İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM GELİŞTİRME SÜRECİNİN İYİLEŞTİRİLMESİ VE TÜRKİYE UYGULAMALARI

YÜKSEK LİSANS TEZİ

Müh. Zuhal GÜL

Anabilim Dalı : İŞLETME MÜHENDİSLİĞİ

Programı : İŞLETME MÜHENDİSLİĞİ

HAZİRAN 2006

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM GELİŞTİRME SÜRECİNİN İYİLEŞTİRİLMESİ VE TÜRKİYE UYGULAMALARI

YÜKSEK LİSANS TEZİ

Müh. Zuhal GÜL
(507031041)

Tezin Enstitüye Verildiği Tarih : 3 Mayıs 2006
Tezin Savunulduğu Tarih : 9 Haziran 2006

Tez Danışmanı : **Öğr.Gör.Dr. Halefşan SÜMEN**
Diğer Jüri Üyeleri **Yrd.Doç.Dr. Ferhan ÇEBİ (İ.T.Ü.)**
Yrd.Doç.Dr. Mehmet Mutlu YENİSEY (İ.T.Ü.)

HAZİRAN 2006

ÖNSÖZ

Günümüzde, yazılımın insanoğlunun hayatında çok önemli bir yeri vardır. Ancak, yazılım geliştirme işi zor bir süreçtir. Değişen talep ve teknoloji karşısında kaliteli, müşterinin isteğine tam cevap verebilen, zamanında ve planlanan bütçede tamamlanabilen yazılımlara ihtiyaç duyuldukça bu sürecin önemi daha da artmaktadır. Bu nedenle yazılım geliştirme süreci bir süreç olarak ciddi bir şekilde ele alınmalıdır. Bu çalışmada da, yazılım geliştirme süreci incelenmeye, Türkiye'de yazılım geliştiren şirketlerin süreç çalışmaları ve süreç iyileştirme aktivitelerinin durumu hakkında bilgi edinilmeye çalışılmıştır.

Çalışma boyunca bana destek veren değerli hocam Sayın Dr. Halefşan SÜMEN'e, aileme, arkadaşlarıma teşekkürlerimi sunarım.

MAYIS 2006 **Zuhal GÜL**

İÇİNDEKİLER

KISALTMALAR

CASE	:Computer Aided Software Engineering
UML	:Unified Modelling Language
RUP	:Rational Unified Process.
CMM	:Capability Maturity Model
CMMI	:Integrated Capability Maturity Model
ISO9000	:International Standard Organization 9000
SPICE	:Software Process Improvement and Capability dEtermination
SEP	:Software Engineering Practices
SPI	:Software Process Improvement
ANSI	:American National Standard Institute
SEI	:Software Engineering Institue
OMT	:Object Modelling Technology
OOSE	:Object Oriented Software Engineering
RAD	:Rapid Application Development
IT	:Information Technology
PMI	:Project Management Institute
US DoD	:United States - Department of Defense
NYP	:Nesne Yönelimli Programlama
FDD	:Feature Driven Development

TABLO LİSTESİ

ŞEKİL LİSTESİ

YAZILIM GELİŞTİRME SÜRECİNİN İYİLEŞTİRİLMESİ VE TÜRKİYE UYGULAMALARI

ÖZET

Günümüzde yazılım sistemleri üretim endüstrisi, eğitim, sağlık, ulaşım, finans, elektrikli aletler, eğlence…gibi ve daha birçok farklı alanda karşımıza çıkmaktadır. Hatta, sektörlerin çoğunda gerekli yazılımlar olmadan organizasyonun yaşamını devam ettirmesi mümkün gözükmemektedir. Bu yüzden, artan rekabet, gelişen teknoloji ve yazılım kuruluşlarının artan kabiliyetlerinin de etkisiyle gelişmiş yazılım sistemlerine olan ihtiyaç her geçen gün artmaktadır. Değişen teknoloji ve talepler, yazılım sistemlerini gün geçtikçe daha da karmaşık hale sokmaktadırlar.Yazılım sistemlerinin gerçekleştirilmesi, ancak bu karmaşık projelerin başarıyla tamamlanmasına bağlıdır.

Yapılan araştırmalar, çoğu projenin hatalı, güvenilirliği az, müşteriyi tatmin etmeyen, zamanında teslim edilmeyen ve umulandan bir kaç kat daha fazla pahalıya mal olmuş bir yazılım ortaya koyduğunu göstermektedir. Peki yazılım projelerinin başarısızlığının sebebi nedir? Projelerin başarısı için CASE araçları,…gibi birçok teknik ve araç geliştirilmiştir, ancak başarısızlığın en temel nedenlerinden birinin, yazılım proje yönetim süreçlerinin içinde yattığının farkına varılmaya başlanmıştır. Görülmektedir ki, yazılım geliştirme işlemlerini destekleyen süreçlerin yokluğunda, projelerin başarıya ulaşma ihtimali çok düşüktür.

Bu çalışmada, yazılım geliştirme süreci temel yönleriyle incelenmeye çalışılