip fonksiyonalitesini anlayamayacaktır. Sonuç olarak, istenilen faydayı alamayacaklardır. Prototipler, evrimsel (gelişmeli) olabileceği gibi, sadece sistem gereksinimlerini doğrulamak veya türetmek için de kullanılabilirler. Evrimsel (gelişmeli) prototiplerde en iyi anlaşılan bileşenlerden başlanılır, gereksinimler

netleştikçe geliştirmeler yapılır. Gereksinimlerini doğrulamak veya türetmek için hazırlanan prototiplerde ise en az anlaşılan bileşen geliştirilir [Sommervile, 2000].

Prototip için derlenmeye (compile) veya bağlanmaya ihtiyaç duymayan yorumlu dilleri kullanmak daha iyidir. Böylelikle prototipin gelişimi daha hızlı olur. Bu diller, derlenmesi gereken dillere göre daha az etkilidir ve bakımında zorlukları vardır. Sadece prototip için uygundur [Schach, 1993].

3. Onaylama ve doğrulama: Müşteri onayı olmayan gereksinimleri tasarıma girdi olarak almak, hatalara ve boşa emek harcanmasına neden olabilir. Tüm gereksinim dokümanlarının resmi olarak onaylatılması gereklidir. Bu onaylamadan sonra ortaya çıkabilecek gereksinim değişikliklerinin maliyeti ve proje planı revizyonları daha kolay olabilecektir. Gereksinimleri doğrulamak için kontrol listelerinin kullanılması gereklidir. Kontrol listesindeki tüm kriterlerin sağlandığından emin olunması ve eksikliklerin tamamlanması zorunludur.

4. Gereksinim yönetimi: İş, organizasyon ve teknik değişiklikler, gereksinim değişikliğine yol açmaktadır. Gereksinim yönetimi, bu değişikliklerin kontrolü ve yönetimi sürecidir. Müşteri görüşmelerinin ve toplantılarının düzenlenmesi ve elde edilen gereksinimlerin doğru analiz edilerek dokümantasyon edilmesi bu faaliyet kapsamındadır [Şen, 2005].

3.1.7 Gereksinim Dokümanları

Gereksinimlerin belirtildiği müşterinin anlayacağı şekilde müşterinin beklentilerinin yazılı olduğu dokümanın dışında bir de sistem tasarımcısının ihtiyaç duyduğu daha teknik bir dokumana ihtiyaç vardır. Bu dokuman da gereksinimlerin ayrıntılarıyla belirlendiği dokümandır. Bunlar birbiriyle eşleşmelidir [Pfleger, 1991].

Yazılım gereksinimleri dokümanında işlevsel ve işlevsel olmayan gereksinimlerin ayrı ayrı ele alınıp analiz edilmesi ideal olandır ama tüm sistemi etkileyen gereksinimler için bu türden ayrıştırma yapmak zorlaşır. Bu

gereksinimin hangi tipe uygun olduğuna karar verirken etki alanına ve önceliklendirilmesine bakılır.

Dokümanların izlenebilir olması gereklidir. Eğer gereksinimler, düzenli olarak sunulur, uygun bir şekilde numaralanır, çapraz karşılaştırılırsa ve indekslenirse test grubu doküman ile takip etmede ve gereksinimleri doğrulamada zorlanmaz [Schach, 1993].

3.1.8 Kullanılan Metot ve Araçlar

Her metot ihtiyaçların net belirlenmesi yolunda organize ve standartlaşmaya yardımcıdır [Pfleger, 1991].

Yazılım geliştirmede biçimsel (formal) metotları kullanmak, yazılım ihtiyaçlarının erken safhada analizini zorlar. Ancak, bu aşamada hataları bulup doğrulamak ise, teslim edilen bir sistemi değiştirmekten daha az maliyetli olacağından fayda sağlamaktadır. Metotları kullanmak, güvenilirliğin önemli olduğu kritik sistemlerin gelişiminde en çok uygulanandır. Formal tanımlama, tam ve belirlidir. Şüpheleri, problemleri ortadan kaldırır. Buna rağmen uzman olmayanlar biçimsel tanımlamanın anlamanın zor olduğunu düşünürler.

3 tane yazılım gereksinimleri belirleme seviyesi vardır. Bunlar, kullanıcı gereksinimleri (en özet tanımlama), sistem gereksinimleri ve tasarım gereksinimleridir (en detaylı tanımlama). Formal tanımlama sistem gereksinimleri ile tasarım gereksinimleri arasında bir yer almaktadır. Uygulama detaylarını içermemekte, fakat sistemin tam matematiksel modelini sunmaktadır. Formal tanımlama teknikleri en çok, kritik sistemlerin gelişimi için uygundur .

İhtiyaçların ayrıntılarıyla belirtimi için kullanılan metotlara birkaç örnek vermek gerekirse:

• HIPO çizelgeleri (hiyerarşi ve girdi-süreç-çıktı/ hierarchy and input-processoutput): Fonksiyonların nasıl ilişkilendirildiği üzerinde durur.

• Hiyerarşi veri yapısı (Hierarchy data structure): Fonksiyonlara odaklanmak yerine, dataların nasıl ilişkilendirildiği üzerinde durur.

• Data akış diyagramı ve analizi.

• Yazılım gereksinim mühendisliği metodolojisi: Gereksinimlerin hem net belirlenmesinde hem de analizinde kullanılır.

• Yapısal analiz ve tasarım teknikleri (structured analysis and design techniques).

• GIST ISI (Information Science Institute) nın geliştirdiği özel ihtiyaç belirleme araçlarından biridir [Pfleger, 1991].

Grafiksel araçlar (bir değişiklik her şeyi yeni baştan çizmeyi gerektirir, çizim araçlarının olması zaman koruyucudur) ve data sözlükleri gereksinim safhasında yardımcı araçlardır [Schach, 1993].

Bu safhada kullanılan araçlar (CASE Tools) ;

• Grafik modeller

• Genel durum diyagramları

• Veri akış diyagramları

• Durum değişikliği diyagramı ve analizi,

• Zaman ve boyut analizi,

• Nesne tabanlı modeller ve analizler,

• Veritabanı kalitesinin artırılması ve veri sözlüğü,

• Sistem simülasyonu

• Prototip model [Sodhi, 1991].

3.1.9 Kullanılan Ölçümler

Bu süreçte önemli olan, hazırlanan prototipin müşterinin gerçek ihtiyaçlarını hangi hızda belirlediğidir. Bu süreçte faydalı ölçüm, ihtiyaçların uçuculuğunu ölçmektir. Gereksinimlerin, gereksinimleri tanımlama sürecinde hangi sıklıkla değiştiğini kaydetmek gereksinim takımına gerçek ihtiyaçlara hangi oranda yaklaştığını tanımlamasına yardım eder. Gereksinim takımının işini ne kadar iyi yaptığı da yazılımın gelişiminin diğer safhalarında ne kadar ihtiyacın değiştiği ile ölçülür. Çok sayıda değişiyorsa takımın yerine getirdiği gereksinim süreci en ince ayrıntılarıyla analiz edilmelidir [Schach, 1993].

Özet olarak;

Geliştiricinin esas amaçlarından biri müşterinin ihtiyaçlarını tam karşılayan bir yazılım üretmektir. Ne yazık ki, birçok geliştirici bu amaçta başarısız olmaktadır. Yapısal, nesne yönelimli metotlara rağmen geç, hesaplanan maliyeti aşan ve sürekli bakım isteyen yazılımlar ortaya çıkmaktadır. Hatalar geliştirme sürecinin herhangi bir aşamasında olabilmektedir. Fakat bulması en zor ve en maliyetli olan hatalar erken safhalarda oluşan hatalardır. Bu nedenle ilk safhalara daha fazla önem verilmelidir.

İlk safhada tanımlamalar biçimsel (formal) olursa ve sonra gelen tanımlamalar formal yolla ifade edilirse, programın son halinin en azından niyet edilen maddeleri karşıladığına ait bir güven oluşabilir.

Gereksinimleri tanımlama, belirleme aşamasında harcanan zamanın hata ayıklamada, testte, kodlamada zaman ve para kazancı olduğu konusunda müşteri ve yöneticiler inandırılmalıdır [Britton et al, 1993].

Gereksinim analizi en önemli ve sıklıkla ihmal edilen bir aktivitedir. İyi bir gereksinim modeli, iş ile IT (Information Technology) nin uygulanacak sistem çözümlerinde ortak bir vizyonu paylaşmalarını sağlayacak iletişimi beslemektedir. Bu, sistemin iş ihtiyaçlarını karşılamasını, zamanında kaliteli ve gelecek ihtiyaçlara ayak uydurması için esnek olmasını sağlayacaktır.

3.2Tasarım

Sistem ihtiyaçlarını çalışır sisteme çevirmekte, yazılımın mimarisini oluşturmaktadır [Sommervile, 2000]. Sistem tasarımı ve program tasarımı olarak 2 bölümde incelemek mümkündür:

Sistem Tasarımı: Müşterinin yeni bir sistem ihtiyacı, yazılım geliştiricileri sistem tasarımını iki açıdan düşünmesi için zorlar. Bunlar ürün ve süreçtir. Sistem tasarımının süreci problemlerin çözüme dönüşmesidir. Sonuç ürün ise çözümün tarifidir.

Program Tasarımı: Program tasarımı sistem tasarımının kodlanabilir modüllere çevrilmesidir. Programın net belirlenmesi programcıya yönergedir [Pfleger, 1991].

3.2.1 Sistem Tasarım Süreci

Sistem tasarım süreci 2 bölümden oluşur:

a) Sistemin net belirlenmesi: Müşteriye sistemin ne yapacağını anlatır. Buna "kavramsal sistem tasarım" denir. Sistemin fonksiyonlarını müşterinin anlayacağı dilde teknik jargon kullanmadan ifade etmelidir.

b) Teknik tasarım: Müşteri kavramsal sistem tasarımı uygun görürse sistem kurucularına, programcılara yazılım ve donanım kurmalarına izin veren "teknik tasarım" hazırlanır. Teknik tasarım; sistem mimarisi (donanım parçalarının fonksiyonları), yazılım yapısı (yazılım unsurları fonksiyonları), veri (veri yapısı ve akışı) içermelidir [Pfleger, 1991].

3.2.2 Tasarım Aktiviteleri

Mimari tasarım: Sistemi oluşturan tüm alt sistemler ve birbirleriyle ilişkileri tanımlanır ve dokümante edilir.

Soyutlama: Her alt sistem için kısıtların ve işlevlerin soyutlaması yapılır.

Ara yüz tasarımı: Alt sistemlerin diğerleriyle ara yüzü tasarlanır, dokümante edilir.

Bileşen tasarımı: Hizmetler farklı bileşenlere ayrılır ve bu parçaların ara yüzleri tasarlanır.

Veri yapısı tasarımı: Veri yapısı ve akışı detaylı ve açıkça belirlenmiş şekilde tasarlanır.

Algoritma tasarımı: Algoritmalar, detaylı ve açıkça belirlenmiş şekilde tasarlanır [Sommervile, 2000].

3.2.3 Sistem Tasarım Dokümanları

Aynı sistem olmasına rağmen farklı kullanıcılara hitap ettiği için 2 farklı doküman hazırlanmaktadır. Kavramsal sistem tasarım dokümanı sistemin

fonksiyonları üzerine odaklanırken, teknik tasarım dokümanı sistemin alacağı form üzerine odaklanmaktadır [Pfleger, 1991].

3.2.4 İyi Tasarım Özellikleri

• Modülerlik (modularity): Yüksek seviyeli modülerlik, problemi bütün olarak görmemizi sağlar ve bizi başka tarafa çekecek detayları saklar.

• Soyutlama (abstraction) seviyesi: Modüllerin yukarıdan aşağı inildikçe detayları artar. Üst seviye detayları saklar. Değişiklikler modül modül yapılabilir.

• Bağlantı (coupling): Ne kadar modülün birbirine bağlı olduğunu ölçer. Hataları takip etme ve herhangi birinde yapılacak değişikliğin diğerlerini etkilememesi açısından düşük seviyeli bağlantılar iyidir.

• Bağlılık (cohesion): Modülün parçalarının birbiriyle ve modülün fonksiyonalitesi ile daha fazla birleşmesi, ilgili olması iyidir.

• Kontrol edilen modül sayısı: Modül tarafından kontrol edilen modüllerin sayısı minimal olmalıdır.

• Kontrol alanı: Bir modül kontrolünde olmayan bir modülü etkilememelidir [Pfleger, 1991].

3.2.5 Tasarım Teknik ve Araçları

Birçok yazılım projesinde tasarım hala disiplinsiz bir süreçtir. Genellikle doğal dilde tanımlanmış gereksinimlerle başlayan formal olmayan bir tasarım süreci vardır. Kodlama başlar ve tasarım kodlamalar gerçekleştikçe değiştirilir. Tasarım yönetiminde formal değişiklik ya çok az ya da yoktur. Gerçekleştirme safhası bittiğinde tasarım ilk belirlenen tanımlamalara göre çok değişmiştir ve orijinal tasarım dokümanı sistemi yanlış veya eksik tarif etmektedir.

Yapısal metotlarla (notasyonlar ve prensipler) metotsal yaklaşımlar ileri sürülmektedir. Bu grafiksel sistem modelleri ve tasarım dokümanlarını kapsamaktadır. CASE araçları bu metotları desteklemek için geliştirilmiştir. Standart notasyonlar, standart dokümanlar kullandıkları için maliyet azalmasına yardım etmektedirler.

Metotlar, standart notasyon ve geliştirilmiş pratikler içermektedir. Bunları izlemek mâkul, akla uygun tasarımlar ortaya çıkaracaktır. Yine de tasarımcının yaratıcığına hala ihtiyaç vardır.

Ampirik çalışmalar (Bansler ve Bodker) göstermiştir ki; yazılım geliştiriciler, metotları köle gibi çok nadir kullanmakta koşullara göre prensiplerden istediklerini seçmekte ve kullanmaktadırlar. Hiçbir metot bir diğerinden daha iyi veya daha kötü olamaz. Metodun başarısı, uygulama alanındaki elverişliliğine bağlıdır. Seçimi sistemin özelliklerine bağlıdır [Sommervile, 2000].

Yapısal metotlar, sistemde tanımlanmış süreç tabanlı bir metottur. Data akış diyagramları gibi yapısal tasarım da top-down yaklaşımı kullanır. Bu tasarımda kullanılan temel araç sistemdeki fonksiyonları, süreci hiyerarşik şekilde sunan ve onlar arasındaki bağlantıları, ara yüzleri sunan yapı çizelgeleri (structure chart) dir.

Nesne tabanlı tasarım popülaritesinin artması ile yapısal tasarım teknikleri gündem dışı kalmaya başlamıştır. Teoride nesne tabanlı tasarımın yapısal tasarım üzerinde avantajları vardır. Nesne tabanlı tasarım, bakımı, modifiye edilmesi kolay ve tekrar kullanma olanağı olan iyi yapılı programlar üretmektedir. Buna rağmen yapısal tasarım yıllarca kullanılmıştır. Pek çok yazılım bu yoldan üretilmiştir [Britton et al, 1993].

Nesne tabanlı tasarımın faydaları arasında şunlar gelmektedir:

• Bilgi saklama: Veriyi ve operasyonları paketlemenin avantajlarından biridir. Nesne kendisine ve operasyonlarına ait tüm detayları içinde saklamaktadır. Dataya giriş sadece tanımlı operasyonlarda görüldüğünden bakım ve değişiklik işlemleri daha kolaydır.

• Mesaj geçimi: Yapısal tasarımda veri süreç birimleri arasında geçiş yapar. Nesne tabanlı tasarımda ise kendi prosesini kontrol eden nesnelere mesajlar geçer.

• Tekrar kullanılabilirlik: Yeni nesneleri her seferinde yeni baştan tanımlamamızı önler [Britton et al, 1993].

Birçok proje için, nesne tabanlı analiz ve UML kullanarak tasarım önemlidir. Diğer bazı teknikler arasında uygulama jeneratörü gelmektedir. Rapor jeneratörleri ve ekran oluşturucular ile kullanıcı raporda gözükmesini istediği birimleri seçerek raporda görülmesini, girdi ekranında olmasını istediği birimleri seçerek ekranda görülmesini sağlayabilir. Çünkü tüm detaylar sözlükte vardır ve CASE araçları isteğe göre rapor veya ekran için kod yaratabilmektedir.

CASE araçları ayrıca, veri sözlüğündeki her kaydın tasarım dokümanında olup olmadığını veya tasarım dokümanındaki her kaydın veri akış diyagramını yansıtıp yansıtmadığını kontrol edebilmektedir [Schach, 1993].

Ayrıca, organizasyonlar tekniklerin kullanımı için kısa süreli danışmanlar kullanırlarsa, maliyet açısından projenin ileriki aşamalarında fayda sağlayabileceklerdir [Natarajan, 2004].

3.2.6 Tasarım Testi

Bu safhadaki testin amacı tanımlamaların doğru ve tam olarak tasarımda birleştirilmesinin doğrulanmasıdır. Örneğin; mantık hataları içermemesi, tüm ara yüzlerin doğru tanımlanması gerekmektedir. Kodlamadan önce hataların bulunması önemlidir, aksi takdirde hataları düzeltme maliyeti daha yüksek olacaktır. Tasarım safhasının en kritik noktası; tasarım dokümanının, gereksinim dokümanı ile tam örtüşmesidir [Schach, 1993].

3.2.7 Kullanılan Ölçümler

Bu safhada, birçok ölçüm kullanılabilir. Modül sayısı, basit olarak hedeflenen ürünün boyut ölçüsünü verir. Modül bağlantıları ve bağlılıkları, tasarımın kalitesinin ölçüleridir. Tasarımın kontrolü sırasında bulunan hataların sayısının ve tipinin kaydedilmesi de çok önemlidir. Bu bilgiler daha sonra ürünün kod kontrolü sırasında ve peşinden gelen ürünlerin tasarım kontrolleri sırasında kullanılabilmektedir [Schach, 1993].

3.3 Kodlama

Toplam proje çabasının çok küçük bir parçası olmasına rağmen, en çok görülen kısımdır [Natarajan, 2004]. Ürünlerin büyük olması ve zaman kısıdının olması birkaç programcının ürünün farklı parçaları üzerinde aynı zamanda çalışmasına yol açmaktadır. Programcıların asıl amacının tasarımı koda çevirmek olmasına rağmen, zayıf işbirliği ve geliştiriciler arasındaki zayıf iletişim zarar görmelerine neden olmaktadır.

Takım organizasyon problemlerini çözmek için tek bir yol yoktur. Takım organize etmenin en uygun yolu; ürüne, önceki deneyimlere, organizasyon yöneticilerinin bakış açısına, hedeflerine bağlıdır [Schach, 1993].

Programlamaya başlamadan önce, çalışılan firmadaki standartlar ve prosedürler bilinmelidir. Program kodlarının başka programcılar tarafından da değiştirilebileceği düşünülerek bazı prosedürler konulmuş olabilmektedir [Pfleger, 1991].

3.3.1 Kodlama Araçları

Tasarımdan sonra iskelet program için yazılım mühendisliği araçları (CASE tools) kullanılabilmektedir. Araçlar, ara yüzlerin tanımlanması ve uygulanması için kodları içermekte, geliştiriciler ise bazı detayları ekleyebilmektedir [Sommervile, 2000].

Ayrıca, yüksek seviyede yazılmış kodu makine koduna çeviren ve çalıştıran derleyiciye (compiler), çalışma zamanında birçok modülü birbirine bağlayan bağlayıcıya (linker), hafızaya ürünün çalışan versiyonunu yüklemek için ise yükleyiciye (loader) ihtiyaç vardır [Schach, 1993].

3.3.2 Programlama Dilleri

3.3.2.1 Programlama Dillerinin Sınıflandırılması

Programlama dillerini çeşitli açılardan sınıflandırabiliriz. En sık kullanılan sınıflandırmalar;

1) seviyelerine göre sınıflandırma

2) uygulama alanlarına göre sınıflandırmadır.

❖ **Bilgisayar Dillerinin Seviyelerine Göre Sınıflandırılması**

1. kuşak diller: Makine dili bilgisayarın doğal dilidir ve bilgisayarın donanımsal tasarımına bağlıdır. Bilgisayarların geliştirilmesiyle birlikte onlara iş yaptırmak için kullanılan ilk diller de makine dilleri olmuştur. Bu yüzden makine dillerine 1. Kuşak diller denilebilinir.

2. kuşak diller: 1950'li yılların hemen başlarında makine dili kullanımın getirdiği problemleri ortadan kaldırmaya yönelik çalışmalar yoğunlaşmıştır. Sembolik makine dilleri (Assembly languages) yalnızca 1 ve 0 dan oluşan makine dilleri yerine İngilizce bazı kısaltma sözcüklerden oluşuyordu. "Compiler" derleyici buluşu bu dönemde yapılmıştır. Assembly diller 2. kuşak diller olarak tarihte yerini almıştır.

3. kuşak diller: Tarihsel süreç içinde Assembly dillerinden daha sonra geliştirilmiş ve daha yüksek seviyeli diller 3. kuşak diller sayılmaktadır. Bu dillerin hepsi algoritmik dillerdir. Fortran, Pascal, Basic, Cobol,…

4. kuşak diller: Çok yüksek seviyeli ve genellikle algoritmik yapı içermeyen programların görsel bir ortamda yazıldığı diller ise 4. kuşak diller olarak isimlendirilirler. Access, Foxpro, Paradox, Xbase, Visual Basic, Oracle Forms,…

Ortalama bir satırındaki kodlama 3. kuşak dildeki 10 satır kodlamaya eşittir.

3. kuşak diller prosedürlüdür. Programcı her işin bilgisayar tarafından "nasıl" yerine getirileceğini detaylı bir şekilde tanımlamak zorundadır. 4.kuşak dillerde buna gerek yoktur. Sadece "ne" yapılması gerektiği tanımlanır. Farklı bir deyişle; 4. kuşak diller öncekilere göre daha problem amaçlıdır. 4. kuşak dille geliştirme daha hızlıdır, daha az programcı gerektirdiğinden daha az maliyetlidir. Dilin öğrenilmesi ve o dilde program yazılması daha kolaydır [Britton et al, 1993].

❖ **Uygulama Alanlarına Göre Sınıflandırma**

a) Bilimsel ve mühendislik uygulama dilleri: Pascal, C, FORTRAN

b) Veri tabanı dilleri : XBASE, Oracle, Clipper, Visual Foxpro....

c) Genel amaçlı programlama dilleri: Pascal, C, Basic.

d) Yapay zekâ dilleri: Prolog, Lisp

e) Sistem programlama dilleri: C, Assembler, sembolik makina dilleri.

3.3.2.2 Programlama Dili Seçimi

Hangi dilin seçileceği belirlenirken tüm kriterlerin göz önünde bulundurulmasıgerekir. En önemli kriterler:

1. Uygunluk: Bazı diller genel amaçlıdır ve çeşitli programlarda kullanılabilir, bazıları ise özel amaçlıdır ve bazı sınırlı işlerde kullanılabilirler. Dil seçimi kullanıcının ve kullanımın ayırt edilmesini gerektirir.

2. Karmaşıklık: Yüksek seviyeli diller karmaşık kontrol yapılarını ve veri yapılarını içermelidir. Kontrol yapıları, okuması ve oluşturması kolay, açık mantıksal yapıdaki programların dış yapısını oluştururlar. Diller birçok veri yapısını destekleyecek şekilde seçilmelidir.

3. Organizasyonel Düşünce: Bir dilin etkili olması için kullanıcı için çabuk öğrenilebilir olması gerekmektedir. Oluşturulması ve değerlendirilmesi kolay ve organizasyonla birlikte büyüyebilecek kadar esnek olmalıdır.

4. Destek: Bir yazılım satın alınırken başka organizasyonlar tarafından geniş bir kullanıma sahip, destek veren firmalar ve servisler tarafından kontrol edilmiş olması önemlidir.

5. Etkinlik: Bir yazılım satın alınırken o dilin derlendiğinde ve çalıştığında kalan performansının etkinliği önemlidir. Birim hafıza başına düşen makine fiyatı düşerken, makine zamanında etkin olmayan fakat programlama zamanında etkin olan dillerin önemi giderek artmaktadır [Atan, 2004].

3.3.3Kullanılan Ölçümler

Hata sayısı tahmin etmek için basit ölçüm kod satırı sayısıdır. Hata sayısı önemlidir, çünkü bulunan hata sayısı önceden belirlenen maksimumu geçerse modül tekrar tasarım ve kod aşamalarına girmelidir.

Hata tipleri de önemlidir. Tipik hata tipleri; tasarımı anlayamamayı, birbirini tutmayan değişkenlerin kullanımını içerir. Tutulacak hata dataları, daha sonraki gelecek ürünlerde hata kontrolü sırasında kullanılan kontrol listesine eklenebilir [Schach, 1993].

3.4 Test

'Yazılım testi', bir sistem veya uygulamanın denetlenebilir koşullar altında çalıştırılması (veya işletilmesi) ve elde edilen sonuçların değerlendirilmesidir. Olması gereken şeylerin olmadığını veya tam tersi olmaması gereken şeylerin olduğunu denemek ve ortaya çıkartmak testin amacı olmalıdır.

Yazılım testi; kaliteli ürünlerle müşteri memnuniyetini artırmak, geliştirme işleminin erken aşamalarında yanlışları saptayarak ileri aşamalara yayılmasını önlemek, böylece zaman ve maliyetten tasarruf sağlamak gibi genel amaçlar için yapılır [Cebeci, 2001].

Test etmek (testing) ve hata ayıklamak (debugging) farklı aktivitelerdir. Fakat hata ayıklamak herhangi bir test stratejisinin içine yerleştirilebilmektedir (Natarajan, 2004). Kodlamada oluşan hataların tespit edilip, bunların ayıklanması gerekmektedir. Bu işleme hata ayıklama "debugging" denir. Test ayıklayıcı, çeşitli hipotezler bularak hata ayıklaması yapar (Sommervile, 2000). Test etmek ise daha geniş bir kavramdır. Doğrulama (verification) & onaylama (validation) ile ilgilidir [Natarajan, 2004].

Doğrulama (verification) ise yazılımın iç doğruluğu ile uğraşır ve şu soruyu sorar: "Geliştirilen sistem doğru yolda mı?".

Onaylama (validation); sistemin müşterinin beklentilerini karşılayıp karşılamadığına bakmaktadır. Yazılımı müşterinin ihtiyaçları ile karşılaştırmaktadır ve şu soruları sormaktadır: "Geliştirilen doğru ürün mü?", "Amaçlara uyacak mı?" [Sommervile, 2000].

3.4.1 Test Aşamaları

Test, süreklilik arz eden bir süreçtir. Yazılım üretiminde ilk testler geliştirme sürecinde programcı tarafından yapılır. Bununla birlikte, asıl hata ayıklama ve geribildirim hizmeti test ekipleri tarafından yapılır. Testler ve geribildirim müşteri yazılımı kullandığı sürece devam eder. Programcıların yaptığı testler ağırlıklı olarak iş akışı değil, teknik testlerdir. İş akışı yönünden

yazılım testi, özel bir ekip tarafından yapılır. Test takımında; profesyonel testçiler (testleri organize ederler ve uygularlar), analistler (müşterinin ihtiyaçlarını bildiklerinden ve tasarımcıyla birlikte çalıştıklarından sistemin çözümü sunmak için nasıl çalışması gerektiğini bilirler), grup yönetimi uzmanları (hatayı düzeltmek için yapılan değişiklikler diğer test planlarını etkileyeceğinden testin revizesinin yapılmasını sağlarlar) ve kullanıcılar yer alabilmektedir [Pfleger, 1991].

Kodlama aşamasında, yazılım sistemlerinin çok sayıda karmaşık formül, aktivite, algoritma içermesinden ve bazen müşterilerin isteklerindeki belirsizliklerden dolayı hatalar olabilmektedir [Pfleger, 1991]. Kodlama aşamasındaki testi programcının yapmamasının çeşitli nedenleri vardır: test sonuçları, programcıdan kurduğu bir şeyi yıkmasını isteyebilir, ayrıca programcı tasarım ve tanımlamaların yönlerini anlamayabilir. Başkasının testi yapması bu tür hataların bulunmasını sağlar. Yine de hatanın kaynağını bulup, düzeltme işlemini en iyi yapan kodları yazan programcıdır [Schach, 1993].

Test aşamalarını şu şekilde belirleyebiliriz:

1. Program testi

a. Modül ve birim testi: Her bir modül, tek bir program gibi test edilir. Her yazılım birimi programının her işlem dizisinin en az bir kez çalıştırılması ve sonuçların beklentilerle karşılaştırılması anlamına gelir.

b. Entegre testi: Birim testinden sonra, birimlerin entegresi sırasında ortaya çıkan yazılımın tasarlanan sistemle karşılaştırması yapılır. Diğer bir deyişle, modüllere ait ara yüzün doğru tanımlanıp tanımlanmadığı test edilir.

2. Sistem testi

a. Fonksiyon testi: Entegre sistemde istenilen fonksiyonların testi yapılır. Bu safhada "doğru yazılımın üretilip üretilmediği" incelenir. Bu test "kara kutu" testi olarak da adlandırılır. Yazılımın gereksinme duyulan şeylere yanıt verip veremediği ve işlevselliği sınanır.

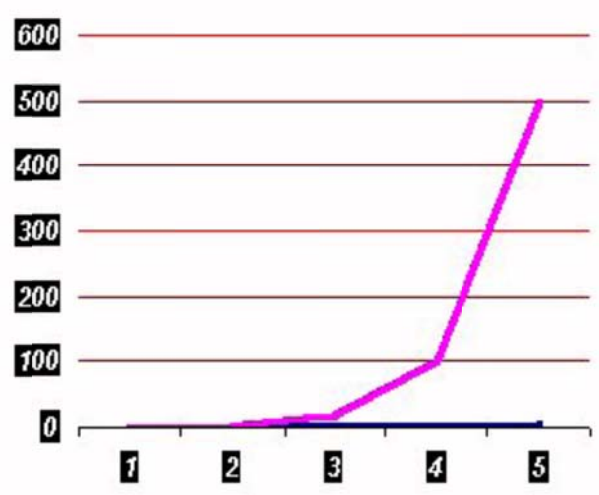
b. Performans testi: Sistem ile diğer arta kalan ihtiyaçlar karşılaştırılır. Testlerden geçerse geçerliliği onaylanmış sistem olur [Natarajan, 2004].

c. Kabul testi: Teste müşteri dahil edilir, ihtiyaçlar kontrol edilir. Bu aşamada alfa testinden söz edebiliriz. Alfa testi, bitirilme aşamasına yakınlaşmış olan bir uygulama için yapılan testtir. Programcılar veya test uzmanlarınca değil, son kullanıcılar tarafından yapılır. Müşterinin datasında test edilir. Kabul testinde, müşteri sistemin fonksiyonları gereksinimleri karşılıyor mu, istenilen raporlar, cevaplar öngörülen zamanda alınabiliyor mu, sistem öğrenme ve kullanım için kolay mı, destekleyici dokümantasyonların kalitesi nedir sorularını sorabilir. Bu, müşterinin sistemi teslim almaya karar verdiği noktadır [Sommervile, 2000].

d. Yükleme testi: Test sistemin kullanılacağı yerde yapılır [Pfleger, 1991]. Bu aşamada da beta testinden söz edebiliriz. Beta testi, sistemin pazara sunulmasından sonra gelen teste denir. Bunlar geliştiricilerin önceden sezemediği hatalar ve/veya sorunları saptamak üzere yapılan testlerdir. Programcılar veya test uzmanlarınca değil son kullanıcılar tarafından yapılmaktadır [Sommervile, 2000]. Yazılımın beta testlerinde mutlaka müşteriden test grupları oluşturması istenmelidir. Bu sayede müşterinin yeni yazılıma adaptasyonu da sağlanmış olur. Testlerin sonunda ilgili birimlerle birlikte değerlendirme toplantıları yapılmalıdır. Bazen hataların kaynağı analizde ya da tasarımda olabilir. Kullanıcıdan gelen yeni bir istek var ise, bu talep doğrudan analiz ekibine iletilmelidir. Çünkü yazılımın mimarisinin temelleri analizciler tarafından hazırlanmıştır [Sezgin, 2000].

Boehm'e göre yazılım sürecinin test aşamasında yapılan hata düzeltmenin masrafı, başlangıçta yapılan hata düzeltme masrafından yaklaşık 70 kat daha fazladır [Boehm, 1989].

1: Analiz
2: Tasarım
3: Kodlama
4: Test
5: Teslim Sonrası
X: Proje Safhası
Y: Hatayı Düzeltmenin Masrafı



Şekil 4. Bir Hatayı Düzeltmenin Proje Safhasına Bağlı Olarak Değişen Masrafı [Gül, 2006]

Şekilden de görüleceği üzere erken safhalarda yapılan hata düzeltme işlemleri maliyet açısından fayda sağlamaktadır. Test safhasına ayrılan sürenin çok fazla olması yazılım geliştirmede çok önemli bir maliyet olup projenin kârlılığını düşüren bir etmen durumundadır.

3.4.2. Test Araçları ve Teknikleri

Test araçları, kodlama başlamadan planlanır ve bu araçlar tasarım ve kodlama yapılırken geliştirilir [Natarajan, 2004]. Test sürecinde önemli olan ne tür test araçlarının kullanılacağıdır.

Programın ya da modüller topluluğunun doğruluğunu araştıran araçlar 4 gruba ayrılır:

1. Kod analizörü: Kodlamada dizim, form (syntax) hatalarını belirler.
2. Yapı kontrolü: Yapının akışını kontrol eder.
3. Data analizörü: Data deklarasyonunu, modül ara yüzlerini analiz eder.
4. Mantıklılık kontrolü: Kodlama yanlış mantık sırasında ise işaretlenir [Pfleger, 1991].

3.4.3 Test Dokümantasyonu

Testin kompleks ve zorluklarını kontrol etmek için dikkatlice hazırlanmış test dokümantasyonu kullanılır [Pfleger, 1991].

3.4.4 Test Yapmanın Zorlukları

Test safhasında, geliştirilen her yazılım için kullanılabilecek ya da özelleştirilerek kullanılabilecek araçlar mevcut değildir. Otomatik araçları temin etmek çok pahalıdır. Test uzmanlarını yeni yöntemler konusunda eğitmek ve eğitilmiş insan kaynaklarını elde tutmak zordur. Bu gibi nedenlerden dolayı yazılım testi hem zor hem pahalıdır [Cebeci, 2001].

Burada şundan da bahsetmekte fayda vardır. Spesifikasyonların formal şekilde ifade edilmesi, programın spesifikasyonlarını karşılayıp karşılamadığını daha doğru test etmemizi sağlar. Formal olmayan, yapısal tekniklerle (data akış, varlık-bağıntı diyagramları,… gibi) yazılan spesifikasyonlara karşı programı test etmek çok daha zordur [Sommervile, 2000].

Yazılım test pratiklerinin temel problemleri arasında ayrıca, testte kestirme yolun izlenmesi, test süresinde azaltma yapılması, şimdi teslim edelim, sonra düzeltiriz - davranışı, zayıf planlama ve düzeltme, kullanıcının sürece katılmasındaki eksiklik, zayıf dokümantasyon, yönetici desteğinin eksikliği, uygulama çevresinin bilgi yetersizliği, uygun nitelikte olmayan çalışan problemlerini de ekleyebiliriz.

Yazılım sistemlerinde zayıf testler sistem güvenilirliğini azaltmakta ve dolayısı ile yazılım kalitesini negatif etkilemektedir. Testin etkinliğini artırmak ve yazılım kalitesini iyileştirmek için yazılım evleri daha yüksek seviyeli yazılım kültürüne geçiş yapmalıdırlar.

Yazılım test teknikleri, metodolojiler, araçlar ve standartlar teste sadece yardım ederler, fakat etkili testi planlayan ve yerine getiren yönetim ve insanlardır. Test etmenin sadece hata ayıklamak ve onları temizlemekten ziyade "müşteri memnuniyeti"ni maksimize etmeye odaklanma ihtiyacı vardır. Yazılım kalite yönetimini etkileyen bu faktörlerdeki iyileşmeler yazılım kalitesini de iyileştirecektir [Gill, 2005].

3.5 Sistem Teslimi

Problemin belirlenmesi, çözümün tasarımı, gerçekleştirilmesi ve test edilmesi aşamalarından sonra sıra sistemin müşteriye sunulmasına gelmiştir.

Bu aşamada; kullanıcıya ürünü anlaması ve ürünün anlaşılabilmesi için yardım edilmelidir. Teslim işlemi başarısız olursa kullanıcı sistemi doğru kullanamayacak belki de sistem performansı ile mutsuz olacaktır. Kullanıcılar olması gerektiği gibi verimli olamazlarsa, geliştirici takım tarafından verilen dikkat boşa gitmiş olacaktır.

Sistemin kullanıcıya başarılı transferi için 2 önemli nokta: eğitim ve dokümantasyondur [Pfleger, 1991].

Satış ve destek aşamalarında kullanılan en önemli malzeme, dokümanlardır. Altyapı gereksinimleri, kurulum, ayarlar ve bunlara ilişkin çeşitli eğitsel dokümanlar, uzman kişilerce hazırlanmalıdır.

Satış öncesi yapılacak hazırlıklar;

- Kurulum ve ayarlama dokümanları,
- Uygulama eğitim programlarının oluşturulması,
- Uygulama kullanım kılavuzu hazırlanması,
- Sıkça sorulan soruların cevaplanması,
- Destek birimi eğitim programı oluşturulması [Sezgin, 2000].

3.6 Bakım

Bakım, yazılım kullanılmaya başlandıktan sonra üzerinde yapılan değişiklik prosesidir. Yazılım, müşterinin isteklerine göre sürekli değişim içindedir [Sommervile, 2000].

Yazılımdaki bakım, donanım bakımı gibi bir şeyin tamir edilmesi veya düzgün çalışması için önlemlerin alınması demek değildir. Yazılım sistemleri

birleşik değişiklikler üzerine kuruludur. Sistem geliştirmeye yöneliktir [Pfleger, 1991].

Anahtar nokta; bakımın sürecin herhangi bir yerinden değil en başından ürüne yerleştirilmesidir.

Teslimden sonra değişiklik yapmanın temelde 3 nedeni vardır:

• Varolan herhangi bir hatayı düzeltmek için (gereksinim, tasarım, kodlama, dokümantasyon hatası gibi) yapılan bakımlar. Buna doğrulayıcı bakım (corrective maintenance) denir.

• Kusursuzlaştırma bakımı (perfective maintenance). Değişiklikler ürünün etkinliğini iyileştirecektir.

• Ürünün çevresinde yapılan değişikliklere cevap verebilmesi için yapılan değişiklikler [Schach, 1993].

Lientz, Swanson ve Tompkins,1978'de, 69 organizasyonda yaptıkları bir çalışma, bakım programcılarının zamanlarının %17,5'ini doğrulayıcı bakıma ayırdıklarını göstermiştir. Zamanlarının %60,5'ini 2.tip bakıma kusursuzlaştırma bakımına, %18 'ini ise 3. tip bakıma ve %4'ünü bunların dışında bakıma ayırdıkları gözlenmiştir [Schach, 1993].

3.6.1 Bakım Testi

Geri dönüş (Regression) testi: Herhangi bir değişiklikte, değişikliğin programın diğer parçalarında bir probleme yol açıp açmadığının test edilmesi gereklidir. Görünürde değişiklik yapılan modülle ilişkisi olmayan tüm ürünün tekrar test edilmesinin gerekmesi zamanı boşa harcamak olduğu tartışılmaktadır. Bakım sırasında yapılan değişikliklerin, farkında olmayan, kasıtsız tarafının sonuçlarının tehlikesi büyüktür. Bu nedenle test, bakımın gerektirdiklerindendir.

3.6.2 Kullanılan Araçlar

İşletim sistemi versiyon kontrolünü kapsamıyor ise versiyon kontrol araçlarına ihtiyaç duyulur. Araçlar, bakımın etkin, verimli olmasına yardım eder.

3.6.3 Kullanılan Ölçümler

Bakım safhasındaki aktiviteler tanımlama, tasarlama, kodlama, entegre etme, test ve dokümantasyon oluşturmak gibi aktivitelerin hepsini içerir. Bu aktiviteler için alınan ölçümler bakım safhası için de uygulanabilir [Schach, 1993] [Gül, 2006].

BÖLÜM 4: TAKIM KAVRAMI VE YAZILIM GELİŞTİRME TAKIMLARI

4.1 Takım Kavramı

Takım, önceden belirlemiş bir görevi yerine getirmek için bir araya gelmiş, birbirine bağımlı ve birlikte hareket eden, görevin sonuçları ile ilgili sorumluluğu paylaşan, üyelerinin kendileri veya başkaları tarafından kendisinden daha büyük bir veya birden fazla sosyal sistem (örn: işletme veya organizasyon) içerisinde bir varlık olarak görüldüğü ve üyeleri arasındaki ilişkileri organizasyon sınırları içerisinde yöneten bireyler topluluğudur [Cohen at al, 1997, s.241, 242, 243]. Yukarıdaki tanımdan da anlaşıldığı gibi, bir topluluğun takım olabilmesi için her şeyden önce "ortak bir hedefi paylaşması" ve "bir amaca doğru ilerlemek isteğinin" bulunması gerekir, aksi halde bu topluluğun herhangi bir gruptan bir farkı olmaz. Schermerhorn ve arkadaşları yaptıkları çalışmada takımı şöyle açıklarlar, "takım, birbirlerini tamamlayıcı becerilere sahip ve kendilerini sorumlu hissettikleri ortak bir amaca ulaşabilmek için çalışan küçük insan topluluğudur" [Schermerhorn at al, 1997, s.275].

Yapılan tanımlamaları kullanarak takım; kabul edilmiş ortak hedefler doğrultusunda üyelerden her birinin öteki üyelerin sosyal ve fonksiyonel rollerini bilerek, karşılıklı dayanışma, işbirliği ve etkileşim içinde olduğu, biçimsel bir üyelik gurubudur diyebiliriz. Takımlar amaçlarına ulaşmak için beraberce nasıl çalışacaklarına ilişkin ortak bir yaklaşım geliştirmelidirler. Gerçekte, amaçlarını şekillendirmek için harcadıkları zaman ve çaba kadar işlerine nasıl yaklaşacakları konusunda da zaman ve çaba sarf etmelidirler.

Takım üyelerinin, belirli işleri kimlerin yapacağı, çalışma zamanlarının hazırlanması ve sadık kalınması sağlanmalıdır. Takımın kararları nasıl vereceği ve nasıl tadil edeceği ve işin yapılması için yaklaşımın nasıl tadil edileceği konularında anlaşmış olmaları gereklidir [Katzenbach at al, 1998, s.83,75, 197, 71].

Takımın üyeleri belli bir hedefi gerçekleştirmek için çalışırlar. (Örneğin, yenibir yazılım geliştirmek veya ortak bir kitap yazmak gibi.) Onları bu hedefe götürecek sorumlulukları paylaşabilmeleri ve bireylerin görev üstlenebilmeleri önemlidir. Tabi tüm bunları bir zorlamayla değil, kendi istek ve iradeleriyle yapmaları gerekir.

Deneyimlerin, kararların, yeteneklerin önemli olduğu durumlarda, takım kaçınılmaz olarak tek başına ya da daha geniş organizasyonel gruplarda hareket eden bireyler topluluklarından daha başarılı olur. Takımlar, büyük organizasyonel gruplardan daha esnektirler; çünkü sürekli yapıları işlemleri sekteye uğratmak yerine bunları destekleyecek şekilde hızlıca bir araya gelebilir, belirli bir plana göre yerleştirilebilir, belirlenmiş hedefler üzerinde tekrar toplanabilir ve dağılabilirler.

Takımlar becerilerin doğru karışımını sağlamalıdır; yani beceriler takımın görevlerini yerine getirebilmesi için birbirini tamamlamalıdır. Bu tür takım becerileri üç grupta ele alınabilir [Katzenbach at al, 1998, s.83,75, 197, 71].

Teknik veya fonksiyonel yetenek: Bir grup doktorun mahkemeye başvurup hastane yöneticilerinin kendilerine iş vermediğini iddia ederek dava açmalarının bir anlamı olmaz. Ama doktor ve avukat takımları çok defa yanlış tıbbi tedavi veya kişisel zararlar iddia edildiğinden kendilerini mahkemede bulurlar. Aynı şekilde, sadece pazarlamacılar veya mühendislerden oluşan ürün geliştirme grupları, her iki tamamlayıcı beceriye de sahip guruplardan daha az başarılı olacaklardır.

Sorun çözme ve karar verme becerileri: Takımlar sorun ve fırsatları tanımalı, ileri gitmek için neler yapılması gerektiğini anlamalı ve sonra da süreci ne gibi kararlarla yürütmek gerektiğini bilmelidir. Gerçi pek çokları bu işler için gerekli becerileri yüklendikleri işleri yürütürken geliştireceklerse de takımların bu tür becerilere sahip kimselere ihtiyaçları vardır.

Sosyal beceriler: Bu tür becerilere dayanan etkili iletişim ve yapıcı anlaşmazlıklar olmaksızın ortak anlayış ve amaç da olmaz. Bu sosyal beceriler

arasında riskleri göze almak, yararlı eleştiriler, objektif, aktif dinleme, şüpheye yer bırakmamak, diğerlerinin çıkar ve başarılarını onaylayarak desteklemek vardır.

Literatürde takım ve grup kavramları ile ilgili birbirine çok yakın tanımlar bulunmaktadır. Bazı araştırmacılar her iki kavram için de aynı tanımı ullanmışlar, fakat bir kısımı ise bu iki kavramı birbirinden ayırmışlardır.

Cohen ve Bailey yaptıkları araştırmada, "takım" ve "gurup" kavramlarının birbirlerinin yerine kullanılabileceğini fakat "takım" teriminin daha sık kullanıldığını belirtmiştirler ve popüler yönetimle ilgili literatürde "takım" kavramının (örn: güçlendirilmiş takımlar, kalite geliştirme takımların, takım etkinliği) tercih edildiğini buna karşın akademik literatürde ise genel olarak "grup" kavramının (örn: gurup birliği, gurup dinamikleri) tercih edildiğini belirtmişlerdir [Cohen at al, 1997, s.241, 242, 243].

Katzenbach ve Smith takım kavramını, "bir takım, kendini ortak bir amaca, performans hedeflerine ve kendilerini beraberce sorumlu tuttukları yaklaşıma adamış, birbirini tamamlayıcı becerilere sahip az sayıda insan topluluğu" olarak tanımlamıştır [Katzenbach at al, 1998, s.83,75, 197, 71].

Takım ve grup terimlerinin farklı kavramlar olduğunu savunan araştırmacılara göre, ikiden fazla kişinin biraraya gelmesi ve bu kişilerin birbirini etkilemesi sonucunda grup oluşturulabilir, grupta bireysellik ön plandadır, grubun başarısı bireysel başarıların toplamıdır. Gruplarda ortak bir amaç olması zorunlu değildir, farklı amaçları olan kişiler bilgi paylaşımı yolu ile amaçlarına ulaşabilmektedirler. Takımlarda tüm takım üyelerinin benimsediği ve bu amaca ulaşmayı arzuladığı ortak bir amaç vardır.

Takım üyeleri sahip oldukları ortak amaç doğrultusunda takıma karşı yüksek bağlılık gösterirler. Takım ve grup üyelerinin bağlılıklarında görülen farklardan dolayı takımlar gruplara göre daha fazla üretkendir. Takımların gruplardan daha başarılı olmalarının bir diğer nedeni, takımların üyelerin çabalarını koordine etmesi ile üyeler bilgi, beceri ve yeteneklerini birleştirerek

pozitif bir sinerji yaratırlar. Oluşan bu sinerji ile takımlarda bireysel başarıların toplamından daha fazla başarı elde edilir.

4.2 Örgütlerde Takımların Oluşturulmasına İlişkin Nedenler

İnsanlar kendilerinin ya da üst kademedeki yöneticilerin istediklerini yaptırmak amacıyla bir takım kurabilirler. Takımın kurulmasındaki nedenlerden birincisi bu ise, değişim konusunda büyük bir direnişle karşılaşılacağı aşikârdır.

Tuskan yaptığı araştırmada takımın meydana gelme amacının hem yönetimin hemde, yönetim dışındakilerin isteklerini karşılayacak iki amaca hizmet etmesi gerektiğini söyler [Weiss Donald, 1998, s.8, 22].

Öncelikleri belirlemek için beraber çalışarak işin sonuçlarını iyileştirmek, problemleri çözmek, karar almak ve grup içindeki çalışma ilişkilerini düzeltmek.

Grup üyeleri arasındaki kişisel ilişkileri düzelterek, grup üyelerini gereksiz yakın gözetimden kurtararak, yaşamlarını zenginleştirerek, yaptıkları işin efendisi olmalarına izin vererek, işleri başarıyla yapmaktan elde edilen yararlara katılmalarını sağlayarak çalışma yaşamının niteliğini iyileştirmek.

Örgütler içerisinde bulundukları belirsiz ve değişken çevreye uyumlu olabilmek için yeni uygulamaların arayışı içerisindedirler. Dünya pazarlarındaki küreselleşme hareketleri, politik çıkarlar, bilgi sistemleri, üretim süreçlerindeki ve bunların yönetimlerinde yaşanan teknik ilerlemeler, şirket birleşmeleri ve örgütlerin küçültülmesi ve yaygınlaştırılması gibi nedenler örgütsel yönetim stratejilerinin ve yapılarının farklılaştırılmasına neden olmaktadır. Bu yaşanan değişim süreci örgütlerin ve projelerin yönetiminde yeniden yapılanma hareketlerini ve takım çalışmalarını zorunlu hale getirmiştir. Bu anlamda takım faaliyetleri ürün kalitesi, verimlilik ve performansı arttırarak stratejik amaçları gerçekleştirecek ve rekabet avantajı sağlayacak bir örgüt dizaynı parametresi olarak önem kazanmaktadır. Takımlarla çalışmak demek özellikle hem örgütün

yeniden yapılanmasını hem de değişim inisiyatifinin desteklenmesini kapsayan bir stratejik riski de kabul etmek demektir [İnce ve ark., 2004, s.426].

Bu manada örgütlerde takım anlayışına geçişin nedenlerini aşağıdaki başlıklar etrafında sıralamak mümkündür [İnce ve ark., 2004, s.426].

• Dış çevrede yaşanan hızlı gelişmelerin ışığında ortaya çıkan yeni bilgi alanlarının takım bilgi ve becerisini gerektirmesi.

• Takım sinerjisinin takımı tek tek bireylerden daha güçlü yapması.

• Örgütsel verimliliğin arttırılması.

• Üretim ve kalite artışının sağlanması.

• İş mükemmelliği anlayışına ulaşılması.

• Çalışanların motivasyonlarının arttırılması.

• Çalışanlara birlikte ve özerk çalışma anlayışının kazandırılması.

• Örgüt gerçeğine uygun etkili fikirlerin üretilmesi.

• Bireylerin bağlılık duygularının gelişimine katkıda bulunması.

• İş tatmininin ve örgütsel bağlılık duygularının gelişmesinin sağlanması.

• Esnek ve yalın örgüt yapısının oluşturulması.

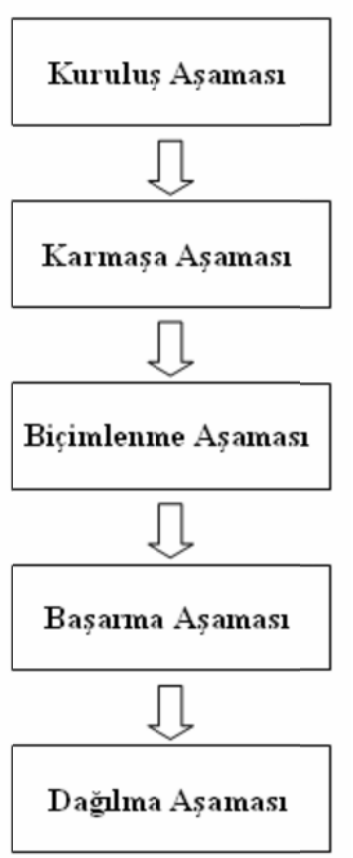
• Çalışanların liderlik ve yaratıcılık yeteneklerinin ortaya çıkması konusunda teşvik edilmeleri.

• Kararların kalitesinin arttırılması ve problem çözümlemenin kolaylaştırılması.

4.3 Takımların Kuruluş Aşamaları

Takımlar bir anda oluşturulamazlar, belli bir zaman süreci içerisinde belli aşamaların meydana gelmesiyle oluşturulabilirler. İyi koordine edilmiş bir grup oluşturmak biraz zaman alır ve grup üyeleri arasında büyük miktarda etkileşim gerektirir. Üyelerin birbirleriyle tanışmaları, birbirlerini anlayıp uyum içinde ortak hareket etmeleri gerekir. Grup üyeleri birbirleri ile etkileşime girecek vakit bulamıyorsa, grubu iyi organize edilmiş bir takım haline getirmek mümkün olmayabilir. Takımlar oluşturulduktan sonra birkaç evreden gerek olgunlaşır. Takım üyeleri başlangıçta takım içinde uyulması gereken kuralları belirlemek, üyelerin rollerini tespit etmek ve işbölümü yapmak ihtiyacı duyarlar. Bu sayede üyeler düzenli bir şekilde işleyen bir bütünün bir parçası haline gelirler. Liderlerin

görevi, takımın hangi gelişim safhasında olduğunu tespit etmek ve gerekli tedbirleri almaktır. Eren'e göre takımların kurulmasında kuruluş, karışıklık, biçimleme, başarma ve dağılma olmak olmak üzere beş aşama vardır [Eren, 2001, s.463].



Şekil 5. Takımların Kuruluş Aşamaları [Eren, 2001]

a) Kuruluş Aşaması: Bu evrede üyeler bir araya gelip birbirlerinin karakterlerini ve çevrelerini tanımaya ve kendilerini diğerlerine tanıtmaya çalışırlar. Bu safhada belirsizlik yüksektir. Üyeler kendilerinden ne beklendiği, buradaki diğer üyelerler beraber verimli bir çalışma yapılıp yapılamayacağı gibi sorulara cevap ararlar. Bu aşamanın başarılı bir şekilde yürütülmesi için genellikle bir lidere ihtiyaç duyulur [Eren, 2001, s.463].

b) Karmaşa Aşaması: Bu aşamada üyeler takım içindeki rollerini ve kendilerinden ne beklendiğini daha iyi anlamışlardır. Bu evredeki en büyük özelliği çatışma ve fikir ayrılıklarının meydana gelmesidir. Ortak çıktıları bulunan alt gruplar arasında takımın genel kuralları ve bu kurallara ulaşmak için

uygulanacak yöntemler hakkında çatışmalar çıkabilir. Takımlar bu evreyi geçemezlerse istenilen koordinasyon sağlanamaz ve yüksek bir başarı seviyesi yakalanamaz veya takım dağılabilir. Bu evrede grup liderlerine önemli bir görev düşer [Eren, 2001, s.463].

Bu dönem kişilerin takımın yapısını anlamaya başladığı ve projedeki liderlik yapısını irdeledikleri bir dönemdir. Bu aşamada üyelerin verdikleri tepkiler genellikle, kendilerine verilen zamanın az olması ve o zaman kadar yapılmış olan planlamanın yetersiz olmasından kaynaklanır. Bu aşamada liderler elemanları, takımın görevleri ve hedefleri hakkındaki belirsizlikleri ve yanlış algılamaları çözemeye teşvik etmelidir [Hingst, 2006, s.4].

c) Biçimlenme Aşaması: Bu aşamada üyelerin arasındaki anlaşmazlıklar ortadan kalkar, takımda birlik ve uyum oluşturulur ve çatışmalar çözüme ulaştırılır. Elemanların arasındaki takım ruhu gelişir. Üyeler ortak bir amaç doğrultusunda kendilerine düşen görevleri yerine getirmeye başlarlar. Üyeler birbirlerini tanırlar ve kabullenirler. Liderin kim olduğu ve takım elemanlarının görevlerinin neler olduğu belirlenir [Eren, 2001, s.463].

d) Başarma Aşaması: Bu aşamada sorunların çözülmesi ve üyelerden beklenen görevlerin yerine getirilmesi konularının üzerinde durulur. Hedeflerin gerçekleşmesini sağlayacak tüm sorunlar teker teker ele alınmalı ve sonuca ulaştırılmalıdır. Sorunlara çözüm getirebilmek için üyeler birbirleriyle etkileşim ve işbirliği içerisinde olmalı, çıkabilecek anlaşmazlıklarda soğukkanlı davranarak çözüm yaratmaya çalışmalıdırlar. Bu evrede liderler takımdan yüksek bir verim elde etmek için çaba harcamalıdırlar [Eren, 2001, s.463].

e) Dağılma Aşaması: Takımların kuruluşunun son aşaması bu aşamadır. Önceden bellirlenmiş bir görevi yerine getirmek için biraraya gelmiş takımlarda dağılma aşaması söz konusudur. Projenin sonunda ulaşılan yüksek performans seviyeleri çok büyük tatmin verir [Mark D., 1991, s13]. Bu aşama boyunca çalışmaların hızı azalır ve gittikçe yavaşlayarak durur. Artık en önemli şey görevi yerine getirmek değil, başlamış görevlerin bitirilmesidir. Üyeler arasında güçlü bir bağ oluşmuştur. Üyeler sonuca ulaştıkları için mutlu olurlar fakat ilişkileri sona

ereceği için bir kaygı duyarlar. Bu evreden sonra takım lideri takımın görevini tamamladığını bir tören ya da toplantı yoluyla bildirir ve dilerse görevin başarıyla tamamlandıgını belirten başarı ödülleri, teşekkür mektupları veya plâketler dağıtır [Eren, 2001, s.463].

4.4 Örgütsel Takımların Ortak Özellikleri

Örgütsel takımların başarıya ulaşması ve etkin bir yapıya sahip olabilmesi için belirli özelliklere sahip olması gerekir. Yapılan çalışmalara dayanarak bu özellikleri üç ana başlık altında toplayabiliriz.

a) Ortak Amaca Sahip Olmak: Takım sayesinde örgütlerde çalışan bireyler temel hedefler ve örgütsel vizyonun belirlenmesi ve bu sürecin gerçekleştirilmesinde takımın yapması gerekenler konusunda belirli bir bilinç düzeyine sahiptirler. Takım çalışması sürecinde çalışanlar yönetimin kendilerinden beklentilerine ve bu beklentilerin nasıl karşılanacağına ilişkin bilgi düzeyine sahiptirler. Örgütlerde takımlar sayesinde paylaşılan bu vizyon çalışanların örgütlerin hedeflerini gerçekleştirmede yoğun bir gayret sarf etmelerini sağlayarak, örgütte güçlü bir kurumsal kültürün yaratılmasına katkı sağlamaktadır [İnce ve ark., 2004, s.426].

Takımlar ortak bir amaç uğrunda çalışırken yön ve bağlılıklarını geliştirirler. Bununla birlikte takımın amacına sahip çıkmak ve bir kimsenin kendisini o amaca adaması, ilk yönün takım dışından alınmasıyla çelişkili değildir. Yönetimden uzak kalmadığı sürece bir takımın kendi amacına "sahip çıkmacağı" varsayımı potansiyel takımlara yardımcı olmaktan çok karışıklığa yol açar. Takımların çoğu genellikle yönetim tarafından önlerine konulan talep ve fırsatlara karşılık kendi amaçlarını şekillendirirler [Katzenbach at al, 1998, s.83,75, 197, 71].

Takım faaliyetlerini takım amaçlarından ayırmak zordur ve bireylerin çoğu takıma çekilirler çünkü takımın gerçekleştirdiği faaliyetlerin amaçlarını ve hedeflerini benimserler. Başka bir deyişle, takım üyeleri için harekete geçirici olan amaçlar olmaktadır.

b) Liderlik: Takım liderlerinin bir potansiyel takımın gerçek bir takım hatta potansiyel bir takım haline gelmesinde önemli rol oynadığı çok açıktır. Kendisini ispat etmiş ve potansiyel kapasitesi olan birisi, bir takımın başına getirildiğinde takım performansı daha başarılı olacaktır. Bilhassa, sebepleri ne olursa olsun takım yaklaşımına tümüyle zıt tutumları olan kimselerin takımların başına getirilmemesi gerekir. Bu tür insanlar belirgin azınlıklardır, fakat onları takım liderliğine getirmek hatadır. Etkili bir lider olmak için takımın amaçlarına ve takım üyelerine inanmak gerekir. Özellikle iş dünyasındaki küçük gruplara lider olarak getirilen kişiler, kendilerini yönetici gibi görür, iş taksimini yapar ve tüm bireysel sorumlulukları üstlenmeye gayret gösterirler. Bu davranışlar çalışma gruplarında etkili olabilir, fakat bir takım liderinin davranışları olması beklenemez. Başarılı takımlarda tüm üyeler farklı zamanlarda potansiyellerini ortaya koyarlar. Her takımın performans zorlaması, takımı oluşturanlar ve yaklaşım farklı olduğundan, liderin işinin de zamanla değişmesi gerekir. Bu tip takımlarda liderlik daha az önem taşır ve liderlerin görevleri kolayca ayırt edilemez [Katzenbach at al, 1998, s.83,75, 197, 71].

Etkin takım liderleri takım amaçlarını iyi tanımlarlar, liderler takım üyelerinin moral ve verimliliklerini en kötü durumlarda bile yüksek seviyede tutarak takımın da performansını arttırırlar. Üyelerin kendi potansiyellerinin farkına varmalarını sağlayarak kendilerine olan güvenlerini arttırırlar.

c) Küçüklük: Katzenbach ve Smith yaptıkları araştırmada takımlarda küçüklük olgusunun pragmatik bir rehber olduğunu söyler, çok fazla sayıda insandan oluşan grupların teorik olarak bir takım olabileceğini, fakat büyük olasılıkla bu boyutlardaki bir takımın, fonksiyonlarını tek parça olarak yerine getiremeyeceğini ve alt gruplara bölüneceğini ifade ederler [Katzenbach at al, 1998, s.83,75, 197, 71]. Bir takımın üye sayısının azlığı ya da çokluğu takım faaliyetlerini de etkiler. İdeal takımın kaç kişiden oluşması gerektiği önemli konulardan bir tanesidir. Takımın sayısı, düşüncelerin açıklanmasına yetecek kadar büyük, ancak kararlara katılıma mani olmayacak ve serbest iletişime engel olmayacak kadar küçük olmalıdır. Bu sınır 5-7 kişi arasında görünmektedir [Karaca, 1994, s.40].

Takımların üye sayıları arttıkça, üyeler arasındaki iletişim azalacaktır. Üyeler tartışmalara katılamayacak, grup üyeleri grubun nasıl hareket edeceği hakkında anlaşmak bir yana grup olarak yapıcı bir biçimde anlaşmakta zorlanacaklardır. Beş kişilik bir grubun ortak bir amaç etrafında çalışıp sonuçlara ulaşması kırk kişilik bir gruba göre çok daha kolaydır. Eğer çok sayıda grup üyesi bir araya gelip bir sorunu görüşecekse toplantıyı küçük alt kümlerden giderek büyüyen kümelere doğru örgütlemek daha doğru olacaktır.

4.5 Takım Türleri

Takımlar literatürde değişik şekillerde sınıflandırılabilirler. Bu sınıflandırmalar takımların farklı kriterlerine göre örneğin tiplerine, çıktılarına, çalışma sürelerine göre yapılırlar. Bu araştırmada takımlar genel farklılıkları açısından dört sınıfta değerlendirilmiştir: Çalışma Takımları, Paralel Takımlar, Proje Takımları ve Yönetim Takımları.

a) Çalışıma Takımları: Bu tip takımların üyeleri çoğunlukla tam zamanlı çalışırlar ve görevleri net bir şekilde tanımlanmıştır. Geleneksel olarak çalışma takımları, ne yapacağı, işi kimin yapacağı ve nasıl yapacağı hakkında kararları veren bir denetleyici tarafından yönetilirler. Son zamanlarda işletmeler verimliliği arttırmak, kaliteyi yükseltmek, maliyetleri düşürmek amacıyla kendini yöneten çalışma takımlarından da yararlanmaya başlamışlardır [Cohen at al, 1997, s.241, 242, 243].

b) Paralel Takımlar: Paralel takımlar, farklı çalışma birimlerinden ya da işlerden insanların, organizasyonun mevcut donanımı ile iyi bir şekilde gerçekleştiremeyeceği fonksiyonların başarılabilmesi için bir araya getirilmesiyle kurulurlar. Bu tip takımlar sınırlı otoriteye sahiptirler ve sadece organizasyonel hiyerarşide kendilerinden üst kişilere öneride bulunabilirler. Paralel takımlar, problem çözme ve iyileştirme faaliyetlerini gerçekleştirmek için kullanılırlar Örneğin kalite çemberleri, kalite iyileştirme takımları paralel takımlardır [Cohen at al, 1997, s.241, 242, 243].

c) Proje Takımları: Zaman sınırı olan takımlardır. Yeni bir ürün ya da hizmet gibi işletme tarafından pazara sürmek için gerekli yeniliği sağlarlar. Proje

takımlarının görevleri tekrarlanabilir değildir ve dikkate değer bilgi, beceri ve tecrübe gerektirir. Bu bağlamda, ortak performans hedeflerine doğru yönelmiş olan takım üyelerinin, projenin başarılması için ihtiyaç duyulan gereklilikleri yerine getirecek niteliklere sahip olması büyük önem arzetmektedir. Proje takımlarının işi hâlihazırda var olan bir konsept üzerinde geliştirme yapmak ya da radikal ve farklı bir fikir ortaya çıkarmak olabilir [Cohen at al, 1997, s.241, 242, 243].

d) Yönetim Takımları: Yönetim takımları organizasyonun genel performansından sorumludur. Geliştirme, Üretim ve Satış gibi her alt birimden sorumlu yöneticilerinden oluşur. Bu tip takımlara sahip işletmelerde yönetimsel kararlar tek bir tepe yönetici tarafından alınmaz, kararlar yönetim takımındaki üyelerin ortak mutabakatı ile alınırlar. Yönetim takımları sorumlulukları paylaşarak, farklı çabaları birleştirerek işletmelerin global iş çevrelerinde rekabet edebilmelerine yardımcı olurlar [Cohen at al, 1997, s.241, 242, 243].

4.6 Takımların Başarısını Etkileyen Faktörler

Takım liderinin veya takım üyelerinin birbirlerine duygusal olarak destek olması takımdaki başarının artması için gereklidir. Takımda her üyenin kendi düşüncesini özgürce ifade edebilmesi, sorunlarına yardım bulabilmesi gibi durumlar onların takıma daha fazla bağlanmalarını ve takımla özdeşleşmelerini sağlar.

Takımlarla grupların başarısını belirleyen faktörler görünüşte aynı olsa da işleyiş açısından farklılıklar vardır. Takım başarısını etkileyen faktörlerin başında takım üyelerinin özellikleri gelir. Bir takımın başarılı olması için üyelerin özellikleri takımın amacıyla uyumlu olmalıdır. Takımın başarısı takım üyelerinin birbirini tamamlayıcı özellikte olmasına bağlıdır. Bu yüzden başarılı bir takımda üyelerin her birinin üst derecede bilgili ve yetenekli kişiler olması gerekli değildir [Erdoğan, 1991, s.470, 476, 477**].** Yani takım içinde görevler paylaştırılırken üyeler bireysel yeteneklerine ve özelliklerine uygun görevleri üstlenirlerse takımın başarısı artar.

Takımın başarısı için takımlarda üyelerin yeteneklerinin doğru karışımı sağlanmalıdır; yani yetenekler takımın görevlerini yerine getirebilmesi için benzer özellikte değil birbirini tamamlayıcı nitelikte olmalıdır.

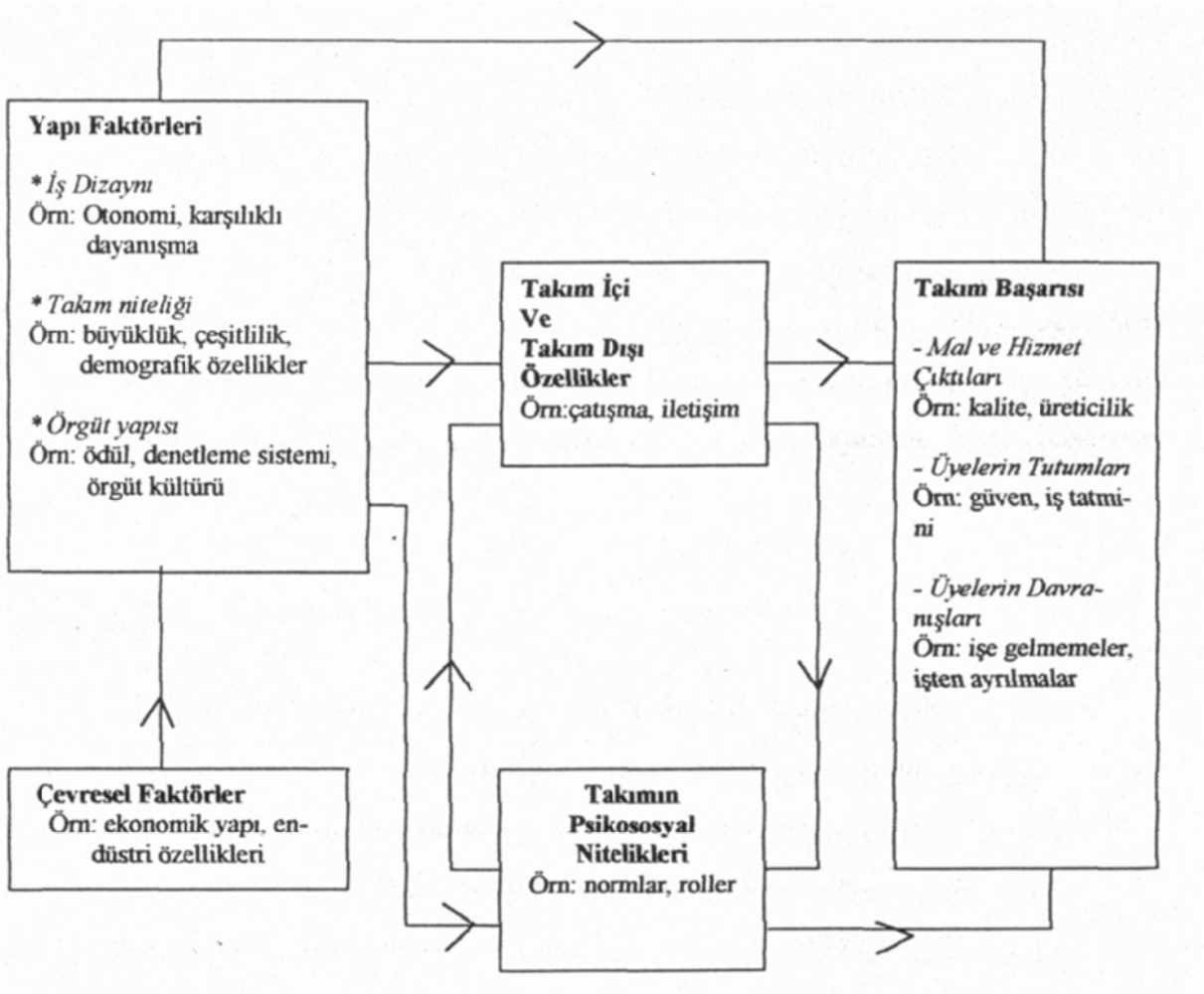
Takımların başarılı olması için gerekli olan bir başka faktör de güçlü bir iletişim sistemidir. Üyelerin birbirleri ile daha etkili iletişim kurabilmesi ve takımın amaçlarına ulaşabilmesi için, takım üyeleri arasında açık ve tutarlı bir iletişimin olması gereklidir. Takım üyeleri arasında güçlü bir iletişimin oluşması demek ilişkinin açık, geri besleme duyarlı, gereksiz sınırlamalardan uzak olması demektir. Takım üyeleri arasındaki ilişkide kurallar olabilir, fakat bu kurallar açıklığı engellemez. Üyelerin takım amacını benimsemesi ve sahip çıkması gerekir. Takım içinde yer alan her üye, amacı benimsemeli, takım amacının gerçekleşmesi için çaba harcamalıdır. Takım üyelerinden bazıları amacı benimsemezse takım başarıyı yakalayamaz. Üyenin amacı benimsemesi için bireysel hedefleri ile takım hedeflerini birleştirmesi, bireysel hedeflerine takım içinde ulaşacağını bilmesi gerekir [Erdoğan, 1991, s.470, 476, 477**].**

Takımların başarılı olabilmesi için öncelikle amaçlarının iyice anlaşılmış olması gereklidir. Başarılı takımlar ortak amaçlarını ölçülebilir, gerçekçi hedeflere çevirebilenlerdir. Gerçekçi hedefleri olan takımlarda üyeler kendilerini hedeflere bağlı hissederler, kendilerinden bekleneni bilir ve birlikte nasıl çalışmaları gerektiğini anlarlar [Kendiroğlu, 2000, s.19, 26, 27].

Takım üyelerinin diğer üyelerle benzer duygu ve düşüncede olması için takım amacını benimsemesi, diğer üyelerle işbirliği yapmaya hazır olması gerekir. Üyeler birbirleri ile ne ölçüde işbirliği yapma arzusunda ve birbirlerine ne ölçüde bağlı olurlarsa takım o denli güçlü bir yapı kazanır [Erdoğan, 1991, s.470, 476, 477**].**

Takımları sıradan gruplardan ayıran özelliklerden birisi üyelerin birbirlerine üst düzeyde bağlı olmaları, bu bağlılığı takım başarısına yansıtmalarıdır. Üyeler ve takım yöneticisinin tutarlı davranış içerisinde olması gerekir. Takımların sürekliliği için üyelerin, üye-yönetici (koç) ilişkisinin tutarlı olması gerekir. Üyelerin davranışlarındaki tutarsızlık zamanla takım anlayışının

ortadan kalkmasına, takım ruhunun zarar görmesine yol açar. Başarılı bir takım için üyelerin davranışlarında süreklilik, doğru ve yanlış anlayışlarında tutarlılık gerekir. Takım başarısını etkileyen bir diğer önemli faktör de üyeler arasındaki adalet duygusudur. İlişkinin adil olması, sonuçlardan tüm ekip üyelerinin belirlenen esaslara göre yararlanması gerekir. Eğer Takım üyeleri arasında adalet anlayışı olmaz, Takım başarısı veya başarısızlığı sınırlı sayıdaki üyeye mal edilirse ekiplerin uzun ömürlü olması beklenemez [Erdoğan, 1991, s.470, 476, 477]. Takım üyeleri arasında karşılıklı güvenin mevcut bulunması başarı için gerekli faktörlerden bir tanesidir. Üyelerin birbirlerinin yeteneklerine ve tüm üyelerin takımın başarılı olabilmesi için gerekli olan özelliklere sahip olduklarına inanmaları ve birbirleri ile uyum içinde olmaları gereklidir. Daha çok proje geliştirme takımları gibi mevcudiyetini uzun süre sürdürmesi beklenen takım türlerinde takım başarısı takım üyeleri arasındaki bireysel uygunluğa bağlıdır.



Şekil 6. Başarılı Takım Modeli [Kendiroğlu, 2000]

Çevresel faktörlerin içerisinde örgütün içinde bulunduğu ekonomik yapı ya da endüstrinin özellikleri yer alır. Modele göre çevresel faktörler iş dizaynı, takım niteliği ve örgüt yapısından oluşan yapıyla ilgili faktörler üzerinde direkt olarak etkili olmaktadırlar. Yapı faktörleri yöneticilerin düzeltip değiştirebileceği faktörler arasında yer alır. İş dizaynının otonomiye veya karşılıklı dayanışmaya göre oluşturulmuş olması, takım üyelerinin kişisel özelliklerinin çeşitliliği, demografik faktörler, örgütün ödül ve denetleme sistemi, örgüt kültürü, örgüt stratejisi gibi faktörler yapı ile ilgili faktörlere örnek oluşturmaktadır. Bu faktörler takım başarısını doğrudan etkilediği gibi, takım içi ve dışı özellikler ile takımın psikososyal özellikleri üzerinde etkili olarak dolaylı yoldan da takımın başarısını etkilemektedir. Takım içi ve dışı faktörler üyelerin kendi aralarındaki veya takım dışı kişilerle iletişim, çatışma gibi ilişkilerini ifade eder [Kendiroğlu, 2000, s.19, 26, 27].

Takımın psikososyal nitelikleri ise normlar, roller, üyelerin paylaştığı inanç ve değerler gibi kavramlardan oluşur. Takım içi ve dışı özellikler ile takımın psikososyal nitelikleri birbirinden etkilenirken, aynı zamanda takım başarısını da etkilerler [Kendiroğlu, 2000, s.19, 26, 27].

4.7 Takımlarda Başarısızlık

Takımların başarısızlığı, hedeflerine ulaşamadan dağılması gibi durumlar yaşanmaktadır. Takımların başarısız olma nedenleri, değişik kaynaklardan incelenerek, aşağıdaki başlıklar altında bir araya getirilmiştir.

a) Üyelerin İnançsızlığı: Takım üyeleri hedeflere ulaşma veya takım üyelerinin amaç için sergiledikleri davranışlara karşı inançlarını yitirirlerse takım başarısız olur. Öncelikle üyelerin bazıları mevcut işleyişin takım amacına uygun olmadığına inanıyorsa, bazı üyelerin takım anlayışının dışında davrandığını düşünüyorsa takım başarısızlığı ortaya çıkacaktır. İnancını yitiren üyeler başarı için çaba harcamayacak, takım başarısına katkıda bulunmayacak, ilk fırsatta da takımdan ayrılmanın yolunu arayacaktır [Erdoğan, 1991, s.470, 476, 477**].**

b) Yönetimsel Engeller: Yöneticiler çoğu zaman kendisini ekibin tehdidi altında görürler. Bir çalışma gurubunun başında nezaretçi olamadan işi yapabilmesi durumunda nezaretine gerek duyulamayacağını düşünüp, güç ve konumlarını yitirmekten korkarlar. Yöneticiler karar almaları ve gereken işleri yapmaları için takımlara genellikle yetki vermezler. Yetki ve sorumluluk verilmemesi takımı olumsuz etkiler. Çünkü takımlar gerekli güç olmadan gelişemezler ve çoğu kez çaba harcamaktan vazgeçerler [Weiss Donald, 1998, s.8, 22].

c) Yanlış Oluşum: Takımın yanlış oluşumu başarısızlık için bilinen temel faktörlerdendir. Takım üyelerinin özellikleri ve beklentileri ile takım amacının uyumsuz olması yanlış oluşumdur ve bu tür takımlar başarılı olamaz. Benzer şekilde üyelerin özellik ve yeteneklerinin takım davranışı açısından uyumsuz olması da takım başarısının önündeki engellerdendir. Uzmanlık ve bilgi dereceleri birbirine çok yakın olan üyelerin oluşturduğu ekipler genellikle başarısız olur. Takım uygun bir çevrede ve zamanda oluşturulmazsa yine başarısız olacaktır [Erdoğan, 1991, s.470, 476, 477**].**

d) İşçiden Kaynaklanan Engeller: Yönetim dışındaki işgörenler takım çalışmasının taleplerini çoğu zaman tehdit olarak görürler. Statülerini kaybetmekten ve kalabalığın içinde yitip gitmekten korkarlar. Bireysel katkı ve etkilerini yutan bir ekibin basit üyeleri oldukları durumda takdir, ödül ve promosyonları kaçıracaklarından kaygılanırlar. Kendi eylemlerinin sorumluluklarını üstlenmek de bazen onları korkutur [Weiss Donald, 1998, s.8, 22].

e) Geribildirim Eksikliği: Takım, çalışmalarının etkinliğinden emin olunamıyorsa, ara dönemlerde yeterli geribildirim alınamıyorsa, doğru yolda olup olmadığından kuşkuya düşer. Takımlar toplumun küçük bir parçasıdır. Her insan topluluğunda meydana gelen sorunlar takımlarda da oluşabilir. Ancak takımların farkı belirli bir görevi yerine getirmek için bir araya gelmiş olmalarıdır. O nedenle başarılı ve sürdürülebilir takımlar oluşturabilmek için takımlarda geribildirim eksikliği olmamalıdır [Baltaş, 2005, s.29].

f) Üyelerin Hazırlıksız Çalışması: Takım üyeleri günlük düşünen, günlük davranan bir yapıda çalışırlarsa takım yine başarısız olur. Takım toplantılarına hazırlıklı gelmeyen, birlikte fikir üretilmesi gerektiğinde hazırlık yapmak yerine o anda aklına geleni tartışmaya açan kişilerin yer aldığı takımlar başarılı olamaz. Takım üyeleri çözüme özendirilmeli, çözüm önerileri için sürekli olarak hazırlık yapmalı, takım çalışmalarını etkin kılmanın yolunu aramalıdır [Erdoğan, 1991, s.470, 476, 477**].**

g) Takımın Karar Alma Yöntemlerinin Belirsizliği: Takım kendi kararlarını alma yetkisine sahip olmalıdır. Takımın kararları lider tarafından mı alınacak, yoksa kararlar, ekip üyelerine danışılıp mı alınacak, tüm üyelerin katılımıyla mı yoksa oybirliği sağlanarak mı oluşacağı takımın kurulum aşamasında belirlenmelidir. Bu konulardaki belirsizlik takım başarısını olumsuz yönde etkiler [Baltaş, 2005, s.29].

h) Sık Amaç Değiştirme: Şüphesiz takımlar bir amaç için oluşturulur. Zamanla da amaçların değiştirilmesi ve güncellenmesi gerekir. Ancak bir takım sık amaç değiştiriyorsa, bir hedefi gerçekleştirmeden başka hedefler için düzenleme yapıyorsa üyelerin inancı azalır. Sık amaç değiştirmede üyelerin kişisel hedefleri ile takım hedefleri arasındaki bağ sınırlanır. Bu tür takımlar genellikle başarısız olur [Erdoğan, 1991, s.470, 476, 477**].**

ı) İşleyiş Kurallarına Uymama: Takım yönetimi de bir dizi kurallara bağlıdır. Takım olmak demek kuralsız çalışmak veya kuralı sık değiştirmek demek değildir. Her sosyal toplulukta olduğu gibi takım yönetiminde de özel kuralların olması gerekir. Özellikle işletme içinde oluşturulan takımların mutlaka oluşum ve işleyiş kuralları vardır. Bazı üyelerin veya üyelerin çoğunluğunun kurallara uymaması takımda başarısızlığa yol açar. Takım başarısı için kurallar benimsenmeli ve uygulanmalıdır. Üyelerin benimsemediği, takım amacına hizmet etmeyen kurallar başarısızlığı doğuracağı gibi, kurallara uymama da takım başarısını engelleyecektir [Erdoğan, 1991, s.470, 476, 477**].**

i) Baskın Üyelerin Çokluğu: Bir takım içerisinde çeşitli nedenlerle baskın üyeler oluşur, bu üyeler zamanla diğerlerine hükmetmeye başlarsa takım başarısı sınırlanır veya yok olur. Takım içinde prensip olarak her üyenin

sorumluluğu eşittir veya her üye takım başarısından sorumludur. Bazı üyeler baskın olur, kendi düşünce ve davranışlarını ön plana çıkarmak takımı kendi isteklerine göre şekillendirmek isterler bu nedenden dolayı takım içerisinde kurallar baskın üyelere karşı esnek tutulursa takım başarısı azalacak veya yok olacaktır. Takımlar adil yönetim sistemi ile yaşar, bu nedenle baskınlık amaca uygun olmalı diğer üyelerce de kabul görmelidir. Diğer üyelerin kabul etmediği baskınlık takımların dağılmasına sebep olur [Erdoğan, 1991, s.470, 476, 477**].**

j) Yanlış Toplantı İdaresi: Takım koçunun toplantıları yanlış idare etmesi de takım başarısızlığında bir etken olarak karşımıza çıkar. Toplantı ortamında sürekli aynı kişilere söz, belirli kişilerin düşüncelerini karara dönüştürme, bazı üyelere karşı adil davranmama takım başarısını engeller. Toplantıyı idare eden kişi adil olmak, tüm takım üyelerinin görüş ve düşüncelerini almak durumundadır. Toplantı ortamındaki baskınlık zamanla takımlarda koçluğun el değiştirmesine ve bölünmesine yol açar [Erdoğan, 1991, s.470, 476, 477**].**

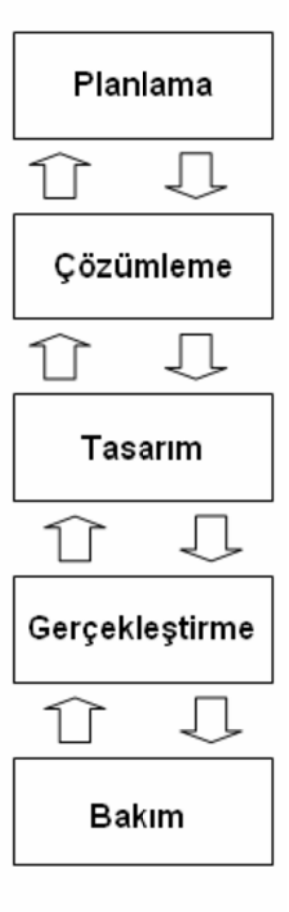
4.8 Yazılım Geliştirme Takımları

Bilgisayar yazılımlarında son yıllarda büyük gelişmeler yaşanmıştır. Geçmişte hayal bile edemeyeceğimiz şeyler normal olarak karşılanır olmuştur. Yapay zekâ kullanılarak, eskiden bilim kurgu filmlerinde gördüğümüz verilen komutları dinleyen, dinlediklerini yorumlayan ve mantıklı kararlar verebilen yani insan gibi düşünebilen bilgisayar programları geliştirilmiştir. Bilgisayar yazılımlarında gittikçe karmaşıklaşmış ve iç içe girmiş bir yapı oluşturmuştur.

Büyük ölçekli ve karmaşık yazılımlar tek bir kişinin bireysel çalışması ile yapılamayacak karmaşık bir hal almıştır. Bu nedenle işletmeler başarılı olabilmek için takım tabanlı organizasyon yapılarından yararlanmaya başlamışlardır.

Yazılım geliştirme takımları daha etkin ve daha profesyonelce uygulamalar geliştirmek için "Yazılım Geliştirme Yaşam Döngüsü" olarak isimlendirilen bir dizi aktiviteyi uygularlar.

Çok değişik yazılım projeleri, çok değişik nedenlerden dolayı başarısız olabilirler. Bu başarısızlığın nedeni bazen çok kolay tespit edilirken bazen de asla tespit edilemez. Bu yüzden yazılım geliştirmesinde bir patern izlemek ve hata analizinde bu paterni takip ederek hatanın nedenini tespit etmeye çalışmak hem büyük bir kolaylık hem de tekrar etmemesini sağlayacak önlemlerin alınmasında uygun bir ortam yaratacaktır. Bu paterne "Yazılım Geliştirme Yaşam Döngüsü" denilmektedir [Kurnaz, 2003, s.4].

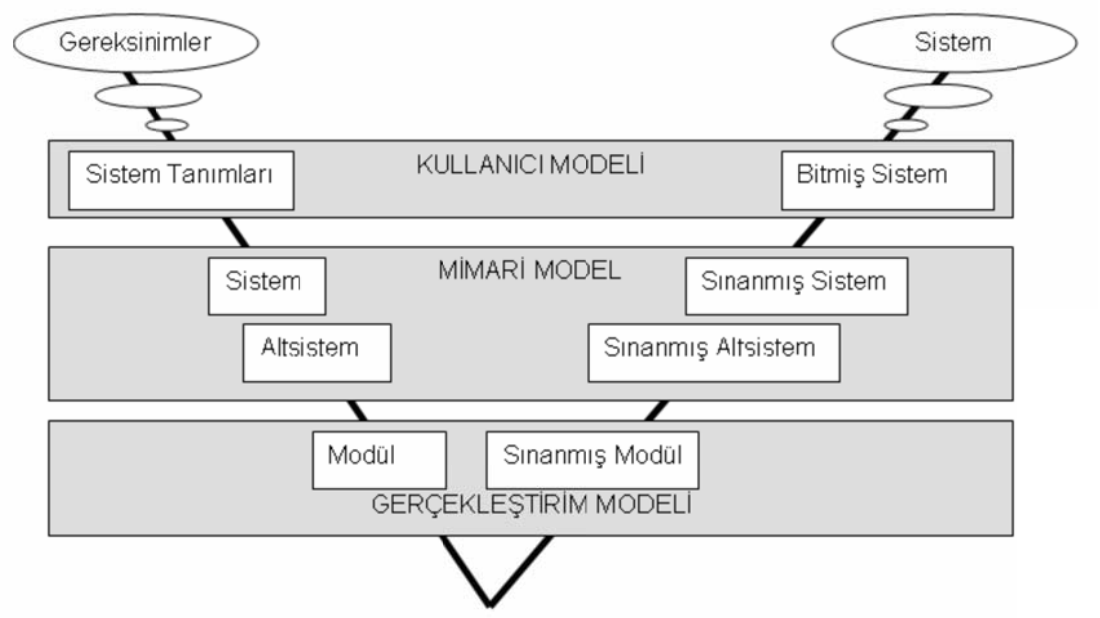


Şekil 7. Yazılım Geliştirme Yaşam Döngüsü [Yılmaz, 2008]

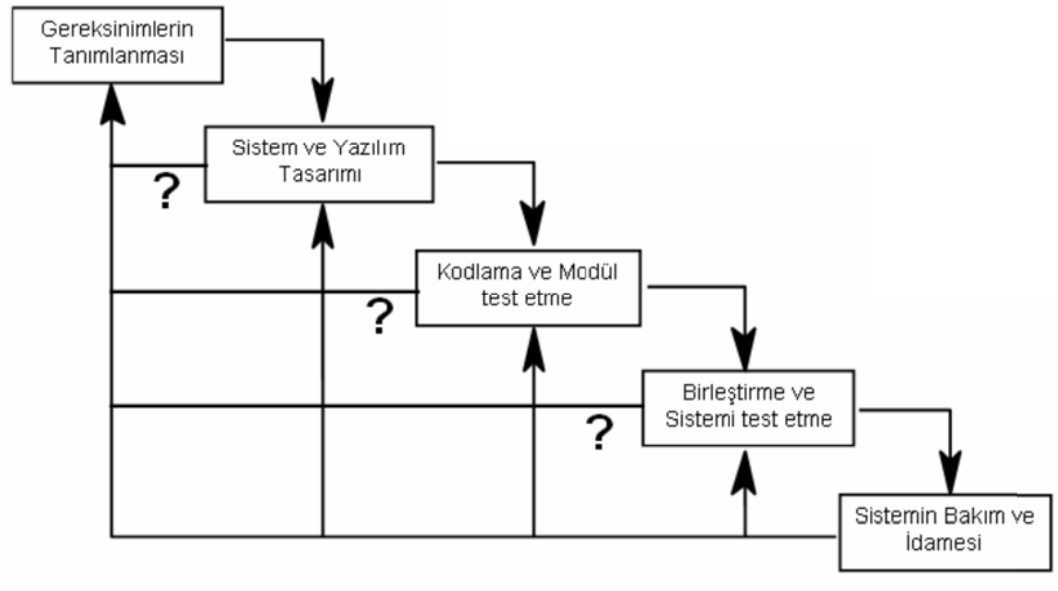
Yazılım Geliştirme Yaşam Döngüsü, herhangi bir yazılımın, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır. Yazılım işlevleriyle ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için söz konusu aşamalar bir döngü biçiminde ele alınır. Döngü içerisinde herhangi bir aşamada geriye dönmek ve ilerlemek söz konusudur. Bu nedenle yazılım yaşam döngüsünün tek yönlü ve doğrusal olduğu düşünülmemelidir. İşlevsel bağımsızlığı (İşlevsel bağımsızlık; modüller arasındaki bağıntıların en aza indirgenmesi anlamına gelip, bir modülde bir hata olduğu zaman diğer modüllerin etkilenmesini engellemek amacıyla yapılır) temin

etmek için modüller arasında ilişkiliği olduğunca azaltmak ve bir modülün yalnızca bir işlev ile görevlendirilmesini sağlamak gerekir. Bu basamakların izleniş sırası ve geri dönüşlerin yapılması yöntemlerine göre bir kaç farklı yöntem geliştirilmiştir.

Bunlara yaşam döngüsü modelleri adı verilir. (Örneğin V Modeli, Şelale Modeli) [Kurnaz, 2003, s.4].



Şekil 8. V Süreç Modeli [Yılmaz, 2007]



Şekil 9. Şelale Modeli [Yılmaz, 2007]

Yukarıdaki şekillerde ifade edilen yaşam döngü modellerine sahip olan yazılım geliştirme takımları, nihayetinde insanlardan kurulu, insanların algılamalarının başarıda önemli rol oynadığı sosyal birliktelikleri ifade etmektedirler. Burada, takım üyelerinin adalet algıları başarı açısından önemli bir kavram olarak açığa çıkmaktadır. İlerleyen bölümde örgütsel adalet kavramı ve ilgili kavramlar üzerinde durulacaktır.

4.9 Kişisel Performans Mı, Takım Performansı Mı?

Kişisel yapılan yazılım projelerinde veya küçük yazılım projelerinde, kişisel performans gerçekten büyük önem taşır. Fakat proje boyutu büyüdükçe ve buna bağlı olarak projede çalışan insan sayışı çoğaldıkça takım performansının önemi ortaya çıkar. Kimse takım çalışması olmadan sadece kişisel performansı ile büyük projeleri başarısızlıktan kurtaramaz. Tabiî ki kişisel performans yüksek olmadan takım performansı yüksek olamaz. Ama bu bir kişinin yüksek performansının bütün takım performansını artıracağı anlamına gelmez. Esas önemli olan her bir bireyin kişisel olarak maksimum performansa ulaşıp, takım içerisinde bir sinerji yaratmasıdır.

"Neden yetenekli ve özel programcıların oluşturduğu takım, gerçek bir takım olamayabilir?". Bunun nedeni bazı programcıların birlikte çalışamamaları veya uyum gösterememeleridir. Bu yeteneklerinden veya özel olmalarından kaynaklanmaz; sorun kişisel konulardandır [K. Todd Stevens, 1998]. Takım içindeki insanların iletişimi güçlü olmadan, gerçek bir uyum sağlanamaz.

4.10 Takım Yaratmak

Takımlar projelere göre yaratılır. Projenin içeriği ne kadar büyük ise takım da o kadar büyük olur. Projeler iki bileşenden var olur diyebiliriz. Birincisi teknik bileşenler; süreçler, prosedürler, kullanılan araçlar, metodolojiler, kullanılan diller gibi. İkincisi ise organik bileşenler; insanlar, motivasyon, düşünceler gibi. Projelerde çoğu zaman teknik bileşenler daha çok önem arz etmektedir. Projenin başından sonuna teknik konularda birçok toplantı birçok değişiklik yaşanır.

Organik bileşenler ise gerekli mevkilere insanlar atandıktan sonra unutulmaktadırlar. Gerekli önem sağlanmamaktadır.

Yazılım takımı oluşturulurken üstesinden gelinmesi gereken bazı noktalar unutulmamalıdır. Bunlar;

- İnsanlar farklı farklıdır ve yazılım geliştiriciler farklı insan tipleri ile çalışmaya alışkın olmayabilirler
- İnsan ilişkilerinde zayıf olan yazılım geliştiriciler olabilir. Hâlbuki bu tarz yazılım geliştiriciler teknoloji konusunda gerekli bilgi birikimine sahip olabilirler
- Düşüncelerde farklılıklar olabilir. Bu da sistem kavramada veya çözümünde farklılıklara yol açabilir
- Takım çalışmasına yatkın olmayan çalışanlar olabilir.

Takımları sağlam şekilde kurmak için aşağıdaki noktalar dikkate alınmalıdır. Bunlar;

- Ortak bir dil oluşturulmalı. Müşteri ilişkileri, takım iletişimi, karar verme yeteneği gibi
- Ortak bir hedef oluşturulmalı
- Başarılar edinilmeli ve paylaşılmalı
- Takım üyeleri arasında bilgi alışverişi olmalı. Takımdaki herkes herkesin ne yaptığını bilmeli, nasıl öğrendiklerini paylaşmalı ve nasıl başarılı olunduğunu anlatmalı
- Mutlaka bireysel ve takım olarak kazançlar elde edilmeli ve paylaşılmalı

Takım içi koordinasyon da önemli noktalardan biridir. İnsanlar, görevler ve kaynaklar arasında koordinasyonu sağlamak gerçekten zor bir işlemdir. Büyük ölçekli yazılım projelerinde birden çok takım bulunabilir. Aynı ürün üstünde paralel olarak çalışan birçok kişi olabilir. Günümüzün gelişmiş yazılım mühendisliği araçlarına rağmen, koordinasyon insan ve kişisel faktörlerden dolayı problem olmaya devam ediyor [J. Alberto Espinosa at al, 2002]

Yazılım takımlarını kurarken veya rollerini analiz ederken R. Meredith Belbin tarafından tanımlanan Belbin Takım Rolleri yöntemi kullanılabilir. Belbin

rolleri kullanılarak takım performansı geliştirilebilir, başarılı takımlar kurulabilir [K. Todd Stevens, 1998].

4.11 Yazılım Takımı ve İçinde Bulundurduğu Roller

Bir yazılım projesinde başarılı olmak için en önemli nokta takım olmaktır. Olması gereken yazılım şirketini anlatırken bir futbol kulübünü örnek göstererek anlatmaya çalışacağım.

En tepeden başlarsak nasıl futbol kulüplerinin başkanları var ise yazılım şirketlerinin de yöneticileri vardır. Futbol kulübü başkanları teknik konularda bilgili olsalar da hiçbir zaman takıma teknik anlamda karışmazlar. Yazılım şirketlerinde de yönetici olan kişiler teknik açıdan yeterli olmalılar fakat proje akışına çok fazla müdahalede bulunmamalılar. Kulüp başkanı oyuncu alımında teknik direktör ile birlikte seçilen oyunculardan kulübün parasının ve oyuncunun yeteneklerinin yeterli olduğuna inandıkları oyuncuyu alırlar. Aynı şekilde yazılım şirketlerinde de şirket yöneticisi ve proje yöneticisi çalışan yeteneklerini ve maddi durumları değerlendirdikten sonra çalışanları ile alırlar.

Proje yöneticileri aynı birer teknik direktör gibidirler. Teknik direktörler uygulanacak taktikleri verirler, antrenmanları düzenlerler, maçlarda oynayacak oyuncuları seçerler, oyuncu transferlerini yönlendirirler ve daha birçok görevleri vardır. Proje yöneticilerinde de durum aynıdır. Projede çalışacak kişileri seçer, kimin hangi görevi yapacağına karar verir, motive eder, olaylara karşı taktik değişikliklerine karar verir. Unutulmamalıdır ki proje yöneticisi çok iyi bir yönetici olabilir, fakat her işi tek başına yapamaz. Bu noktada Yazılım Kalite Güvence Uzmanları, Yazılım Test Mühendisleri ve Konfigürasyon Yöneticilerinin rolü ortaya çıkmaktadır.

Yazılım projelerinde en önemli görevlerden birkaçını Yazılım Kalite Güvence Uzmanları, Yazılım Test Mühendisleri ve Konfigürasyon Yöneticileri sahiplenir. Bu görevleri futbol kulüpleri ile örneklemek gerekirse, bu görevdeki kişiler birer antrenör gibidirler. Bir futbol takımında değişik pozisyondaki oyuncular için değişik antrenörler bulunur. Takımdaki oyuncuların eksik yönlerini

bulup, teknik direktör bilgisinde o eksiklikleri gidermeye çalışırlar. Yazılım projelerinde de durum aynıdır. Antrenörler gibi, Yazılım Kalite Güvence Uzmanları, Yazılım Test Mühendisleri ve Konfigürasyon Yöneticileri yazılım projesinin daha iyi olması için o projenin eksik yönlerini saptarlar, düzenlerler, kontrol ederler ve takibini sağlarlar. Bu örnekleme ışığında, antrenörler futbol takımı için ne kadar önemli ise Yazılım Kalite Güvence Uzmanları, Yazılım Test Mühendisleri ve Konfigürasyon Yöneticileri de yazılım projeleri için o kadar önemlidir. Projenin daha iyi olması için yazılım mühendislerine birçok konuda katkıda bulunurlar.

Yazılım projesinde çalışan programcılar ise aynı birer futbolcu gibidirler. Asıl işi yapan onlardır. Fakat gerekli teknik desteği Proje Yöneticisinden, Yazılım Kalite Güvence Uzmanlarından, Yazılım Test Mühendislerinden ve Konfigürasyon Yöneticilerinden almadan kesinlikle başarıya ulaşamazlar. Bireysel olarak ne kadar iyi programcılara sahip olunursa olunsun takım ruhu olmadan başarıya ulaşılamayacağı yazılım takımlarına veya futbol takımlarına bakılarak görülebilir [SELBES**,** Yazılım Takımlarında Başarı].

4.12 Yazılım Takım Psikolojisi

Yazılım projelerinde programcılar genelde Yazılım Kalite Güvence Uzmanlarına, Yazılım Test Mühendislerine ve Konfigürasyon Yöneticilerine ekstra iş yaratıyorlarmış gözü ile bakarlar. Hâlbuki istedikleri ve verdikleri işler o takımın ürettiği yazılımın daha kaliteli olması içindir. Dolayısıyla programcılar ve Yazılım Kalite Güvence Uzmanları, Yazılım Test Mühendisleri ve Konfigürasyon Yöneticileri arasında iletişim çok büyük önem taşımaktadır. Bu noktada iletişimi sağlayacak, takım ruhunu her zaman dinamik tutacak bir göreve ihtiyaç duyulmaktadır. Futbol takımı örneğini vermek gerekirse, futbol takımlarında futbol şube sorumlusu görevi vardır. Bu görevdeki kişiler futbol takımına her türlü motivasyonu sağlarlar, manevi sorunları çözmeye çalışırlar, yöneticilerle iletişimi sağlarlar. Motivasyon ve iletişimin çok önemli olduğu yazılım projelerinde de bu tarz bir görevde çalışacak deneyimli, iletişim yeteneği yüksek, ve motivasyon gücü olan bir çalışan olması yazılım projesi için yararlı olabilir [SELBES**,** Yazılım Takımlarında Başarı].

BÖLÜM 5: VERİMLİLİK

5.1 Verimlilik Kavramı ve Anlamı

"Prodüktivite" yaklaşık 200 yıldan beri kullanılagelen bir sözcük olup, Fransızca'da "produire" (üretmek) mastarından türetilmiştir. Türkçeye tam karşılığı aktarılmak istenirse "verimlilik" veya "üretme yeteneği" olarak ifade edilebilir. Prodüktivite, üretkenliğe elverişli nitelikler taşıma anlamına gelmektedir. Oysaki verimlilik, üretkenliğe elverişli durumu anlatmaktan çok, iktisadi faaliyetlerden elde edilen sonuçların ölçülerek, karşılaştırılarak ortaya konduğu olumlu bir sonucu vurgulamaktadır. Ancak prodüktivite ve verimlilik kelimelerinin eşanlamlı olarak kullanıldığı görülmektedir.

Çeşitli yazar ve kurumlar verimliliği şöyle tanımlamışlardır:

- Hammadde ve işçiliğin yararlı mal ve hizmetlere dönüşme oranı.
- Çeşitli kaynakların kullanılışındaki etkenliği saptayan bir ölçüt.
- Ekonomik sistemin, malların tüketiciye ulaşması sürecindeki başarısının ölçütü.
- Kaynakların, insanların gerek duyduğu mal ve hizmetlere dönüştürülmesindeki etkenlik.
- Değişme gücü olan mal ve hizmet üretme yeteneğinin ölçüsü.
- Kullanılan kaynaklar karşılığı elde edilen çıktı miktarı.
- Belli üretim faaliyetleri için çıktılar ile girdiler arasındaki oran [Scherer at al, 1992, s.27, s. 385].

5.2 Verimliliğin Önemi

Verimlilik daha üstün, daha iyi, daha huzurlu bir hayat sağlamak için, var olduğu günden bu yana çaba gösteren insanoğlunu, bu amaçlara ulaştıracak, üretim sürecinden beklenen sonuçları daha olumlu hale getirecek bir araçtır. En genel tanımlamaya göre; verimlilik üretim sürecine dahil edilmiş öğelerin, birbirleriyle karşılıklı etkileşimleri sonucunda, elde edilen çıktıyı optimal noktaya çıkaracak bir miktar (kantite) ilişkisi içerisinde olmalarıdır. Buradan anlaşıldığı

gibi verimlilik; hizmet veya mal üreten bir sürecin, ürettiği çıktı ile bu çıktıyı elde etmek için kullandığı girdi (kaynaklar) arasındaki ilişkiler bütünüdür.

Sınırsız ihtiyaçları karşılayacak kaynakların kıt olması bunların en iyi şekilde değerlendirilmesini gerektirmektedir. Eski iktisat kitaplarında kıt kaynak sayılmayan temiz hava, temiz su, temiz çevre artık kıt kaynak sayılmaktadır. Genel anlamda verimlilik insanın mükemmele ulaşma çabasının bir sonucudur. ILO 2 Ekim 2000'de düzenlenen forumda verimlilik "kullanıcıların talep ettiği ürün ve hizmetlerin hangi etkinlikte üretildiğini" belirleyen bir kavram olarak tanımlanmaktadır. Burada önemli olan talep edilen mal ve hizmetleri üretmek ve bu üretim sırasında kaynakları etkin olarak kullanmaktır. Talep edilmeyen bir ürün üretiliyorsa etkinliğe bakılmaksızın girdilerin israf edildiğini söylemek mümkündür. Eğer etkinlik ihmal ediliyorsa mamulün üretim maliyeti ne kadar düşük olursa olsun verimli olunmadığı sonucuna ulaşılabilir. Bu aşamada çıktıyı büyütme ya da girdiyi azaltma durumu söz konusu olmaktadır. Üretim sırasında yapılan üretim planına göre mevcut durum belirlenerek hedefler saptanır ve verimlilik programı yapılır. Daha sonra verimlilik ölçümü yapılarak gerekli iyileştirmeler gündeme getirilir. Bütün bu süreçlerde nitelikli elemanlar oldukça önemli olmaktadır. Verimlilik konusu büyük ölçüde üretimdeki emek faktörüne bağlıdır. Diğer tüm üretim faktörleri açısından da verimlilik söz konusu olmakla birlikte emek verimliliği yapılan verimlilik tartışmalarında en önemli yeri tutmaktadır [http://www.izto.org.tr/NR/readonlyres/OBF7029E-435E-4E77-A938-BB2B1CC3]

5.3 Verimlilik Ölçüm Yöntemleri

Verimlilik, daha önce de yapılan tanımlamalardan da anlaşılacağı üzere en genel anlamda üretim faaliyeti sonucu elde edilen çıktının girdiye bölünmesiyle bulunan bir katsayıdır. Başka bir ifade ile, üretimin üretim faktörleriyle ilişkilendirilmesi sonucu ortaya çıkan bir değerdir.

Verimlilik formülünde paydadaki değerlerin sabit kabul edilerek en yüksek çıktı miktarlarının elde edilmesi verimliliğin maksimizasyonu olarak adlandırılır. Paydadaki değerler sabit kabul edilerek bu çıktıların en az girdi

miktarları ise gerçekleştirilmesine de verimliliğin minimizasyonu adı verilmektedir.

Verimliliğin söz konusu olabilmesi için pay ve paydadaki değişkenlerin artış veya azalış göstermesi gerekmektedir. Bu durumda paydaki artışın paydadaki artıştan daha büyük olması verimliliğin arttığı sonucunu vermektedir. Pay ve paydadaki azalmalarda verimliliğin hesaplanmasında dikkate alınmaktadır. Ancak, paydaki azalmanın paydadaki azalmadan daha büyük olması halinde verimlilik ölçümü söz konusu olabilmektedir [Canbey Özgüler, 2005].

5.4 Verimlilik Arttırma Yöntemleri

Bir işletmenin verimliliğini artırmak için çok çeşitli tekniklerden yararlanılabilir. Var olan yöntemlerden hangilerinin seçileceği birçok faktöre bağlıdır: İşletmenin uğraşları, sanayii kolu, pazar yapısı, işletmenin zayıf ve kuvvetli yönleri, işletmenin istihdam yapısı, verimlilik artırıcı yöntem için öngörülen süre ve maddi kaynaklar, vb. Yine de bu tekniklerin hemen hemen hepsi çoğunlukla bilgi toplama ve iş etkinliğini artırma amacına yöneliktir.

Verimlilik artırmak için kullanılan yöntemler başlıca iki ana gruba ayrılır:

- Teknik yaklaşımlar yoluyla verimlilik artırma yöntemleri
- Davranışsal yöntemler yoluyla verimlilik artırma yöntemleri

Teknik yaklaşımlar yoluyla verimlilik artırma yöntemlerinin en yaygın olarak kullanılanları iş etüdü, tam zamanında üretim yöntemi, pareto analizi ve fayda/maliyet analizidir.

Verimlilik artışı için kullanılan davranışsal yöntemler "işletme psikolojisini" değiştirmeyi amaçlar. Bunu gerçekleştirmek için de çeşitli organizasyon şekillerinden yararlanırlar. Bu organizasyon şekilleri de çeşitli motivasyon ve katılımı güçlendirme araçlarını kullanırlar. Davranışsal yaklaşımlar yoluyla verimliliğin artırılmasında başarı sağlanması, teknik yaklaşımlar yoluyla verimliliğin artırılması yöntemlerine oranla çok daha zordur. Her şeyden önce

işletmelerde bu konudaki bilgi ve birikim çok azdır. Bu yaklaşım doğrudan insan psikolojisini ele aldığından, bu tekniklerin standart kalıplara sokulması veya kesin sınırlarının çizilmesi çok güçtür. Üstelik insan, psikolojik açısından çok zor kontrol edilen bir varlıktır. Bu da davranışsal tekniklerin uygulanmasını zorlaştırmaktadır. Bir diğer zorluk ise, bu yaklaşımların yardımıyla verimlilik artışı elde edilebileceği konusunda birçok kişinin özellikle yöneticilerin ikna olamamasıdır. Bu da genellikle davranışsal yaklaşıma yönelik tekniklerin güven ve bilgi eksikliğinden dolayı uygulanmaması veya uygulansa bile, gerekli şartlar sağlanmadan uygulanmaları sonucu başarısız olmalarını doğurur.

Verimlilik artırma yöntemlerinin teknik ve davranışsal yaklaşımlar olarak ikiye ayrılmasının sebebi bu tekniklerin birbirlerinin alternatifleri olmadığındandır. Aksine bu teknikler birbirlerinin tamamlayıcısı durumundadırlar. Birçok durumda birlikte uygulanmadıkları takdire etkisiz kalırlar. Örneğin karar verme sürecini aşağıya çekmemiş bir işletmede, tam zamanında üretim tam anlamıyla başarılı olamayacaktır.

Davranışsal yaklaşımlar yoluyla verimlilik artırma yöntemleri olarak şu yöntemler sayılabilir:

- Yaratıcı Düşünmenin Yaygınlaştırılması
- Toplam Kalite Yönetimi
- Kalite Çemberleri Uygulamaları
- Örgüt Geliştirme Tekniği
- Beyin Fırtınası Tekniği
- Nominal Gruplama Tekniği
- Verimlilik Artırıcı Teklif Sistemi (VATS) [Canbey Özgüler, 2005].

5.5 Yeni Bilgi Teknolojileri ve Verimlilik

Belirli ekonomik girdilerden daha çok çıktı elde edilmesi demek olan verimlilik artışı, BİT (Bilişim İletişim Teknolojileri) söz konusu olduğunda hem sermaye yoğunlaşması hem de toplam faktör verimliliği büyümesi yoluyla

verimlilik artışı söz konusu olmaktadır. Sermayenin yoğunlaşması işçi başına daha çok sermayenin varlığı ile açıklanabilir. 1990'ların sonlarından itibaren küreselleşme ve BİT devrimi ile şekillenen yeni ekonomik durumda ortaya çıkan büyüme, verimlilik artışı, firma, endüstri ve ekonomi açısından, küresel rekabeti başarabilmek mücadelesiyle birlikte ortaya çıkmaktadır.

Yeni ekonomi savunucularının savunduğu temel hipotez; uzun dönemde BİT'nin verimlilik artışlarına yol açtığıdır.

Ekonominin hangi hızda büyüyebildiği ile ilgili temel belirleyici verimlilik artışıdır. Yüksek verimlilik yüksek büyüme sağlar. Ulusal ölçekte BİT harcaması ile verimlilik ilişkisi çok açıktır [W.Lewis at al, 2002]. BİT sektöründe kalite artarken fiyatlar azalmaktadır. BİT ekipmanları, yazılım, diğer mal ve hizmetler açısından küresel BİT üretimi olmasına rağmen ülkeler arasında önemli farklılıklar bulunmaktadır.

ABD'de son 25 yılda verimliliğin artması yeni ekonomi tartışmalarını başlatan temel konulardır. ABD'de kişi başına çıktı II. Dünya Savaşından sonra artmaya başlamıştır. Toplam Faktör Verimliliği (TFV)' nin arkasında çok faktör vardır [Scherer at al, 1992, s.27, s. 385].

BİT kullanımının işgücü verimliliği ve milli gelirde yarattığı artış verimlilik tartışmalarının çıkış noktasını oluşturmaktadır. Verimlilik artışı BİT'nden kaynaklanır. Ücretler ve karlar anlamında düşünüldüğünde böyle bir gelişmede ABD'de her ikisininde arttığı gözlenmiştir. Enflasyonun düşük düzeylerde olması ve gelir dağılımının durumu yaşam standardının önemli göstergelerinden olan reel ücretlerde artışa yol açmıştır.

Ekonominin hız limitleri anlamına gelen bu konuda yüksek gelir ve istihdam artışı enflasyon olmaksızın ortaya çıkmıştır. Gözlemler, işgücü verimliliğindeki artışın da eskiye nazaran daha fazla olduğunu göstermektedir. Yüksek işgücü verimi artışı, hızlı teknolojik gelişmeden veya ilave girdilerden kaynaklanıyor olabilir [Gundlach, 2001].

BÖLÜM 6: YAZILIM PROJELERİNDE BAŞARISIZLIK

1960'ların sonlarından bu yana, yazılım projelerinin, geliştirilen sistemlerin tümünün gecikmesine, planlı bütçeyi aşmasına, kullanıcı istekleriyle örtüşmemesine, kullanım, sürdürülebilirlik ve genişletilebilirlik sorunlarına yineleyen bir biçimde yol açması nedeniyle, "Yazılım Krizi" terimi kullanılagelmektedir. Yazılımın elle tutulur olmaması, doğanın alışılagelen fizik kurallarıyla hükmedilmemesi, üretim biçiminin farklılığı gibi nedenlerle, günümüz toplumunda bile, birçok kişi, yönetici ve müşteri tarafından yazılıma hala kuşkuyla bakılmaktadır [Dorfman et al, 1996].

Bununla birlikte, insanlar işlerinde, kararlarında, eğlence, eğitim, finans, iletişim, sağlık, güvenlik, ulaşım, v.b., gibi yaşamın hemen her alanında yazılımlara güvenir olmuşlardır. Bu geniş kullanım, yazılım ağırlıklı sistemleri ve çözmeye çalıştığı problemleri de giderek karmaşıklaştırmaktadır. Akademisyenler, yöneticiler ve yazılım uzmanları anılan bu problemlere daha disiplinli ve yöntembilimsel bir yaklaşımın gerekli olduğu düşüncesiyle "yazılım mühendisliği" disiplinini geliştirmişlerdir. Teknik cephesindeki ilerlemelere karşın, hala gerekli olgunluk seviyesine ulaşılamamıştır. Buna bağlı olarak, yüksek kaliteli yazılım sistemlerinin geliştirilmesi ve ilgili projelerinin yönetilmesi için gerekli olan becerilerde ustalık kazanılamamıştır [Pressman, 1997], [Abdel-Hamid et al, 1991]. Kötü stratejik yönetim ve buna bağlı insan etkenleri başarısızlıkların en önemli nedenlerinden biri olarak gösterilmiştir. Anılan sorunların kaynağı araştırıldığında; politikalar, bütçeleme ve diğer dışsal kısıtlayıcılar gibi yazılım dışı konular yüzünden problemler yaşandığı görülmüştür [Thomsett, 1980]. Ancak, yazılım projelerinin başarılı olması isteniyorsa; yazılım mühendisliğiyle ilgisi olmayan bu konuların da yazılım uzmanlarınca bilinmesi ve yöneticilerce bu sorunlarla başa çıkma yöntemlerinin geliştirilmesi gerekmektedir. Çünkü kötü yönetim yazılım maliyetlerini diğer tüm etkenlerden daha hızlı bir şekilde artırmaktadır [Boehm, 1981].

Sonuç olarak, modern dünyamızın en zorlu alanlarından biri olan yazılımda başarıyı başarısızlıktan ayıran üç temel etken mevcuttur. Bunlar; **teknik, yönetimsel** ve **sosyal** nedenler olarak sınıflandırılabilir.

6.1. Başarısızlık Nedir?

Yazılım projesi

· Tamamlanmadan iptal edilirse,

· Tasarlanmış bütçesini aşarsa,

· Öngörülen tamamlanma süresinden daha uzun sürerse,

· Önceden belirlenen özellik ve işlevlerin daha azını karşılarsa

başarısız olmuş demektir. Yukarıdaki ilk maddenin gerçekleşmesi mutlak başarısızlık *(absolute failure)*, diğerleri ise göreceli başarısızlık *(relative failure)* olarak da adlandırılır.

Yazılım dünyasınca küçük farklılıklarla birlikte benzer anlamlarda kullanılmakta olan diğer terimler şunlardır: Son derece sıkı programa sahip projeleri tanımlamak için Çatırtı Modu *(Crunch Mode)* deyimi kullanılır. Projede yer alanlar tarafından hissedilen baskıları dile getirir [Boddie, 1987]. Ölüm Marşı *(Death March)* deyimi neredeyse olanaksız programa sahip projeler için kullanılır. Proje ekibini saran potansiyel başarısızlık hissini tanımlar [Yourdon, 1997]. Kaçak Proje *(Runaway Project)* deyimi ise iptal olmuş ya da iptale yakın projeleri tanımlamak için kullanılır [Glass**,** 1998], [Cole, 1995, s. 3-5]. Tipik bir projeyi bu terimler cinsinden ifade etmek gerekirse: Yönetim, proje sonuçlarına ilişkin çok kısa bir süre için çok fazla vaatlerde bulunmuşsa, çatırtı moduna girilmiştir. Proje devam ederken, kısa bir süre sonra proje ekibi gerçekleşmesi imkansız bu hedefe ulaşmak uğruna, kendisini ölüm marşında bulur. Projenin başarılı olamayacağı apaçık olduğunda proje artık kaçak bir projedir.

Başarı ve başarısızlık örüntüleri *(pattern)* cinsinden, yazılım dünyasını ilgilendiren en önemli unsurlar aşağıdadır [Jones, 1996]:

· Yazılım projelerinin öngörülen teslim tarihleri ve zamanında pazara yetiştirme düşüncesi veya kaygısı,

· Yazılım uygulamasını geliştirme maliyeti ve gereksinim duyulan kaynaklar,

· Yazılımın teslimindeki kalite ve güvenirlik düzeyi,

· Yazılımın işleyişindeki öğrenme ve kullanım kolaylığı,

· Problem olduğunda müşteri desteği ve hizmet seviyesi,

· Uygulamanın olgunluk kazanması sırasında değişiklik ve sürdürülebilirlik kolaylıkları.

Bir projenin başarılı sayılması için yukarıdaki kıstasların tümünü sağlaması gerekmesine karşın, ilk üçünü yerine getirememesi başarısız olması için yeterlidir.

6.2. Yazılım Felaketlerinin Temel Nedenleri

Araştırmanın izleyen bölümlerinde, yazılım projelerindeki başarısızlık nedenleri; teknik, yönetimsel ve sosyal başlıkları altında incelenecektir.

6.2.1 Teknik Nedenler

Yazılım projelerinin başarısızlığına neden olan pek çok etken bulunmaktadır. Bu etkenler arasında, yazılımın doğasından kaynaklanan ve yazılımın büyüklüğü ile doğrudan ilintili karmaşıklık *(complexity)* ön plana çıkmaktadır. Tüm yazılım alt endüstrilerinde, büyük sistemlerin iptal edilme ya da gecikmiş bir şekilde tamamlanabilme olasılıkları küçük uygulamalara oranla daha büyüktür. Tablo-1, projelerin zaman planlamaları açısından sonuçlanma olasılıklarını göstermektedir [Jones, 1996], [Jones, 1998, s. 13-17].

Function Points	Erken	Zamanında	Gecikmiş	İptal Edilmiş
1 FP	14.68%	83.16%	1.92%	0.25%
10 FP	11.08%	81.25%	5.67%	2.00%
100 FP	6.06%	74.77%	11.83%	7.33%
1,000 FP	1.24%	60.76%	17.67%	20.33%
10,000 FP	0.14%	28.03%	23.83%	48.00%
100,000 FP	0.00%	13.67%	21.33%	65.00%
Ortalama	5.53%	56.94%	13.71%	23.82%

Yazılım projelerinin başarısızlığında sıkça karşılaşılan teknik nedenler aşağıda sıralanmıştır:

· Etkili olmayan yazılım teknolojilerinin kullanılması

· Uygun olmayan yazılım araçlarının seçimi

· İş süreçlerinin belirlenmemesi

· Çözümlemenin *(analysis)* uygun bir şekilde gerçekleştirilmemesi

· Geçmişe yönelik yazılım ölçüm verilerinin bulunmaması

· Etkin bir mimarinin seçilmemesi

· Etkin geliştirme yöntemlerinin kullanılmaması

· Tasarım gözden geçirmelerinin *(design review)* yapılmaması

· Kod denetimlerinin *(code inspection)* yapılmaması

· Uygun olmayan, disiplinsiz sınama *(testing)* yöntemlerinin uygulanması

· Belirtim *(specification) ve* tasarımın el ile yapılması

· Düzenleşim *(configuration)* denetiminin uygulanmaması

· Kullanıcı gereksinimlerinin belirsizliği veya %30'dan daha fazla değişmesi

· Uygun programlama dillerinin kullanılmaması

· Aşırı ve ölçülmemiş karmaşıklık düzeyi

· Yeniden kullanılabilir bileşenlerin kullanılmaması

· Uygun veritabanı planlamalarının ve tasarımının yapılmaması

· Yeni teknolojileri kullanmaya geçişin birdenbire olması, deneyim eksikliği

6.2.2 Yönetimsel Nedenler

Yazılım mühendisliğinin teknik alanlarında pekçok süreç modeli, araç ve yöntemin geliştirilmiş olmasına karşın, yönetim alanında aynı düzeye erişildiği söylenemez. Yazılım projeleri sadece kötü hatalar, kötü şans, kötü programcılar ya da kötü test personeli yüzünden değil çoğunlukla yönetimsel nedenlerden başarısızlığa uğramaktadır [Jones, 1998, s. 13-17]. Üst yönetimin genellikle yazılım konularına uzak oluşu ve kendi uzmanlık alanlarında göstermiş oldukları yetkeyi (otorite) bu alanda yeteri kadar uygulayamamaları yönetimsel sorunların başında gelmektedir. Başarılı yazılım projeleri ile başarısız projeler arasındaki en önemli fark iki sözcük ile özetlenebilir: "Sürpriz yok!" Başarılı projelerde tahmin,

planlama ve kalite kontrol tüm proje süresi boyunca gözönünde bulundurulur. Bu nedenle, beklenmedik gecikmelere rastlanmaz. Yazılım projelerini başarısızlığa iten diğer yönetimsel nedenlerden başlıcaları aşağıdadır:

· Üst yönetim, pazarlama bölümü veya proje yöneticisi tarafından saflıkla verilen sözler ve diğer politik baskılar

· Gerçekçi olmayan kestirimlere bağlı olarak çok yoğun bir zaman baskısı

· Büyüklük, işgücü ve maliyet kestirimlerinin üst yönetimce geri çevrilmesi

· Üst yönetimin bilgisizliği ve ilgisizliği

· Proje yönetimi uygulamalarındaki yanlışlıklar

· Proje amacının ve kapsamının açık bir şekilde belirlenmemesi

· Proje yöneticisinin yetersizliği

· Niteliksiz teknik personelin çalıştırılması

· Etkin olmayan geliştirme süreçlerinin bulunması

· Geliştirme ekibinde deneyimli personelin yer almaması

· Otomatik kestirim araçlarının kullanılmaması

· Otomatik planlama araçlarının kullanılmaması

· Otomatik proje gelişim izleme yöntemlerinin kullanılmaması

· Etkin olmayan proje değişim yönetimi usullerinin kullanılması

· Kalite kontrol yaklaşımlarının az ya da hiç kullanılmaması

· İşe uygun personel, personele uygun iş ölçütünün gözardı edilmesi

· Yapılan yazılım proje yönetim planına uyulmaması

· Uygun olmayan metriklerin toplanması veya hiç toplanmaması

· Bazı kritik görevlerde (kalite güvence, sınama, planlama, kestirim, veritabanı yöneticisi, tasarım, değişim kontrolü, hata ayıklama, v.b.) uzmanlar yerine sıradan teknik personelin kullanılması

· Başarısızlık uyarı imlemlerinin bulunmaması ya da geç fark edilmesi

· Belgelemenin *(documentation)* uygun bir şekilde yapılmaması

· Geliştirme ekibinin iş süreçleri ve sorun sahası üzerine bilgilendirilmemeleri

· Pazarın küreselleşmesinden kaynaklanan yoğun çekişme ortamı

· Yeni teknolojilerin üretilmesinden kaynaklanan çekişme ortamı

· Yazılım/donanım sağlayıcılarının zayıf performans sergilemeleri
· Beklenmedik ve öngörülemeyen bunalımla

6.2.3 Sosyal Nedenler

Yazılım projelerinin başarısızlık örüntüleri arasında bir de kültürel ve sosyal etkenler bulunmaktadır. Hem alıcı hem de sağlayıcı açısından, kurumun kültürü ve sosyal yapısı projenin başarısına doğrudan etki yapmaktadır. Müşteri tarafı genellikle yoğun bir zaman baskısına yönelir. Kuruluşların kültür ve yapılarına bağlı olarak, üst yönetimin projeye yerinde olmayan tavsiyeleri, çoğunlukla projelerin başarısızlığında etkendir. Bilişim projelerinin çoğu beraberinde bazı köklü değişiklikleri de getirir. Ancak, müşterinin bu değişime tepkisi genellikle olumsuz yönde gerçekleşir. Bunda; alışılagelmiş yöntemlerin dışına çıkmanın getireceği tedirginlik, mevcut sistemde ellerinde bulundurdukları güçten ödün verme zorunda kalma duygusu ve işini yitirme korkusu büyük rol oynamaktadır. Buna bağlı olarak, yeni sistemin geliştirilmesi için yürütülen projeye destek olmak yerine, açıkça veya dolaylı bir biçimde karşı çıkıldığı da çeşitli projelerde gözlemlenmiştir [Yourdon**,** 1997] [Glass**,** 1998] [Cole, 1995, s. 3-5] [Jones, 1996] [Weinberg, 1997] [The Standish Group International, 2003] [Flowers**,** 1996] [Brooks, 1995] **[**DeMarco et al, 1999]. Başarısız projelerde sıklıkla görülen sosyal nedenlerin en önemlileri şunlardır:

· Takım içi iletişim ortamının zayıf olması
· Geliştirici personelin çalışma ortamının uygunsuz olması
· Alıcı ile zayıf bağlantı kurulması sonucu girdilerinin alınamaması
· Alıcıların gerçekçi olmayan beklentileri
· Alıcı ve diğer paydaşlar arasında şiddetli sürtüşmelerin yaşanması
· Deneyimsizlikten kaynaklanan iyimserlik
· Deneyimsiz yönetim ekibi
· Deneyimsiz teknik ekip
· Deneyimsiz müşteri ile çalışılması
· Teknolojik gelişmelerden habersiz olunması
· Şirket politikalarında ayrımcı uygulamaların bulunması
· Yönetimin soğukluğundan kaynaklanan mesafeli iletişim
· Zayıf rapor verme yapısı

· Teknik personelin sürekli zaman baskısı altında çalışması

· Bölüm ve birimler arası iletişim ve desteğin olmaması

· Karmaşık bağımlılıkların mevcut olması

· Hırsa bağlı yeni yeni hedeflerin ortaya konması

· İşsiz kalma korkusu

· Öç alma duygusu ve düşmanca tavır

6.2.4 Diğer Nedenler

Yukarıda belirtilen nedenlerin dışında, sıklıkla bütün projelerde görülmeyen, ancak projelerin başarısızlığına neden olabilecek diğer etkenler de aşağıda sıralanmıştır:

· Coğrafi olarak birbirinden uzak ve birden fazla şehir/ülke için geliştirilecek projeler

· Proje sözleşmesinin dağıtık ve birden fazla alt-yüklenicinin sorumlu olduğu projeler

· Hem yazılım hem de donanımın eşzamanlı olarak birlikte geliştirildiği hibrid projeler

· Gerçek zamanlılık kısıtlayıcısının en yoğun olduğu projeler (uzay, havacılık, güdümlü mermi, v.b.)

· Yasal kısıtlayıcılar altında geliştirilen projeler

· İhaleyi alma kriterinin en düşük fiyat teklifi olduğu sözleşmeli projeler

· Parasal olarak güçsüz, riskli fona sahip şirketlerin projeleri

· Köklü personel indirimine giden veya kilit personelini işten çıkaran şirketlerin yönettiği projeler

· Personel aşınma oranı %40'ı aşan projeler

· Personel artışı ayda %10'dan fazla olan projeler

· Teknik personelin genel becerilerinin dışındaki alanlarda yürütülen projeler

· Yönetim ekibinin genel deneyimlerinin dışındaki alanlarda yürütülen projeler

· Teknik personel değişim hızının yüksek olduğu projeler

Yazılım projelerinde başarısızlığın nedenlerinin anlaşılması ve bunlardan dersler çıkarılması, gelecekteki projelerinbaşarısını etkiler. Biten her projenin ardından nelerin iyi, nelerin kötü gittiğinin çözümlemesinin yapılacağı toplantı ve çalışmalar *(post-mortem)*, projede kullanılan ölçün *(standard),* yöntem, ölçüm ve verilerin kaydedilerek geleceğe ışık tutacak bilgilere dönüştürülmesi ve bu bilginin kurumsallaştırılması için en iyi fırsattır.

BÖLÜM 7: TAKIM ÇALIŞMASININ PROJELERİNİN VERİMLİLİĞİNE ETKİSİNİ İNCELEYEN BİR UYGULAMA

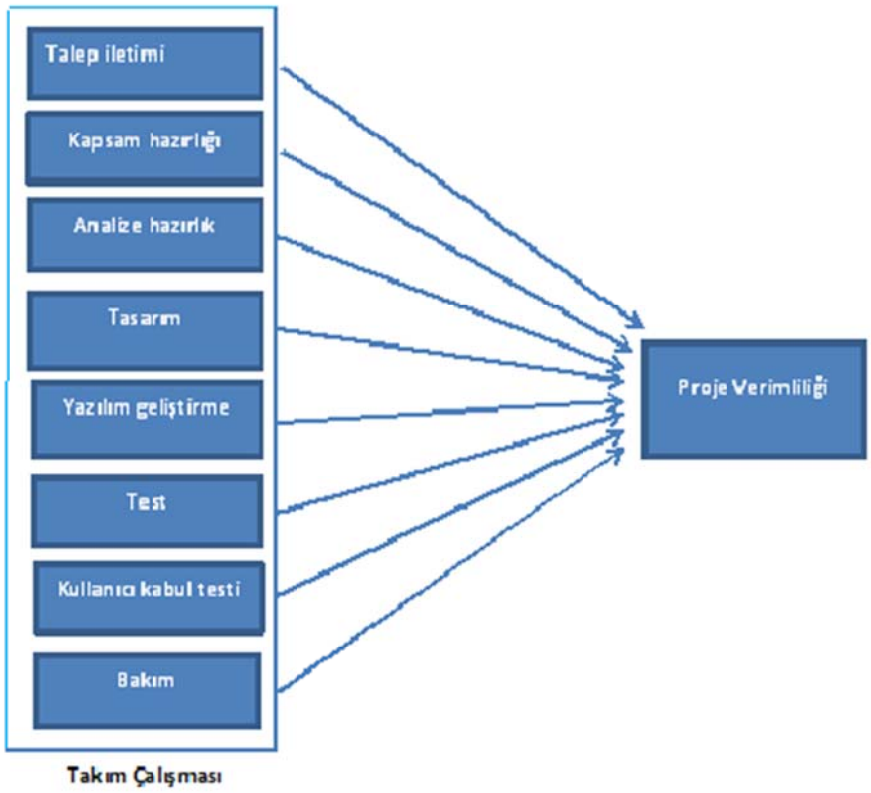
7.1. Çalışmanın Amacı

Araştırmada, takım çalışması ve yazılımcının yazılım projesi verimliliğine olan etkisinin belirlenmesi amaçlanmıştır.

Ayrıca yazılım geliştirme sürecinin aşamalarından talep iletimi, kapsam hazırlığı, analiz, tasarım, yazılım geliştirme, test, kullanıcı kabul testi, bakım aşamaları ile, çalışanların kişisel özelliklerinin ve yazılım geliştirme süresince kullanılan teknik ve araçların proje verimliliği değerlendirmesi bakımından aralarında ilişki oluşturup oluşturmadığı incelenmiştir.

7.2. Araştırmanın Modeli ve Hipotezleri

Araştırmanın modeli Şekil 11'de görülmektedir.



Şekil 10 Araştırmanın Kavramsal Modeli

Çoklu değişken doğrusal denklem :

$y = \beta_0 + \beta_1 x + \beta_2 x + \beta_3 x + \beta_4 x + \beta_5 x + \beta_6 x + \beta_7 x + \beta_8 x + e$

$y = \beta_0 + \beta_1 * TI + \beta_2 * KH + \beta 3 * AH + \beta_4 * T + \beta_5 * YG + \beta_6 * TA + \beta_7 * KKT + \beta_8 * BA + error$

TI = Talep İletimi
KH = Kapsam Hazırlığı
AH = Analize Hazırlık
T = Tasarım
YG = Yazılım Geliştirme
TA = Test
KKT = Kullanıcı Kabul Testi
BA = Bakım Aşaması

Araştırmanın amacına yönelik 10 hipotez oluşturulmuştur.

$H_{TÇ}$:Takım çalışması proje verimliliğini olumlu olarak etkilemektedir.

H_Y:Yazılımcı proje verimliliğini olumlu olarak etkilemektedir.

$H_{Tİ}$:Talep iletimi aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_{KH}:Kapsam hazırlığı aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_{AH}:Analize hazırlık aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_T:Tasarım aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_{YG}:Yazılım geliştirme aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_{TA}:Test aşaması proje verimliliğini olumlu olarak etkilemektedir.

H_{KKT}:Kullanıcı kabul testi proje verimliliğini olumlu olarak etkilemektedir.

H_{BA}:Bakım aşaması proje verimliliğini olumlu olarak etkilemektedir.

7.3 Anket Formunun Hazırlanması

Anketler literatür analizinden elde edilen bilgiler ışığında geliştirilmiştir. Anket formu dört bölümden oluşmaktadır. Birinci bölümü yazılım geliştirme sürecinde çalışan kişilerin kişisel özelliklerine ilişkin sorular, ikinci bölümü yazılım geliştirme süresince kullanılan teknik ve araçlara ilişkin sorular oluşturmaktadır. Üçüncü bölümde ise talep iletimi, kapsam hazırlığı, analize hazırlık, tasarım, yazılım geliştirme, test, kullanıcı kabul testi ve bakım aşamalarının verimliliğe olan etkisini ölçmek için 63 soru 5'li likert ölçeği kullanılarak hazırlanmıştır.

Anket soruları hazırlanmadan önce konu ile ilgili 3 ay boyunca genel bir literatür araştırması yapılmıştır. Anket 3 aylık bir süreçte oluşturulmuş, 2 aylık bir süreçte katılımcılara uygulanmıştır.

Anket özel bir bankanın Bilgi Teknolojileri bölümünde çalışan 451 kişiye ulaştırılmıştır. 140 anket başarılı bir şekilde tamamlanmıştır. 16 anket eksik ve hatalı işaretlemelerden dolayı analiz kısmına alınmamıştır.

7.4. Araştırmada Kullanılan Ölçekler

Yazılım geliştirme süreci ile ilgili sorulan 63 soruda 5-li Likert Ölçeği kullanılmıştır.

5-li Likert Ölçeğinin kullanılması için oluşturulan faktörler şelale modelindeki faktörlerden dikkate alınarak oluşturulmuştur. Bu faktörler dikkate alınarak soru içerikleri tarafımızca geliştirilmiştir.

7.5 Araştırma Verilerinin Analizi

Araştırma verilerinin analizinde, SPSS 17.0 programı kullanılmıştır. Analizler sırasıyla, anketi cevaplayanların demografik özelliklerine ait frekans dağılımları, normallik testi, faktör analizi, güvenilirlik analizi, korelasyon katsayıları analizi ve regresyon analizinden oluşmaktadır.

7.5.1 Veri Toplama Yöntemi ve Analizi

Veri toplama metodu olarak sosyal bilimlerde sıkça kullanılan Anket Yöntemi seçilmiştir. Anket formları Mart 2011 – Nisan 2011 döneminde seçilen özel bir kurum çalışanlarına yüz yüze ve mail aracılığıyla uygulanmıştır. İlgili anket şirket çalışanlarına dağıtılarak işaretlemeleri istenmiş ve 140 anket doldurtulmuştur.

Araştırmanın amacına uygun olarak yapılan analizler ve bu analizlerden çıkan sonuçlar aşağıda verilmiştir.

7.5.2 Araştırmaya Katılanlarla İlgili Genel Bilgiler

Araştırmaya katılanların yaş, cinsiyet, eğitim durumu, firmadaki pozisyonu ve firmanın çalışma alanına dair bilgiler aşağıda belirtilmektedir.

Frekans Tabloları

- Kişisel sorular

Tablo 2. Araştırmaya katılan çalışanların demografik bilgileri ile ilgili frekans dağılımı tablosu

Cinsiyet	Frekans	%
Bay	93	66,4
Bayan	47	33,6
Toplam	140	100,0

Yaş	Frekans	%
18-24	20	14,3
25-31	80	57,1
32-38	30	21,4
39-45	10	7,1
Toplam	140	100,0

Eğitim Durumu	Frekans	%
Üniversite	112	80,0
Lisansüstü / Doktora	28	20,0
Toplam	140	100,0

Pozisyon	Frekans	%
Üst Düzey Yönetici	1	0,7
IT Yöneticisi	5	3,6
Yazılım Geliştirme Yöneticisi	6	4,3
Yazılım Geliştirici	71	50,7
Yazılım Mimarı	16	11,4
İş analisti	38	27,1
Danışman / Teknik Uzman	1	0,7
Diğer	2	1,4
Toplam	140	100,0

Çalışma Alanı	Frekans	%
Finans (Bankacılık, Sigortacılık vb.)	99	70,7
Hizmet (Bilişim Hizmetleri, Danışmanlık, AR-GE vb.)	41	29,3
Toplam	140	100,0

Araştırmaya katılanların % 57.1'i 25-31, % 21.4'ü 32-38 yaşları arasındadır.

Araştırmaya katılanların %66.4'ü bay, % 33.6'sı bayandır.

Araştırmaya katılanların % 80'i üniversite, % 20'si lisansüstü/doktora mezunudur.

Araştırmaya katılanların % 50.7'si yazılım geliştirici, % 27.1'i iş analisti, % 11.4'ü yazılım mimarı, % 4.3'ü yazılım geliştirme yöneticisi, % 3.6'sı IT Yöneticisi olarak çalışmaktadır.

Araştırmaya katılanların % 70.7'si finans (bankacılık & sigortacılık) sektöründe, % 29.3'ü Hizmet (Bilişim Hizmetleri, Ar-ge vb.) alanında faaliyet gösteren şirketlerde çalışmaktadır

- Yazılım geliştirme sürecince kullanılan teknik ve araçlara ilişkin sorular

Tablo 3. Araştırmaya katılan çalışanların yazılım geliştirme sürecinde kullandıkları teknik ve araçlara ilişkin frekans dağılım tablosu

İhtiyaç Analizi	Frekans	%
Data akış diyagramları	82	26,9
Veri sözlüğü	33	10,8
Nesne tabanlı analiz	45	14,8
Hızlı prototip	24	7,9
Veri standartlaştırılması	27	8,9
UML	53	17,4
Varlık-Bağıntı diyagramları	17	5,6
Durum değişikliği diyagramı ve analizi	24	7,9
Toplam	305	100,0

Tasarım	Frekans	%
Rapor tasarlayıcılar	39	13,0
Ekran tasarlayıcılar	72	24,1
Yapı çizelgeleri	18	6,0
Karar ağaçları	40	13,4
Nesne tabanlı tasarım	69	23,1
Prototip tasarım	16	5,4
İşlem hacim analizi	11	3,7
Diyalog akış diyagramları	24	8,0
HIPO (hiyerarşil girdi-süreç-çıktı) çizelgeleri	5	1,7
Diğer	5	1,7
Toplam	299	100,0

Kodlama	Frekans	%
4. kuşak diller (Java, C#, Visual Basic, Oracle Forms, ...)	120	90,9
3. kuşak diller (Cobol, Fortran, Basic, Pascal, C, ...)	4	3,0
Diğer	8	6,1
Toplam	132	100,0

Test	Frekans	%
Kara kutu testi (İşlevselliğin sınandığı test)	90	20,3
Beyaz kutu testi (yazılım kodunun sınanması)	79	17,8
Değişikliklerin entegresinden sonra yapılan bağlanım testi	38	8,6
Walkthrough (gözlem)	54	12,2
Yük, zorlanım, performans testi	58	13,1
Güvenlik testi	43	9,7
Birim testi	73	16,5
Diğer	8	1,8
Toplam	443	100,0

Bakım	Frekans	%
Yazılım değişim yönetimi prosedürleri	81	51,6
Konfigürasyon yönetim prosedürleri	61	38,9
Diğer	15	9,6
Toplam	157	100,0

Araştırmaya katılanlar tarafından % 26.9 oranında ihtiyaç analizi safhasında Data Akış Diagramı, % 17.4 oranında UML, % 14.8 oranında Nesne Tabanlı Analiz, % 10.8 oranında Veri Sözlüğü teknikleri kullanılmaktadır.

Araştırmaya katılanlar tarafından % 24.1 oranında tasarım safhasında Ekran Tasarlayıcıları, % 23.1 oranında Nesne Tabanlı Tasarım, % 13.4 oranında Karar Ağaçları, %13.0 oranınd Rapor Tasarlayıcılar kullanılmaktadır.

Araştırmaya katılanlar tarafından % 90.9 oranında kodlama safhasında 4. Kuşak Diller kullanılmaktadır.

Araştırmaya katılanlar tarafından % 20.3 oranında test safhasında Kara Kutu Testi (işlevselliğin sınandığı test), % 17.8 oranında Beyaz Kutu Testi (yazılım kodunun sınanması), % 16.5 oranında Birim Test, % 13.1 oranında Yük, Zorlanım, Performans Testi, % 12.2 oranında Walkthrough (gözlem) teknikleri kullanılmaktadır.

Araştırmaya katılanlar tarafından % 51.6 oranında bakım safhasında Yazılım Değişim Yönetimi Presedürleri, % 38.9 oranında Konfigürasyon Yönetim Prosedürleri kullanılmaktadır.

7.5.3 Normallik Testi

İstatiksel çalışmalarda pek çok analizi uygulayabilmek için verilerin dağılımının normal ya da normale yakın olması gerekmektedir. Verilerin dağılımını görebilmek için genellikle histogram, saplı kutu grafiği, detrended normallik grafiği ve dal yaprak gibi görsel amaçlı grafikler kullanılmaktadır. Bunun yanı sıra Kolmogrov Smirnov ve Shapiro Wilks gibi test yöntemleri de kullanılır.

"Extreme Values" tablosuna göre;

"*Talep iletimi verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Katılmıyorum",

"*Kapsam hazırlığı verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Kararsızım",

"*Analize hazırlık verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Kararsızım",

"*Tasarım verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Kararsızım",

"*Yazılım geliştirme verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Katılmıyorum",

"*Test verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Kararsızım",

"*Kullanıcı kabul testi verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Katılmıyorum",

"*Bakım aşaması verimliliği artırdı*" sorusuna en çok "Kesinlikle Katılıyorum", en az "Kesinlikle Katılmıyorum"

cevapları verilmiştir.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Talep iletimi verimliliği artırdı.	,243	140	,000	,808	140	,000
Kapsam hazırlığı verimliliği artırdı.	,280	140	,000	,782	140	,000
Analize hazırlık verimliliği artırdı.	,285	140	,000	,762	140	,000
Tasarım verimliliği artırdı.	,289	140	,000	,766	140	,000
Yazılım geliştirme verimliliği artırdı.	,263	140	,000	,769	140	,000
Test verimliliği artırdı.	,309	140	,000	,765	140	,000
Kullanıcı kabul test verimliliği artırdı.	,247	140	,000	,803	140	,000
Bakım aşaması verimliliği artırdı.	,245	140	,000	,830	140	,000

a. Lilliefors Significance Correction

Yukarıdaki Kolmogrov Smirnov ve Shapiro Wilks test yöntemleri ile oluşan tabloya göre değişkenlerin tümü anlamlıdır.

7.5.8 Faktör Analizi

Faktör analizi, başlıca amacı aralarında ilişki bulunduğu düşünülen çok sayıdaki değişken arasındaki ilişkilerin anlaşılmasını ve yorumlanmasını kolaylaştırmak için daha az sayıdaki temel boyuta indirgemek veya özetlemek olan bir grup çok değişkenli analiz tekniğine verilen genel bir isimdir. Diğer bir ifade ile faktör analizi, aralarında ilişki bulunan çok sayıda değişkenden oluşan bir veri setine ait temel faktörlerin (ilişkinin yapısının) ortaya çıkarılarak araştırmacı tarafından veri setinde yer alan kavramlar arasındaki ilişkilerin daha kolay anlaşılmasına yardımcı olmaktır. Bir ölçeğin yapısal geçerliliğinin test edilmesinde en yaygın kullanılan yöntemlerden biri faktör analizidir. Bu araştırma kapsamında kullanılan ölçeklere yönelik uygulanan faktör analizi soncunda her bir faktörün toplam açılanan varyansları ve faktör yükleri dikkate alınmıştır. Toplama açıklanan varyans için 0,50 değeri T. Yeniçeri ve E. Erten'e göre iyi bir oran olarak kabul etmelerine rağmen, bu değerin literatürde kabul görmediği tarafımızda yapılan araştırmalarda görülmüştür [Yeniçeri ve ark., 2008, s. 232-247]. Toplam açıklanan varyans için literatürde % 60 ve üzeri iyi bir oran olarak kabul edilmektedir.

Faktör analizi ile ölçme aracını oluşturan soruların kendi aralarında kümelendikleri tespit edilir. Bulunan her faktörün bir teorik yapıyı temsil ettiği düşünülürse faktör analizi, çok sayıda değişkenden az sayıda tanımlanabilen anlamlı yapılara ulaşmayı hedeflemektedir [Büyüköztürk, 2002, s433-s470]

Güvenilir kabul edilen 6 madde için faktör analizi uyguluyoruz.

KMO ve Bartlett's Testi

Örneğin uygunluk ölçümü %80 üzeri çıkmıştır. 0,854 değeri anketin Faktör Analizi için uygunluğunu göstermektedir.

Tablo 5. KMO ve Bartlett's Testi

Kaiser-Meyer-Olkin Measure of Sampling Adequacy.	0,854
Bartlett's Test of Sphericity Approx. Chi-Square	1918,078
df	276
Sig.	0,000

Factor extraction - Total Variance Explained

Total Variance Explained tablosunda ölçeğin kaç faktörden oluştuğu ve bu faktörlerin ölçülmek istenen olguyu ne derecede ölçtükleri görülmektedir.

Total Variance Explained

Component	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared Loadings		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	8,635	35,980	35,980	8,635	35,980	35,980	4,595	19,144	19,144
2	2,388	9,949	45,929	2,388	9,949	45,929	3,996	16,649	35,793
3	1,920	8,000	53,928	1,920	8,000	53,928	2,198	9,157	44,950
4	1,479	6,165	60,093	1,479	6,165	60,093	2,041	8,505	53,455
5	1,154	4,807	64,900	1,154	4,807	64,900	1,971	8,213	61,668
6	1,038	4,325	69,225	1,038	4,325	69,225	1,814	7,556	69,225

Tablo 6'ya göre 1. Faktör (Yazılım Geliştirme) değişimin %19,144'ünü, 2. Faktör (Kapsam Hazırlığı) değişimin % 16,649'unu, 3. Faktör (Test) değişimin % 9,157'sini, 4. Faktör (Talep İletimi) değişimin % 8,505'ini, 5. Faktör (Tasarım) değişimin % 8,213'ünü, 6. Faktör (Analiz) değişimin % 7,556'sını açıklamaktadır.

Modelde yer alan 6 faktör ve 43 soru, modelin %69,225'ini açıklayabilmektedir.

Faktör Rotasyonu (Rotated Component Matrix)

Rotated Component Matrix tablosu incelenirken her bir maddenin hangi faktör altında en yüksek değere sahip olduğuna bakılmalıdır.

Talep iletimi ile ilgili 2 soru , Kapsam hazırlığı ile ilgili 6 soru, Analize Hazırlık ile ilgili 2 soru, Tasarım ile ilgili 2 soru, Yazılım Geliştirme ile ilgili 6 soru, Test aşaması ile ilgili 3 soru ilgili faktöre yüklenmiştir. Tabloda en düşük yükleme oranı 0,542 en yüksek yükleme oranı 0,937 olarak saptanmıştır.

Tablo 7'ye göre Takım çalışmasına en çok yüklemeyi Yazılım geliştirme, sonrasında sırasıyla Kapsam hazırlığı, Test, Talep iletimi, Tasarım ve Analize hazırlık testi soruları oluşturmaktadır.

Tablo 7. Faktör Analizi (Rotated Component Matrix)

	Bileşenler					
	Yazılım Geliştirme	Kapsam Hazırlığı	Test Aşaması	Talep İletimi	Tasarım aşaması	Analize Hazırlık
Yazılımcıya verilen donanım ve yazılım araçları yazılım geliştirme için uygun olmalıdır.	0,844					
Yazılım geliştirme için uygun ortam hazırlanmalıdır.	0,827					
Tasarım aşamasında ortaya konan veriler doğrultusunda yazılımın gerçekleştirildiği aşamadır.	0,771					
Yazılım gerçekleştirme için yazılımcıya iletilen tasarım uygulanabilir olmalıdır.	0,769					
Geliştirilen yazılım, yazılım standartlara uygun olmalıdır.	0,730					
Tasarım, iletilen analize cevap verecek nitelikte olmalıdır.	0,660					
İletilen tasarımın yeterli olmadığı durumlarda analiz ve tasarım tekrar gözden geçirilmelidir.	0,622					
Öncesinde sistem sınırları proje ekibi tarafından belirlenir.		0,758				
Yazılım geliştirme yapılacak ihtiyaçların alt ve üst sınırları belirlenir.		0,718				
Bu safhada proje ekibinin sorumluluğu, yapılacaklar ve yapılmayacaklar belirlenir.		0,705				
Kapsam, tüm proje ekibinin katılımı ile beraber belirlenir.		0,649				
Kapsam belirlendikten sonra kapsam dokümanı proje ekibi ile paylaşılır.		0,640				
Kapsamdan nasıl istekler/problemler geldiği analiz safhasında incelenir ve tasarım yapan kişiye iletilmek üzere analiz edilir.		0,600				0,543
Talep iletiminde bildirilen isteğin tam olarak karşılanıp karşılanmadığı belirlenir.		0,542				
Test sırasında geliştirilen yazılım için performans testi uygulanır.			0,790			
Test sonuçları proje ekibi ile paylaşılır.			0,732			
Test safhasından önce test edilecek durumlar belirlenir.			0,701			
Talep iletiminde, gereksinimler eksik gerçekleştirilirse ne gibi yaptırımlar olacağına dair yaptırımlar sözleşmede bulunmalıdır				0,937		
Talep iletiminde, gereksinimler zamanında gerçekleştirilemezse ne gibi yaptırımlar olacağına dair yaptırımlar sözleşmede bulunmalıdır				0,932		

Tasarım iletilirken yazılı iletişimin yeterli olmadığı durumlarda sözlü iletişim tercih edilmelidir.					0,842	
Analiz iletilirken yazılı iletişimin yeterli olmadığı durumlarda sözlü iletişim tercih edilmelidir.					0,831	
Tasarım aşamasında yazılımcının kullanacağı yazılım araçlarına karar verilir.					0,547	
Test sırasında karşılaşılan problemler için tekrar geliştirme yapılır.						0,649
Analiz safhasında kapsam hazırlığında iletilen problem tanımlanır.						0,612

Açıklanan Toplam Varyans: %69

7.5.7 Cronbach Alfa Güvenilirlik Analizi

Güvenilirlik değeri bir ölçme aracının tekrarlanan ölçümlerde aynı sonucu verme derecesinin göstergesidir.

Reliability Statistics tablosundan faktörün güvenilirliğinin $\alpha = 0{,}916$ yüksek bir değer olduğu görülmektedir. Bununla birlikte alfa katsayısı yalnız başına yeterli olmamaktadır. Sağlıklı bir değerlendirme yapabilmek için faktördeki her bir sorunun bu katsayıya katkısının incelenmesi gerekmektedir.

Reliability Statistics

Cronbach's Alpha	N of Items
,916	8

Item – Total Statistics tablosunun Cronbach's Alpha if Item Delated (Madde Silindiğinde Cronbach Alfa) sütunundaki değerler incelenmelidir. "Bakım aşaması verimliliği artırdı" maddesinin silinmesi halinde ölçeğin güvenilirlik katsayısının $\alpha = 0{,}916$'dan $\alpha = 0{,}919$ 'a yükseleceği görülmektedir.

3 Faktör

Item-Total Statistics

	Scale Mean if Item Deleted	Scale Variance if Item Deleted	Corrected Item-Total Correlation	Cronbach's Alpha if Item Deleted
Talep iletimi verimliliği artırdı.	29,46	24,610	,699	,907
Kapsam hazırlığı verimliliği artırdı.	29,31	25,239	,767	,902
Analize hazırlık verimliliği artırdı.	29,36	24,001	,791	,899
Tasarım verimliliği artırdı.	29,29	25,198	,764	,902
Yazılım geliştirme verimliliği artırdı.	29,38	24,079	,778	,900
Test verimliliği artırdı.	29,30	24,399	,796	,899
Kullanıcı kabul test verimliliği artırdı.	29,52	24,352	,672	,909
Bakım aşaması verimliliği artırdı.	29,68	24,220	,592	,919

"Bakım aşaması verimliliği artırdı" ifadesini anketten çıkartılıp analiz tekrarlandığında güvenilirliğinin α = 0,919'e yükseldiği görülmektedir.

Reliability Statistics

Cronbach's Alpha	N of Items
,919	7

Item-Total Statistics

	Scale Mean if Item Deleted	Scale Variance if Item Deleted	Corrected Item-Total Correlation	Cronbach's Alpha if Item Deleted
Talep iletimi verimliliği artırdı.	25,53	18,366	,657	,917
Kapsam hazırlığı verimliliği artırdı.	25,38	18,553	,783	,905
Analize hazırlık verimliliği artırdı.	25,42	17,397	,819	,900
Tasarım verimliliği artırdı.	25,35	18,330	,813	,902
Yazılım geliştirme verimliliği artırdı.	25,44	17,572	,789	,903
Test verimliliği artırdı.	25,36	17,917	,797	,902
Kullanıcı kabul test verimliliği artırdı.	25,59	18,101	,637	,920

"Kullanıcı kabul testi verimliliği artırdı" ifadesini anketten çıkartılıp analiz tekrarlandığında güvenilirliğinin α = 0,920'ye yükseldiği görülmektedir.

- 6

Reliability Statistics

Cronbach's Alpha	N of Items
,920	6

Item-Total Statistics

	Scale Mean if Item Deleted	Scale Variance if Item Deleted	Corrected Item-Total Correlation	Cronbach's Alpha if Item Deleted
Talep iletimi verimliliği artırdı.	21,44	12,938	,675	,920
Kapsam hazırlığı verimliliği artırdı.	21,29	13,155	,798	,904
Analize hazırlık verimliliği artırdı.	21,33	12,251	,818	,899
Tasarım verimliliği artırdı.	21,26	13,084	,803	,903
Yazılım geliştirme verimliliği artırdı.	21,35	12,244	,817	,900
Test verimliliği artırdı.	21,27	12,947	,745	,910

Kalan 6 maddenin ölçülmek istenen olguyu başarıyla ölçtüğü sonucuna varılmaktadır [Eymen, 2007].

7.5.4 Korrelasyon Analizi

İki değişken arasındaki ilişkinin yönünü ve büyüklüğünü belirlemek için en sık kullanılan istatistik yönetimi, korelasyon (karşılıklı ilişki) analizidir. Doğrusal ilişkileri ortaya çıkaran bu analiz, doğrusal olmayan bir ilişkide anlamlı çıkmayabilir. Korelasyon katsayısı, bir değişkendeki değişimin diğer bir değişkendeki değişimin ne kadarı ile ilişkili olduğunu göstermektedir. Korelasyon karşılıklı ilişkiyi göstermektedir. Sebep-sonuç ilişkisini göstermediği için, bir bağımlı ve bağımsız değişken arasındaki aranabildiği gibi, iki bağımlı yada bağımsız değişken arasında da aranabilmektedir. [Nakip ve ark., 2006, s. 369-385].

Tablo 11. Tüm Değişkenlere Ait Ortalama, Standart Sapma ve Korelasyon Katsayıları

	μ	σ	Talep iletimi verimliliği artırdı	*Kapsam hazırlığı verimliliği artırdı*	*Analize hazırlık verimliliği artırdı*	*Tasarım verimliliği artırdı*	*Yazılım geliştirme verimliliği artırdı*	*Test verimliliği artırdı*	*Kullanıcı kabul testi verimliliği artırdı*	*Bakım aşaması verimliliği artırdı*
Talep iletimi verimliliği artırdı	4,15	0,897								
Kapsam hazırlığı verimliliği artırdı	*4,30*	*0,756*	,697**							
Analize hazırlık verimliliği artırdı	*4,26*	*0,884*	,613**	,712**						
Tasarım verimliliği artırdı	*4,33*	*0,763*	,485**	,739**	,738**					
Yazılım geliştirme verimliliği artırdı	*4,24*	*0,886*	,652**	,635**	,748**	,725**				
Test verimliliği artırdı	*4,31*	*0,832*	,496**	,616**	,662**	,743*	,699**			
Kullanıcı kabul testi verimliliği artırdı	*4,09*	*0,959*	,402**	,497**	,574**	,597**	,465**	,721**		
Bakım aşaması verimliliği artırdı	*3,94*	*1,074*	,607**	,449*	,426**	,333**	,477**	,522**	,565**	

*** P< 0.01 (Tüm değerler için korelasyon 0,01 değeri için anlamlıdır. (Çift yönlü))*

Tablo 11'de modeldeki değişkenlere ait değişkenlere ait ortalama, standart sapma ve korelasyon analizi sonuçları verilmektedir. Değişkenlere ait standart sapma değerleri 0,756 ile 1,074 arasında hesaplanmış olup bu değerler arasındaki varyans (değişkenlik) miktarının, geçerli analiz yapılması için yeterli seviyede olduğunu göstermektedir.

Tablo 11'de verilen korelasyon katsayıları değişkenlerin birbirleri ile olan ilişkilerini göstermektedir. %1 anlamlılık düzeyinde her ilişki için pozitif bir ilişki olduğu söylenebilir. Tablo 11'deki korelasyon katsayıları dikkate alınacak olursa, ikili seviyede incelenen değişkenlerin hepsinin arasında %1 anlamlılık düzeyinde pozitif bir ilişki olduğu görülmektedir.

Tabloya göre, %1 anlamlılık düzeyinde en yüksek ilişki (,748) ile yazılım geliştirme ve analize hazırlık arasındaki iken, en düşük ilişki (,33) ile bakım aşaması ve Tasarım arasındadır.

Tabloya göre, %1 anlamlılık düzeyinde,

"Kapsam hazırlığı verimliliği artırdı" ifadesi (,697) ile "Talep iletimi verimliliği artırdı" arasında pozitif yönde bir ilişki vardır.

"Analize hazırlık verimliliği artırdı" ifadesi (,712) ile "Kapsam hazırlığı verimliliği artırdı" arasında güçlü bir ilişki vardır.

"Tasarım verimliliği artırdı" ifadesi (,739) ile "Kapsam hazırlığı verimliliği artırdı" arasında pozitif yönde güçlü, (,485) ile "Talep iletimi verimliliği artırdı" arasında pozitif yönde ama zayıf bir ilişki vardır.

"Yazılım geliştirme verimliliği artırdı" ifadesi (,748) ile "Analize hazırlık verimliliği artırdı" arasında güçlü bir ilişki vardır.

"Test verimliliği artırdı" ifadesi (,743) ile "Tasarım verimliliği artırdı" arasında pozitif yönde güçlü, (,496) ile "Talep iletimi verimliliği artırdı" arasında pozitif yönde ama zayıf bir ilişki vardır.

"Kullanıcı kabul testi verimliliği artırdı" ifadesi (,721) ile "Test verimliliği artırdı" arasında pozitif yönde güçlü, (,402) ile "Talep iletimi verimliliği artırdı" arasında pozitif yönde ama zayıf bir ilişki vardır.

"Bakım aşaması verimliliği artırdı" ifadesi (,607) ile "Talep iletimi verimliliği artırdı" arasında pozitif yönde, (,333) ile "Tasarım verimliliği artırdı" arasında pozitif yönde ama zayıf bir ilişki vardır.

7.5.6 Regresyon Analizi

Bir bağımsız değişken ve buna bağlı bulunduğu varsayılan bağımlı değişken arasındaki ilişki geçmişteki gözlemler yardımı ile saptanınca, iki değişken arasında matematik bir ilişki bulunup bulunmadığı, var ise bunun ne biçimde olduğu ve daha sonra, bağımsız değişken bilindiği takdirde buna bağlı bulunan değişkenin ne olacağı tahmin olunabilir. Bu regresyon denilen bir analizle yapılır [Hatipoğlu, 1994;188].

Regresyon analizi iki değişken arasında sabep-sonuç ilişkisini araken, sebep-sonuç ilişkisini ortaya çıkarmaz. Sadece iki değişken arasında bir birlikteliğin olduğunu göstermektedir. Regresyon analizi, sadece birlikte bir değişimin olup olmağını göstermektedir [Nakip, 2006; s. 309].

Tablo 12. Talep iletimi, Kapsam hazırlığı, Analize hazırlık, Tasarım, Yazılım geliştirme, Test, Kullanıcı kabul testi ve Bakım aşamalarının proje verimliliğine etkisiyle ilgili regresyon analizi

	R^2	F	Sig
Talep iletimi	0,666	37,559	,000
Kapsam hazırlığı	0,714	46,991	,000
Analize hazırlık	0,698	43,534	,000
Tasarım	0,765	61,537	,000
Yazılım geliştirme	0,728	50,445	,000
Test	0,723	49,270	,000
Kullanıcı kabul testi	0,630	32,120	,000
Bakım aşaması	0,540	22,168	,000

%1 anlamlıdır.

Tablo 12'de Talep iletimi, Kapsam hazırlığı, Analize hazırlık, Tasarım, Yazılım geliştirme, Test, Kullanıcı kabul testi ve Bakım aşamalarının proje verimliliği üzerine etkisi için regresyon analizi sonuçları verilmiştir. Tablodaki F

değeri, modelin anlamlığını gösteren bir değerdir. R^2 değeri ise (belirlilik veya tanımlayıcılık katsayısı) bağımlı değişkendeki değişimin ne kadar bağımsız değişkenler tarafından tanımlanabildiğini gösteren bir ölçüdür. Buna göre modeldeki "Talep iletimi" değişkeni, proje verimliliğinin %67'sini, "Kapsam hazırlığı" değişkeni, proje verimliliğinin %71'ini, "Analize hazırlık" değişkeni, proje verimliliğinin %70'ini, "Tasarım" değişkeni, proje verimliliğinin %76'sını, "Yazılım geliştirme" değişkeni, proje verimliliğinin %73'ünü, "Test" değişkeni, proje verimliliğinin %72'sini, "Kullanıcı kabul testi" değişkeni, proje verimliliğinin %63'ünü, "Bakım aşaması" değişkeni, proje verimliliğinin %54'ünü açıklamaktadır. Modele göre değişkenler $p < 0,01$ düzeyinde anlamlı farka sahip olduğu görülmektedir

Tablo 13. Araştırma Hipotezlerinin Sonuçları

HİPOTEZLER	RED/KABUL
$H_{TÇ}$:Takım çalışması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{Y}:Yazılımcı proje verimliliğini olumlu olarak etkilemektedir.	KABUL
$H_{Tİ}$:Talep iletimi aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{KH}:Kapsam hazırlığı aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{AH}:Analize hazırlık aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{T}:Tasarım aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{YG}:Yazılım geliştirme aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{T}:Test aşaması proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{KKT}:Kullanıcı kabul testi proje verimliliğini olumlu olarak etkilemektedir.	KABUL
H_{B}:Bakım aşaması proje verimliliğini olumlu olarak etkilemektedir. etkilemektedir.	KABUL

7.5.7 Bulguların Özeti

Bu bölümde araştırma kapsamında gerçekleştirilen istatistiksel analizler neticesinde elde edilen sonuçlar özetlenmektedir.

Model 1

Yaptığımız analizlerin sonucunda yazılım geliştirme sürecinin aşamalarından Talep İletimi bağımlı değişken "dependent variable" olarak seçildiğinde, Kapsam Hazırlığı, Yazılım Geliştirme ve Bakım aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Analize Hazırlık, Tasarım, Test ve Kullanıcı Kabul Testi aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Talep iletimi bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta_0 + \beta_2 * KH + \beta_5 * YG + \beta_8 * BA$$

Model 2

Yazılım geliştirme süreci aşamalarından Kapsam Hazırlığı bağımlı değişken "dependent variable" olarak seçildiğinde, Talep İletimi ve Tasarım aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Analize Hazırlık, Yazılım Geliştirme, Test, Kullanıcı Kabul Testi ve Bakım aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Kapsam hazırlığı bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta_2 + \beta_1 * TI + \beta_4 * T$$

Tablo 14 - Yazılım geliştirme aşamalarının proje verimliliğine etkisiyle ilgili bulguların gösterimi

	Model 1			Model 2			Model 3			Model 4		
	Talep İletimi			Kapsam Hazırlığı			Analize Hazırlık			Tasarım		
	Beta	t	Sig.	Beta	t	Sig.	Beta	t	Sig.	Beta	t	Sig.
Talep İletimi	-	-	-	0,430	6,022	0,000	0,113	1,377	0,171	-0,166	-2,328	0,021
Kapsam Hazırlığı	0,502	6,022	0,000	-	-	-	0,217	2,488	0,014	0,409	5,821	0,000
Analize Hazırlık	0,125	1,377	0,171	0,206	2,488	0,014	-	-	-	0,138	1,826	0,070
Tasarım	-0,237	-2,328	0,021	0,499	5,821	0,000	0,178	1,826	0,070	-	-	-
Yazılım Geliştirme	0,348	3,801	0,000	-0,184	-2,091	0,038	0,364	4,228	0,000	0,311	4,089	0,000
Test	-0,090	-0,949	0,344	0,052	0,589	0,557	-0,021	-0,236	0,814	0,237	3,064	0,003
Kullanıcı Kabul Testi	-0,055	-0,667	0,506	-0,060	-0,785	0,434	0,203	2,653	0,009	0,163	2,406	0,018
Bakım	0,320	4,650	0,000	0,028	0,412	0,681	-0,077	-1,090	0,278	-0,173	-2,865	0,005
R^2	0,666			0,714			0,698			0,765		
F	37,559			46,991			43,534			61,537		
Sig.	0,000			0,000			0,000			0,000		

	Model 5			Model 6			Model 7			Model 8		
	Yazılım Geliştirme			Test			Kullanıcı Kabul Testi			Bakım		
	Beta	t	Sig.	Beta	t	Sig.	Beta	t	Sig.	Beta	t	Sig.
Talep İletimi	0,283	3,801	0,000	-0,075	-0,949	0,344	-0,061	-0,667	0,506	0,440	4,650	0,000
Kapsam Hazırlığı	-0,175	-2,091	0,038	0,050	0,589	0,557	-0,077	-0,785	0,434	0,045	0,412	0,681
Analize Hazırlık	0,328	4,228	0,000	-0,020	-0,236	0,814	0,249	2,653	0,009	-0,116	-1,090	0,278
Tasarım	0,361	4,089	0,000	0,280	3,064	0,003	0,257	2,406	0,018	-0,339	-2,865	0,005
Yazılım Geliştirme	-	-	-	0,306	3,654	0,000	-0,326	-3,348	0,001	0,172	1,533	0,128
Test	0,301	3,654	0,000	-	-	-	0,491	5,395	0,000	0,177	1,592	0,114
Kullanıcı Kabul Testi	-0,240	-3,348	0,001	0,368	5,395	0,000	-	-	-	0,427	4,763	0,000
Bakım	0,102	1,533	0,128	0,107	1,592	0,114	0,344	4,763	0,000	-	-	-
R^2	0,728			0,723			0,630			0,540		
F	50,445			49,270			32,120			22,168		
Sig.	0,000			0,000			0,000			0,000		

Model 3

Yazılım geliştirme süreci aşamalarından Analize Hazırlık bağımlı değişken "dependent variable" olarak seçildiğinde, Yazılım Geliştirme aşaması proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Talep İletimi, Kapsam Hazırlığı, Tasarım, Test, Kullanıcı Kabul Testi ve Bakım aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Analize hazırlık bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta 0 + \beta 5 * YG$$

Model 4

Yazılım geliştirme süreci aşamalarından Tasarım bağımlı değişken "dependent variable" olarak seçildiğinde, Kapsam Hazırlığı ve Yazılım Geliştirme aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Talep İletimi, Analize Hazırlık, Test, Kullanıcı Kabul Testi ve Bakım aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Tasarım bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta 0 + \beta 2 * KH + \beta 5 * YG$$

Model 5

Yazılım geliştirme süreci aşamalarından Yazılım Geliştirme bağımlı değişken "dependent variable" olarak seçildiğinde, Talep İletimi, Kapsam Hazırlığı, Tasarım ve Test aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Kapsam Hazırlığı, Kullanıcı Kabul Testi ve Bakım aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Yazılım geliştirme bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta0 + \beta1*TI + \beta3*AH + \beta4*T + \beta6*TA$$

Model 6

Yazılım geliştirme süreci aşamalarından Test bağımlı değişken "dependent variable" olarak seçildiğinde, Yazılım Geliştirme ve Kullanıcı Kabul Testi aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Talep İletimi, Kapsam Hazırlığı, Analize Hazırlık, Tasarım ve Bakım aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Test bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta0 + \beta5*YG + \beta7*KKT$$

Model 7

Yazılım geliştirme süreci aşamalarından Kullanıcı Kabul Testi bağımlı değişken "dependent variable" olarak seçildiğinde, Test ve Bakım aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Talep İletimi, Kapsam Hazırlığı, Analize Hazırlık, Tasarım ve Yazılım Geliştirme aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Kullanıcı Kabul Testi bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta 0 + \beta 6*TA + \beta 8*BA$$

Model 8

Yazılım geliştirme süreci aşamalarından Bakım aşaması bağımlı değişken "dependent variable" olarak seçildiğinde, Talep İletimi ve Kullanıcı Kabul Testi aşamaları proje verimliliğini olumlu olarak etkilemiştir. Yazılım geliştirme sürecinin aşamalarından Kapsam Hazırlığı, Analize Hazırlık, Tasarım, Yazılım Geliştirme ve Test aşamaları proje verimliliğini etkilememektedir.

Tablo 14'teki sonuçlara Bakım Aşaması bağımlı değişken olarak seçildiğinde oluşan çoklu değişken doğrusal denklem :

$$y = \beta_0 + \beta_1*TI + \beta_7*KKT$$

BÖLÜM 8: SONUÇ ve DEĞERLENDİRME

1. Araştırmada, takım çalışması ve yazılımcının yazılım projesi verimliliğine olan etkisinin belirlenmesi amaçlanmıştır. Araştırmaya 140 denek katılmış %66.4'ü bay, % 33.6'sı bayan, katılanların % 80'i üniversite, % 20'si lisansüstü/doktora, % 50.7'si yazılım geliştirici, % 27.1'i iş analisti, % 11.4'ü yazılım mimarı, % 4.3'ü yazılım geliştirme yöneticisi, % 3.6'sı IT yöneticisidir.

2. Araştırmaya katılanların % 26.92'u ihtiyaç analizi safhasında Data Akış Diagramı, % 17.4'i UML, % 14.8'i Nesne Tabanlı Analiz, % 10.8'i Veri Sözlüğü Teknikleri, % 24.1'i tasarım safhasında Ekran Tasarlayıcıları, % 23.1'i Nesne Tabanlı Tasarım, % 13.4'ü Karar Ağaçları, %13.0'ü Rapor Tasarlayıcılar, % 90.9'u kodlama safhasında 4. Kuşak Dilleri, % 20.3'ü test safhasında Kara Kutu Testi (işlevselliğin sınandığı test), % 17.8 Beyaz Kutu Testi (yazılım kodunun sınanması), % 16.5'i Birim Test, % 13.1'i Yük, Zorlanım, Performans Testi, % 12.2'si Walkthrough (gözlem) teknikleri, % 51.6'sı bakım safhasında Yazılım Değişim Yönetimi Presedürleri, % 38.9'u Konfigürasyon Yönetim Prosedürlerini kullanmaktadır.

3. Modelde yer alan 8 faktörden ikisi atıldıktan sonra 6 faktör ve 43 soru, modelin %69,225'ini açıklayabilmektedir. Bu durumda modeli açıklamak için faktör seçiminin isabetli olduğunu göstermektedir.

4. 8 faktörün güvenirlik sonucu 0.916 olmasına rağmen iki faktörün atılmasıyla güvenirlik katsayısı 920 yükselmiştir. Kalan 6 maddenin ölçülmek istenen olguyu başarıyla ölçtüğü sonucuna varılmıştır.

5. Korelasyon analizi modelin değişkenleri arasında pozitif yönde güçlü bir ilişkinin olduğunu göstermektedir.

6. Regresyon analizi sonucunda modeldeki 8 faktörün modeli açıklamada tutarlı olduğu gözlendi ve faktörlerin modeli açıklama gücü 0.540 ile 0.765 arasında değiştiği sonucuna ulaşıldı. Dolayısı ile modelin hipotezleri kabul

edildi. Modelde yer alan her bir faktörün aralarında anlamlı farklılıklara sahip olduğu görüldü.

8.1 Çalışmanın Kısıtları Ve Araştırmacılara Öneriler

Bu çalışma teorik ve pratik açıdan değerlendirildiğinde birçok kısıtla karşı karşıya olduğunu ifade etmek gerekir. Gelecekte bu konu ile ilgili çalışma/lar yapmak isteyen araştırmacı/lara yol göstermesi açısından aşağıdaki kısıtları dikkate almaları yararlı olacaktır.

İlk olarak anket uygulaması ağırlıklı olarak Kocaeli'nin Gebze İlçesinde konumlanmış bilgi teknolojileri kullanan bir firmada yapılmış olması veri toplamada heterojen özelliği sağlamadığı için bir kısıttır. Homojen bir örneklem kitlesinin seçilmesinden dolayı araştırmanın genellenebilir sonuçlar açısından tartışılması muhtemeldir.

İkinci olarak; araştırmada kullanılan verilerin sadece özel sektörden elde edilmesi kamu sektörünün dikkate alınmaması sonuçların kıyaslanması açısından bir kısıt teşkil etmektedir.

Üçüncü olarak da; bu araştırmada bankacılık yazılım projeleri yapan yazılımcılardan verilerin elde edilmesi diğer alanlarda yazılım yapan firmaların ve yazılımcıların fikirlerinin alınmaması tarafımızdan başka bir kısıt olarak değerlendirilmektedir.

Gelecekte bu konu ile ilgili çalışma yapmak isteyen araştırmacıların yukarıdaki kısıtları dikkate almaları durumunda önemli sonuçlara ulaşacakları literatüre ve bilime önemli katkılar sağlayabileceklerini düşünmekteyiz.

8.2 Yöneticiler İçin Öneriler

Bilişim sektöründe yapılan projelerin verimliği çalışanların ve firmanın başarısını ortaya koymaktadır. Yaptığımız çalışma doğrultusunda yöneticilere yardımcı olacak bazı öneriler sunulacaktır.

Bir yazılım projesi hazırlanırken öncelikle projeyle ilgili bilgiler net olarak tanımlanmalıdır. Projenin adı, projenin amacı, projenin hedefleri, projeden istenenler geliştirme süreci başlamadan önce netleştirilmelidir. Proje yöneticisi projenin sahip olduğu kaynaklara (zaman, para, çalışan) en uygun proje yönetim metodunu proje başlamadan önce belirlemelidir.

Araştırmanın bulgularından hareketle yazılım projesi geliştiren şirketlerin yöneticilerine şu önerilerde bulunulmaktadır;

- Yazılım geliştirme sürecinin her aşamasının projenin verimliliğine olumlu etkisinin olduğu görülmüştür. Bu doğrultuda yöneticiler proje verimliliğini artırmak için öncelikle yazılım geliştirme sürecinin her safhasına ve bu süreçlerde çalışan kişilere fark gözetmeksizin gereken önemi vermelidirler.
- Takım çalışmasının proje verimliliğine olan etkisi göz önüne alınarak proje grubundaki kişilere birbirleri ile rahat iletişim kurabilecekleri çalışma ortamlarının yöneticiler tarafından oluşturulması gerekmektedir.

Araştırma sonuçlarına göre çalışanların da proje verimliliğine etkisinin olduğu görülmüştür. Projede bireysel çalışmadan ziyade takım çalışmasının verimliliği etkilediği görüldüğünden, çalışan herkesin projede bulunan diğer kişilerle uyumlu bir çalışma içinde olması gerekmektedir.

Sonuç olarak yazılım geliştirme işi zor ve pahalı bir süreçtir. Yazılım geliştirme aşamalarından herhangi birinde yapılan eksik bir çalışma, projenin ileriki safhalarında maliyet açısından çok daha büyük yüke neden olacaktır. Dolayısıyla geliştirme sürecinin her aşamasının gerektiği şekilde yapılması büyük önem taşımaktadır.

Yazılım geliştirme süreci sonrasında çok az yazılım projesi başarı ile sonuçlanmaktadır. Geliştirme yapan şirketler sektörde ilk saflarda bulunmak isterlerse yukarıda bahsi geçen maddeleri başarı ile gerçekleştirmelidirler.

KAYNAKÇA

Abdel-Hamid, Tarek K. ve Madnick, Stuart E., 1991, Software Project Dynamics: An Integrated Approach, Prentice-Hall, New Jersey.

Anselmo, Donald ve Ledgard, Henry, 2002, Measuring Productivity in the Software Industry

Atan, M., 2004, Bilişim sistemleri ders notları, Gazi Üniversitesi, Ekonometri Bölümü, Ankara.

Baltaş Prof. Dr. A., 2005, Ekip Çalışması ve Liderlik, 2. Basım.

Boddie, John, 1987, Crunch Mode: Building Effective Systems on a Tight Schedule, Yourdon Press, New York.

Boehm, B., 1981, Software Engineering Economics, Prentice-Hall, New Jersey.

Boehm, B., 1989, IEEE tutorial on software risk management, IEEE Computer Society Press, New York.

Britton, C. and Doake, J. 1993, Software system development: a gentle introduction, New York: McGraw-Hill, London.

Brooks, Jr.,F.P., 1986, No Silver Bullet—Essence and Accidents of Software Engineering, Information Processing '86, New York.

Brooks Jr., F. P., 1995, The Mythical Man-Month (20th Anniversary Edition).

Brooks, Jr., F.P., 1995, The Mythical Man-Month: Essays on Software Engineering.

Büyüköztürk, Ş., 2002, Faktör Analizi: Temel Kavramlar ve Ölçek Geliştirmede Kullanımı. *Eğitim Yöntim Dergisi*.

Canbey Özgüler, Yrd. Doç. Dr. Verda, 2005, Verimlilik

Cebeci, Z., 2001, Yazılım testleri, Çukurova Üniversitesi, Adana.

Cohen Susan G. ve Bailey Diane E., 1997, What Makes Teams Work: Group Effectiveness Research From The Shop Floor To The Executive Suite.

Cole, A., 1995, "Runaway Projects-Cause and Effects," Software World (UK).

DeMarco, T. ve Lister, T. R., 1999, Peopleware: Productive Projects and Teams, 2nd Ed., Dorset House Publishing, NY.

Dorfman, M. ve Thayer, R. H., 1996, Software Engineering, IEEE Computer Society Press, Los Alamitos, CA.

Erdoğan Prof. Dr. İ., 1991, İşletmelerde Davranış.

Eren Prof. Dr. E., 2001, Örgütsel Davranış ve Yönetim Psikolojisi.

Eymen, U. Erman, 2007**,** SPSS 15.0 Veri Analiz Yöntemleri

Flowers, S., 1996, Software Failure: Management Failure, Amazing Stories and Cautionary Tales, Wiley.

Glass, R. L., 1998, Software Runaways, Prentice-Hall PTR, Upper Saddle River, NJ.

Gill, S. N., 2005, Factors affecting effective software quality management revisited, *ACM SIGSOFT Software Engineering Notes.*

Gundlach, Erich, 2001**,** "Interpreting Productivity Growth in the New Economy: Some Agnostic Notes", Kiel Institute of World Economics.

Gül, Zuhal, 2006, Yazılım Geliştirme Sürecinin İyileştirilmesi ve Türkiye Uygulamaları.

Hingst, Raymond D., 2006**,** Tuckman's theory of group development in a call centre context.

İnce M., Bedük A. ve Aydoğan E., 2004**,** Selçuk Ünv. S.B.E. Dergisi 2004 sayı 11 , Örgütlerde Takım Çalışmasına Yönelik Etkin Liderlik Nitelikleri.

J. Alberto Espinosa, Robert E. Kraut, Sandra A. Slaughter, Javier F. Lerch, James D. Herbsleb, Audris Mockus, 2002**,** "Shared Mental Models, Familiarity and Coordination: A Multi-Method Study of Distributed Software Teams".

Jones, Capers, 1996, Patterns of Software System Failure and Success, International Thomson Computer Press, Boston, MA.

Jones, Capers, 1998, "Project Management Tools and Software Failures and Successes," CrossTalk, 2003.

K. Todd Stevens, 1998, "The Effects of Roles and Personality Characteristics on Software Development Team Effectiveness".

K. Todd Stevens, Sallie M. Henry "Analyzing Software Teams Using Belbin's Innovative Plant Role".

Karaca E., 1994, Örgütsel Takımlar – Takım Çalışması Ve İş Tatmini Arasındaki İlişkiyi Belirlemeye Yönelik Bir Araştırma, İstanbul Üniversitesi Sosyal Bilimler Enstitüsü, İşletme Fakültesi Davranış Bilimleri Ana Bilim Dalı, Yayımlanmamış Yüksek Lisans Tezi.

Karadağ, L., 2002. Proje yönetimi, BT proje yönetimi ve başarısızlık nedenleri, Bilgi Yönetimi.

,

Katzenbach Jon R. ve Smith Douglas K., Çev. Muallimoğlu N., 1998, Takımların Bilgeliği Yüksek Performanslı Takımlar Oluşturmak.

Kendiroğlu Ç., 2000, Takım Performansını Belirleyen Kişisel ve Kültürel Faktörler, Marmara Üniversitesi Sosyal Bilimler Enstitüsü İngilizce İşletme Anabilim Dalı Örgütsel Davranış Bilim Dalı, Yayımlanmamış Yüksek Lisans Tezi.

Kettunen, P. And Laanti, M., 2005, How to steer an embedded software project: tactics for selecting the software process model, *Information and Software Technology*.

Kossiakoff, A., 2003. Systems engineering: principles and practices, J. Wiley, New York.

Kurnaz Dr. S., Çetin Ö., Prof. Dr. İnce F., 2003, Yazılım Mühendisliğinde Kalite ve Uml, Havacılık ve Uzay Teknolojileri Dergisi,Temmuz 2003 Cilt 1 Sayı 2.

Kurnaz, S., Çetin, Ö. ve İnce, F., 2003. Yazılım mühendisliğinde kalite ve UML, *Havacılık ve Uzay Teknolojileri Dergisi,* **1**, Sayı:2.

Mark D.,Student group approach to teaching using Tuckman modelof group development, Department of Educational Administration and Higher Education, Oklahoma State University,1991.

McConnel, S., 1999. Software engineering principles, *IEEE Software*, **16**, No. 2, March/April 1999.

Nakip, M., Varinli, İ. ve Güllü, K. (2006). Süpermarketlerde Çalışanların ve Türketicilerin Hizmet Kalitesi Beklentilerinin ve Algılamalarının

Karşılaştırılmasına Yöenlik Bir Araştırma. *Atatürk Üniversitesi İktisadi vee İdari Bilimler Fakültesi Dergisi*.

Natarajan, K. V., 2004. Efficient software development, Proceedings of MASPLAS'04, *Mid-Atlantic Student Workshop on Programming Languages and Systems,* Seton Hall University, April 3.

Olson, D. L., 2001. Introduction to information systems project management, Irwin/McGraw-Hill, Boston.

Pauca, V.P., 2003. Software life cycle, Wake Forest University, CSC 331/631, Spring.

Pfleger, S. L., 1991. Software engineering: the production of quality software, Macmillan Pub. Co., New York.

Pressman, R. S., Software Engineering, A Practitioner's Approach, 4th Ed., McGraw-Hill, 1997, ISBN: 0071146032.

Rehber, Devrim, Yazılım Projelerinde Başarısızlık, Ankara

Schach, S.R., 1993. Software engineering, Aksen Associates: Irwin, Homewood, IL.

Scherer, M. And Perelman, Mark, 1992, Entrepreneurship Technological Innovation and Economic Growth Studies in the Schumpeterian Tradition, içinde Richard R. Nelson, US. Technological Leadership: Where Did it Come From and Where Did it Go?.

Schermerhorn John R.,Hunt James G. ve Osborn Richard N., 1997, Organizational Behavior.

Selbes, Tunca, Yazılım Takımlarında Başarı

Sezgin, H.Ö., 2000, Yazılım geliştirme süreci, *Bilişim 2000 Bildirisi*.

Sodhi, J., 1991, Software engineering: methods, management, and CASE tools, PA: Tab Professional and Reference Books, Blue Ridge Summit.

Sommervile, I., 2000, Software Engineering, New York: Addison-Wesley, Harlow, England.

Şen, K., 2005. Gereksinim Mühendisliği, *EMO Bilgisayar Mühendisliği Dergisi*, Ocak.

The Standish Group International, Inc, 2003, "CHAOS: A Recipe for Success".

Thomsett, R., 1980, People Project Management, Yourdon Press, Inc., New York, NY.

W.Lewis, William and Palmade, Vincent and Regout, Baudouin and Webb, Allen P., 2002, "What's Right with the US Economy".

Weinberg, G. M., 1997, Quality Software Management, Vol.1, Systems Thinking.

Weiss Donald H., Çev. Tuskan E. , 1998, Başarılı Ekip Oluşturma.

Yeniçeri, T. ve Erten, E., 2008, Mağaza Sadakat Programlarının Algılanması, Güven, İlişkiyi Sürsürme İsteği ve Mağaza Arasındaki İlişkilerin Yapısal Eşitliği Modeli ile İncelenmesi. *Doğuş Üniveristesi Dergisi*.

Yılmaz, Yrd.Doç.Dr. G., 2007, Yazılım Mühendisliği Notları Bölüm 2.

Yılmaz, Ali Rıza., 2008, Yazılım Geliştirme Takımlarında Adalet Algısı ve Proje Başarısına Etkileri, İstanbul.

Yourdon, Ed, 1997, Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects.

ÖZGEÇMİŞ

Elvan ARSLAN 1984 yılında Manisa'da doğmuştur. 2002 yılında Bornova Anadolu Lisesi'nden mezun olmuştur. 2002-2007 yılları arasında Ege Üniversitesi Bilgisayar Mühendisliği'nde Lisans öğrenimini tamamlamıştır. 2008 tarihinde Gebze Yüksek Teknoloji Enstitüsü İşletme Anabilim Dalı'nda yüksek lisans öğrenimine başlamıştır. 2007 yılından beri özel bir bankanın Bilgi Teknolojileri bölümünde Yazılım Mühendisi olarak çalışmaktadır.

EKLER

Ek 1 - Kişisel Sorularla İlgili Frekans Tabloları

Ek 1.1 – Yaş Frekans Tablosu

Yaş

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	18-24	20	14,3	14,3	14,3
	25-31	80	57,1	57,1	71,4
	32-38	30	21,4	21,4	92,9
	39-45	10	7,1	7,1	100,0
	Total	140	100,0	100,0	

Ek 1.2 – Cinsiyet Frekans Tablosu

Cinsiyet

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Bay	93	66,4	66,4	66,4
	Bayan	47	33,6	33,6	100,0
	Total	140	100,0	100,0	

Ek 1.3 – Eğitim Durumu Frekans Tablosu

Eğitim Durumu

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Üniversite	112	80,0	80,0	80,0
	Lisansüstü / Doktora	28	20,0	20,0	100,0
	Total	140	100,0	100,0	

Ek 1.4 – Pozisyon Frekans Tablosu

Pozisyon

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Üst Düzey Yönetici	1	,7	,7	,7
	IT Yöneticisi	5	3,6	3,6	4,3
	Yazılım Geliştirme Yöneticisi	6	4,3	4,3	8,6
	Yazılım Geliştirici	71	50,7	50,7	59,3
	Yazılım Mimarı	16	11,4	11,4	70,7
	İş Analisti	38	27,1	27,1	97,9
	Danışman / Teknik Uzman	1	,7	,7	98,6
	Diğer	2	1,4	1,4	100,0
	Total	140	100,0	100,0	

Ek 1.5 Çalışma Alanı Frekans Tablosu

Çalışma Alanı

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Finans (Bankacılık, Sigortacılık vb.)	99	70,7	70,7	70,7
	Hizmet (Bilişim Hizmetleri, danışmanlık, AR-GE vb.)	41	29,3	29,3	100,0
	Total	140	100,0	100,0	

Ek 2 - Yazılım Geliştirme Süresince Kullanılan Teknik ve Araçlara İlişkin Sorularla İlgili Frekans Tabloları

Ek 2.1 İhtiyaç Analizi Frekans Tablosu

İhtiyaç Analizi

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Data akış diyagramları	82	26,9	26,9	26,9
	Veri sözlüğü	33	10,8	10,8	37,7
	Nesne tabanlı analiz	45	14,8	14,8	52,5
	Hızlı prototip	24	7,9	7,9	60,3
	Veri standartlaştırılması	27	8,9	8,9	69,2
	UML	53	17,4	17,4	86,6
	Varlık-Bağıntı diyagramları	17	5,6	5,6	92,1
	Durum değişikliği diyagramı ve analizi	24	7,9	7,9	100,0
	Total	305	100,0	100,0	

Ek 2.2 Tasarım Frekans Tablosu

Tasarım

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Rapor tasarlayıcılar	39	13,0	13,0	13,0
	Ekran tasarlayıcılar	72	24,1	24,1	37,1
	Yapı çizelgeleri	18	6,0	6,0	43,1
	Karar ağaçları	40	13,4	13,4	56,5
	Nesne tabanlı tasarım	69	23,1	23,1	79,6
	Prototip tasarım	16	5,4	5,4	84,9
	İşlem hacim analizi	11	3,7	3,7	88,6
	Diyalog akış diyagramları	24	8,0	8,0	96,7
	HIPO (hiyerarşik girdi-süreç-çıktı) çizelgeleri	5	1,7	1,7	98,3
	Diğer...	5	1,7	1,7	100,0
	Total	299	100,0	100,0	

Ek 2.3 Kodlama Frekans Tablosu

Kodlama

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	4. kuşak diller (Java, C#, Visual Basic, Oracle Forms, ...)	120	90,9	90,9	90,9
	3. kuşak diller (Cobol, Fortran, Basic, Pascal, C,...)	4	3,0	3,0	93,9
	Diğer...	8	6,1	6,1	100,0
	Total	132	100,0	100,0	

Ek 2.4 Test Frekans Tablosu

Test

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Kara kutu testi (işlevselliğin sınandığı test)	90	20,3	20,3	20,3
	Beyaz kutu testi (yazılım kodunun sınanması)	79	17,8	17,8	38,1
	Değişikliklerin entegresinden sonra yapılan bağlanım testi	38	8,6	8,6	46,7
	Walkthrough (gözlem)	54	12,2	12,2	58,9
	Yük, zorlanım, performans testi	58	13,1	13,1	72,0
	Güvenlik testi	43	9,7	9,7	81,7
	Birim testi	73	16,5	16,5	98,2
	Diğer	8	1,8	1,8	100,0
	Total	443	100,0	100,0	

Ek 2.5 Bakım Frekans Tablosu

Bakım

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Yazılım değişim yönetimi prosedürleri	81	51,6	51,6	51,6
	Konfigürasyon yönetim prosedürleri	61	38,9	38,9	90,4
	Diğer	15	9,6	9,6	100,0
	Total	157	100,0	100,0	

Ek 3 - Normallik Testi

Extreme Values

			Case Number	Value
Talep iletimi verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	8	5
		4	9	5
		5	15	5[a]
	Lowest	1	47	1
		2	46	1
		3	103	2
		4	95	2
		5	7	2[b]
Kapsam hazırlığı verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	8	5
		4	9	5
		5	14	5[a]
	Lowest	1	117	2
		2	112	2
		3	7	2
		4	135	3
		5	128	3[c]
Analize hazırlık verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	6	5
		4	8	5
		5	9	5[a]
	Lowest	1	138	1
		2	46	1
		3	7	1
		4	135	3
		5	128	3[c]
Tasarım verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	6	5
		4	8	5
		5	9	5[a]
	Lowest	1	7	1
		2	117	2
		3	135	3
		4	128	3
		5	127	3[c]
Yazılım geliştirme verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	6	5
		4	8	5
		5	12	5[a]
	Lowest	1	47	1
		2	46	1
		3	7	1
		4	106	2
		5	83	2
Test verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	6	5
		4	8	5
		5	9	5[a]
	Lowest	1	106	1
		2	91	2
		3	7	2
		4	135	3
		5	128	3[c]
Kullanıcı kabul test verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	6	5
		4	8	5
		5	9	5[a]
	Lowest	1	106	1
		2	103	1
		3	48	1
		4	23	1
		5	117	2[b]
Bakım aşaması verimliliği artırdı.	Highest	1	3	5
		2	5	5
		3	8	5
		4	10	5
		5	11	5[a]
	Lowest	1	106	1
		2	103	1
		3	62	1
		4	47	1
		5	46	1[d]

a. Only a partial list of cases with the value 5 are shown in the table of upper extremes.

b. Only a partial list of cases with the value 2 are shown in the table of lower extremes.

c. Only a partial list of cases with the value 3 are shown in the table of lower extremes.

d. Only a partial list of cases with the value 1 are shown in the table of lower extremes.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Talep iletimi verimliliği artırdı.	,243	140	,000	,808	140	,000
Kapsam hazırlığı verimliliği artırdı.	,280	140	,000	,782	140	,000
Analize hazırlık verimliliği artırdı.	,285	140	,000	,762	140	,000
Tasarım verimliliği artırdı.	,289	140	,000	,766	140	,000
Yazılım geliştirme verimliliği artırdı.	,263	140	,000	,769	140	,000
Test verimliliği artırdı.	,309	140	,000	,765	140	,000
Kullanıcı kabul test verimliliği artırdı.	,247	140	,000	,803	140	,000
Bakım aşaması verimliliği artırdı.	,245	140	,000	,830	140	,000

a. Lilliefors Significance Correction

Ek 4 - Regresyon Analizi ile İlgili Tablolar

Ek 4.1 – Model 1 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,816[a]	,666	,648	,532

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Test verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	74,464	7	10,638	37,559	,000[a]
	Residual	37,386	132	,283		
	Total	111,850	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Test verimliliği artırdı.

b. Dependent Variable: Talep iletimi verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,345	,286		1,205	,230
	Kapsam hazırlığı verimliliği artırdı.	,595	,099	,502	6,022	,000
	Analize hazırlık verimliliği artırdı.	,127	,092	,125	1,377	,171
	Tasarım verimliliği artırdı.	-,279	,120	-,237	-2,328	,021
	Yazılım geliştirme verimliliği artırdı.	,352	,093	,348	3,801	,000
	Test verimliliği artırdı.	-,098	,103	-,090	-,949	,344
	Kullanıcı kabul test verimliliği artırdı.	-,052	,077	-,055	-,667	,506
	Bakım aşaması verimliliği artırdı.	,267	,057	,320	4,650	,000

a. Dependent Variable: Talep iletimi verimliliği artırdı.

Ek 4.2 - Model 2 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,845[a]	,714	,698	,415

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Test verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	56,662	7	8,095	46,991	,000[a]
	Residual	22,738	132	,172		
	Total	79,400	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Test verimliliği artırdı.

b. Dependent Variable: Kapsam hazırlığı verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,481	,220		2,181	,031
	Talep iletimi verimliliği artırdı.	,362	,060	,430	6,022	,000
	Analize hazırlık verimliliği artırdı.	,176	,071	,206	2,488	,014
	Tasarım verimliliği artırdı.	,495	,085	,499	5,821	,000
	Yazılım geliştirme verimliliği artırdı.	-,157	,075	-,184	-2,091	,038
	Test verimliliği artırdı.	,047	,080	,052	,589	,557
	Kullanıcı kabul test verimliliği artırdı.	-,047	,060	-,060	-,785	,434
	Bakım aşaması verimliliği artırdı.	,020	,048	,028	,412	,681

a. Dependent Variable: Kapsam hazırlığı verimliliği artırdı.

Ek 4.3 - Model 3 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,835[a]	,698	,682	,499

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Test verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	75,876	7	10,839	43,534	,000[a]
	Residual	32,866	132	,249		
	Total	108,743	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Test verimliliği artırdı.

b. Dependent Variable: Analize hazırlık verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-,154	,269		-,570	,569
	Talep iletimi verimliliği artırdı.	,112	,081	,113	1,377	,171
	Kapsam hazırlığı verimliliği artırdı.	,254	,102	,217	2,488	,014
	Tasarım verimliliği artırdı.	,207	,113	,178	1,826	,070
	Yazılım geliştirme verimliliği artırdı.	,363	,086	,364	4,228	,000
	Test verimliliği artırdı.	-,023	,097	-,021	-,236	,814
	Kullanıcı kabul test verimliliği artırdı.	,188	,071	,203	2,653	,009
	Bakım aşaması verimliliği artırdı.	-,063	,058	-,077	-1,090	,278

a. Dependent Variable: Analize hazırlık verimliliği artırdı.

Ek 4.4 - Model 4 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,875[a]	,765	,753	,379

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Analize hazırlık verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Talep iletimi verimliliği artırdı., Test verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	61,913	7	8,845	61,537	,000[a]
	Residual	18,972	132	,144		
	Total	80,886	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Analize hazırlık verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Talep iletimi verimliliği artırdı., Test verimliliği artırdı.

b. Dependent Variable: Tasarım verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,511	,200		2,554	,012
	Talep iletimi verimliliği artırdı.	-,141	,061	-,166	-2,328	,021
	Kapsam hazırlığı verimliliği artırdı.	,413	,071	,409	5,821	,000
	Analize hazırlık verimliliği artırdı.	,119	,065	,138	1,826	,070
	Yazılım geliştirme verimliliği artırdı.	,268	,066	,311	4,089	,000
	Test verimliliği artırdı.	,218	,071	,237	3,064	,003
	Kullanıcı kabul test verimliliği artırdı.	,130	,054	,163	2,406	,018
	Bakım aşaması verimliliği artırdı.	-,123	,043	-,173	-2,865	,005

a. Dependent Variable: Tasarım verimliliği artırdı.

Ek 4.5 - Model 5 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,853[a]	,728	,713	,474

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Test verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	79,502	7	11,357	50,445	,000[a]
	Residual	29,719	132	,225		
	Total	109,221	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Test verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

b. Dependent Variable: Yazılım geliştirme verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-,065	,256		-,253	,801
	Talep iletimi verimliliği artırdı.	,280	,074	,283	3,801	,000
	Kapsam hazırlığı verimliliği artırdı.	-,205	,098	-,175	-2,091	,038
	Analize hazırlık verimliliği artırdı.	,328	,078	,328	4,228	,000
	Tasarım verimliliği artırdı.	,420	,103	,361	4,089	,000
	Test verimliliği artırdı.	,320	,088	,301	3,654	,000
	Kullanıcı kabul test verimliliği artırdı.	-,222	,066	-,240	-3,348	,001
	Bakım aşaması verimliliği artırdı.	,084	,055	,102	1,533	,128

a. Dependent Variable: Yazılım geliştirme verimliliği artırdı.

Ek 4.6 - Model 6 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,850[a]	,723	,709	,449

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	69,552	7	9,936	49,270	,000[a]
	Residual	26,620	132	,202		
	Total	96,171	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Kullanıcı kabul test verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

b. Dependent Variable: Test verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,276	,242		1,144	,255
	Talep iletimi verimliliği artırdı.	-,069	,073	-,075	-,949	,344
	Kapsam hazırlığı verimliliği artırdı.	,055	,094	,050	,589	,557
	Analize hazırlık verimliliği artırdı.	-,019	,078	-,020	-,236	,814
	Tasarım verimliliği artırdı.	,305	,100	,280	3,064	,003
	Yazılım geliştirme verimliliği artırdı.	,287	,079	,306	3,654	,000
	Kullanıcı kabul test verimliliği artırdı.	,319	,059	,368	5,395	,000
	Bakım aşaması verimliliği artırdı.	,082	,052	,107	1,592	,114

a. Dependent Variable: Test verimliliği artırdı.

Ek 4.7 - Model 7 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,794[a]	,630	,610	,598

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Analize hazırlık verimliliği artırdı., Test verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	80,520	7	11,503	32,120	,000[a]
	Residual	47,273	132	,358		
	Total	127,793	139			

a. Predictors: (Constant), Bakım aşaması verimliliği artırdı., Tasarım verimliliği artırdı., Talep iletimi verimliliği artırdı., Analize hazırlık verimliliği artırdı., Test verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı.

b. Dependent Variable: Kullanıcı kabul test verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,081	,323		,249	,804
	Talep iletimi verimliliği artırdı.	-,065	,098	-,061	-,667	,506
	Kapsam hazırlığı verimliliği artırdı.	-,098	,125	-,077	-,785	,434
	Analize hazırlık verimliliği artırdı.	,270	,102	,249	2,653	,009
	Tasarım verimliliği artırdı.	,324	,134	,257	2,406	,018
	Yazılım geliştirme verimliliği artırdı.	-,353	,105	-,326	-3,348	,001
	Test verimliliği artırdı.	,566	,105	,491	5,395	,000
	Bakım aşaması verimliliği artırdı.	,307	,064	,344	4,763	,000

a. Dependent Variable: Kullanıcı kabul test verimliliği artırdı.

Ek 4.8 - Model 8 ile İlgili Regresyon Tabloları

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,735[a]	,540	,516	,747

a. Predictors: (Constant), Kullanıcı kabul test verimliliği artırdı., Talep iletimi verimliliği artırdı., Tasarım verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Test verimliliği artırdı.

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	86,684	7	12,383	22,168	,000[a]
	Residual	73,737	132	,559		
	Total	160,421	139			

a. Predictors: (Constant), Kullanıcı kabul test verimliliği artırdı., Talep iletimi verimliliği artırdı., Tasarım verimliliği artırdı., Analize hazırlık verimliliği artırdı., Yazılım geliştirme verimliliği artırdı., Kapsam hazırlığı verimliliği artırdı., Test verimliliği artırdı.

b. Dependent Variable: Bakım aşaması verimliliği artırdı.

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	,313	,403		,777	,439
	Talep iletimi verimliliği artırdı.	,527	,113	,440	4,650	,000
	Kapsam hazırlığı verimliliği artırdı.	,065	,157	,045	,412	,681
	Analize hazırlık verimliliği artırdı.	-,141	,130	-,116	-1,090	,278
	Tasarım verimliliği artırdı.	-,477	,166	-,339	-2,865	,005
	Yazılım geliştirme verimliliği artırdı.	,208	,136	,172	1,533	,128
	Test verimliliği artırdı.	,228	,143	,177	1,592	,114
	Kullanıcı kabul test verimliliği artırdı.	,478	,100	,427	4,763	,000

a. Dependent Variable: Bakım aşaması verimliliği artırdı.

Ek 5 - Anket

Sayın Katılımcı

*Bu anket "**Projelerin Verimliliğinde Takım Çalışması ve Yazılımcının Önemi**"ni ve anlayışını uygulamada karşılaşılan sorunları tespit etmek ve sistemin daha iyi uygulanabilmesi için önerilerinizi almak amacıyla hazırlanmıştır. Siz değerli yöneticilerimizin ve çalışanların vereceği cevaplar sadece bilimsel araştırma amacı ile kullanılacaktır. Hiçbir şekilde birim isimleri ile verilen cevaplar bağdaştırılmayacak ve anketteki bilgiler tek tek açıklanmayacaktır. Sadece toplu sonuçlar değerlendirilmeye tabi tutularak, bilimsel amaçla kullanılacaktır. Uygulamanın içinde bulunan siz değerli yöneticilerimiz ve çalışanların vereceği bilgiler, yazılım ve yönetim bilimine önemli katkılar sağlayacaktır. Yardımlarınız için şimdiden TEŞEKKÜR EDERİZ.*

Doç. Dr. Halim KAZAN (Tez Danışmanı)
halimkazan@ gyte.edu.tr

Elvan İnce
elvanince@gmail.com

A) Kişisel sorular

Yaşınız: …			
Cinsiyet:	☐ Erkek	☐ Bayan	
Eğitim durumu:	☐ Lise	☐ Üniversite	☐ Lisansüstü/doktora
Firmadaki pozisyonunuz: ☐Üst Düzey Yönetici ☐IT Yöneticisi ☐Yazılım Geliştirme Yöneticisi ☐Yazılım Geliştirici ☐ Yazılım Mimarı ☐ İş Analisti ☐Danışman / Teknik Uzman ☐Diğer			
Firmanızın çalışma alanı: ☐Finans (Bankacılık, Sigortacılık vb.) ☐Hizmet (Bilişim hizmetleri, danışmanlık, AR-GE vb.) ☐Endüstri (Beyaz eşya, mobilya, makine-otomotiv vb.) ☐Kamu (Savunma, adalet vb.) ☐Diğer			

B) Yazılım geliştirme süresince kullanılan teknik ve araçlara ilişkin soruları içermektedir. Kullandığınız teknik ve araçlara ilişkin kutucukları işaretleyiniz.

1- İhtiyaç analizi safhasında;
- ☐Data akış diyagramları
- ☐Veri sözlülüğü
- ☐Nesne tabanlı analiz
- ☐Hızlı prototip
- ☐Veri standartlaştırılması
- ☐UML
- ☐Varlık-Bağıntı diyagramları
- ☐Durum değişikliği diyagramı ve analizi

2- -Tasarım safhasında;
- ☐Rapor tasarlayıcılar
- ☐Ekran tasarlayıcılar
- ☐Yapı çizelgeleri
- ☐Karar ağaçları
- ☐Nesne tabanlı tasarım
- ☐Prototip tasarım
- ☐İşlem hacim analizi
- ☐Diyalog akış diyagramları
- ☐HIPO (hiyerarşik girdi-süreç-çıktı) çizelgeleri
- ☐Diğer…

3- Kodlama safhasında;
- ☐4.kuşak diller (Java, C#, Visual Basic, Oracle Forms ...)
- ☐3.kuşak diller (Cobol, Fortran, Basic, Pascal, C,...)
- ☐Diğer…

4- Test safhasında;
- ☐Kara kutu testi (işlevselliğinin sınandığı test)
- ☐Beyaz kutu testi (yazılım kodunun sınanması)
- ☐Değişiklerin entegresinden sonra yapılan bağlanım t.
- ☐Walkthrough (gözlem)
- ☐Yük, zorlanım, performans testi
- ☐Güvenlik testi
- ☐Birim testi
- ☐Diğer

5- Bakım safhasında;
- ☐Yazılım değişim yönetimi prosedürleri
- ☐Konfigürasyon yönetim prosedürleri
- ☐Diğer

C) Yazılım geliştirme süreci ile ilgili sorular

Kesinlikle Katılmıyorum	Katılmıyorum	Kararsızım	Katılıyorum	Kesinlikle Katılıyorum
1	2	3	4	5

- ***TALEP İLETİMİ***

	1	2	3	4	5
İhtiyaçların iletildiği yazılım geliştirme sürecidir.	☐	☐	☐	☐	☐
Fonksiyonel ve fonksiyonel olmayan tüm gereksinimleri kapsar.	☐	☐	☐	☐	☐

İhtiyaçların net olarak iletilmesi proje planının uygulanabilirliğini kolaylaştırır.	☐	☐	☐	☐	☐
Bildirilen ihtiyaçlarla ilgili son gerçekleştirme zamanı belirtilir.	☐	☐	☐	☐	☐
Talep iletiminde, gereksinimler zamanında gerçekleştirilemezse ne gibi yaptırımlar olacağına dair yaptırımlar sözleşmede bulunmalıdır.	☐	☐	☐	☐	☐
Talep iletiminde, gereksinimler eksik gerçekleştirilirse ne gibi yaptırımlar olacağına dair yaptırımlar sözleşmede bulunmalıdır.	☐	☐	☐	☐	☐
Gereksinim belirleme taraflar arasında hazırlanan bir sözleşmeye bağlı olmalıdır.	☐	☐	☐	☐	☐
Yazılım tarafından gerçekleştirilecek gereksinimlerle ilgili, başlangıç ve bitiş tarihleri taraflar arasında hazırlanan sözleşmede belirtilmelidir.	☐	☐	☐	☐	☐

- ***KAPSAM HAZIRLIĞI***

Bildirilen ihtiyacın karşılanıp karşılanamayacağının belirlendiği yazılım geliştirme sürecidir.	☐	☐	☐	☐	☐
Öncesinde sistem sınırları proje ekibi tarafından belirlenir.	☐	☐	☐	☐	☐
Yazılım geliştirme yapılacak ihtiyaçların alt ve üst sınırları belirlenir.	☐	☐	☐	☐	☐
Kapsam, tüm proje ekibinin katılımı ile beraber belirlenir.	☐	☐	☐	☐	☐
Bu safhada proje ekibinin sorumluluğu, yapılacaklar ve yapılmayacaklar belirlenir.	☐	☐	☐	☐	☐
Talep iletiminde bildirilen isteğin tam olarak karşılanıp karşılanmadığı belirlenir.	☐	☐	☐	☐	☐
Kapsam belirlendikten sonra kapsam dokümanı proje ekibi ile paylaşılır.	☐	☐	☐	☐	☐

- ***ANALİZE HAZIRLIK***

Analiz safhasında kapsam hazırlığında iletilen problem tanımlanır.	☐	☐	☐	☐	☐
Kapsamdan nasıl istekler/problemler geldiği analiz safhasında incelenir ve tasarım yapan kişiye iletilmek üzere analiz edilir.	☐	☐	☐	☐	☐
Analiz safhasında yazılımın ne yapacağı / nasıl yapacağı tanımlanır.	☐	☐	☐	☐	☐
Analiz aşamasında ihtiyaç halinde geliştirme ekibinden destek alınır.	☐	☐	☐	☐	☐
Hazırlanan analiz dokümanı ihtiyacın net olarak anlaşılmasını sağlar.	☐	☐	☐	☐	☐
Tasarım aşamasına geçmeden önce hazırlanan analiz dokümanı proje ekibi ile paylaşılır.	☐	☐	☐	☐	☐
Analiz iletilirken yazılı iletişimin yeterli olmadığı durumlarda sözlü iletişim tercih edilmelidir.	☐	☐	☐	☐	☐

- ***TASARIM***

İletilen analize göre bir ön tasarım yapılır.	☐	☐	☐	☐	☐
Bu aşamada uygulanabilir bir tasarım yapılmasına dikkat edilmelidir.	☐	☐	☐	☐	☐
Tasarım aşamasında yazılımın temel yapısı oluşturulur.	☐	☐	☐	☐	☐
Tasarım, iletilen analize cevap verecek nitelikte olmalıdır.	☐	☐	☐	☐	☐
Tasarım, yazılım geliştirme ortamı için uygun olmalıdır.	☐	☐	☐	☐	☐
Tasarım aşamasında yazılımcının kullanacağı yazılım araçlarına karar verilir.	☐	☐	☐	☐	☐
Tasarım aşaması yazılım için genel bir çizgi oluşturmalıdır.	☐	☐	☐	☐	☐
Yazılım geliştirme aşamasına geçmeden önce hazırlanan tasarım dokümanı proje ekibi ile paylaşılır.	☐	☐	☐	☐	☐
Tasarım iletilirken yazılı iletişimin yeterli olmadığı durumlarda sözlü iletişim tercih edilmelidir.	☐	☐	☐	☐	☐

- ***YAZILIM GELİŞTİRME***

Tasarım aşamasında ortaya konan veriler doğrultusunda yazılımın gerçekleştirildiği aşamadır.	☐	☐	☐	☐	☐
Yazılım gerçekleştirme için yazılımcıya iletilen tasarım uygulanabilir olmalıdır.	☐	☐	☐	☐	☐
İletilen tasarımın yeterli olmadığı durumlarda analiz ve tasarım tekrar gözden geçirilmelidir.	☐	☐	☐	☐	☐
Yazılımcıya verilen donanım ve yazılım araçları yazılım geliştirme için uygun olmalıdır.	☐	☐	☐	☐	☐
Yazılım geliştirme için uygun ortam hazırlanmalıdır.	☐	☐	☐	☐	☐
Geliştirilen yazılım, yazılım standartlara uygun olmalıdır.	☐	☐	☐	☐	☐

- ***TEST***

Yazılım geliştirme aşamasından sonra gerçekleştirilen doğrulama ve sınama aşamasıdır.	☐	☐	☐	☐	☐

Test safhasından önce test edilecek durumlar belirlenir.	☐	☐	☐	☐	☐
Test sırasında iletilen problemin/isteğin karşılanıp karşılanmadığı belirlenir.	☐	☐	☐	☐	☐
Test sırasında geliştirilen yazılım için performans testi uygulanır.	☐	☐	☐	☐	☐
Test sırasında karşılaşılan problemler için tekrar geliştirme yapılır.	☐	☐	☐	☐	☐
Test sonuçları proje ekibi ile paylaşılır.	☐	☐	☐	☐	☐

- ***KULLANICI KABUL TESTİ***

Geliştirilen yazılımın isteği karşılayıp karşılamadığının belirlendiği aşamadır.	☐	☐	☐	☐	☐
İsteğin karşılanmadığı belirlenirse tüm süreç gözden geçirilir.	☐	☐	☐	☐	☐
Kullanıcı kabul testinde işlevsellik, performans gözden geçirilir.	☐	☐	☐	☐	☐
Kullanıcının ilettiği istekle, geliştirilen karşılaştırılır ve ne kadarının karşılandığı belirlenir.	☐	☐	☐	☐	☐
Kullanıcı kabul testini, konu hakkında detaylı bilgiye sahip kişilerin yapması beklenir.	☐	☐	☐	☐	☐
Kullanıcı kabul testinde, herhangi bir sistemsel hata ile karşılaşılmaması beklenir.	☐	☐	☐	☐	☐
Gerçekleştirilen, müşterinin isteklerini karşılıyorsa yazılım geliştirme süreci başarılı bir şekilde tamamlanır.	☐	☐	☐	☐	☐

- ***BAKIM***

Yazılım teslim edildikten sonra gözlem yapılan aşamadır	☐	☐	☐	☐	☐
Yazılımla ilgili herhangi bir eklentinin talep edildiği süreçtir.	☐	☐	☐	☐	☐
Yazılım kullanılmaya başladıktan sonra, yazılımı kullanacak kişilere eğitim verilir.	☐	☐	☐	☐	☐
Bakım süreci yazılım yaşadığı sürece devam eden bir süreçtir.	☐	☐	☐	☐	☐
Bakım sürecinde verilen desteğin sınırı hazırlanan sözleşme ile belirlenir.	☐	☐	☐	☐	☐

Talep iletimi verimliliği artırdı.	☐	☐	☐	☐	☐
Kapsam hazırlığı verimliliği artırdı.	☐	☐	☐	☐	☐
Analize hazırlık verimliliği artırdı.	☐	☐	☐	☐	☐
Tasarım verimliliği artırdı.	☐	☐	☐	☐	☐
Yazılım geliştirme verimliliği artırdı.	☐	☐	☐	☐	☐
Test verimliliği artırdı.	☐	☐	☐	☐	☐
Kullanıcı kabul test verimliliği artırdı.	☐	☐	☐	☐	☐
Bakım aşaması verimliliği artırdı.	☐	☐	☐	☐	☐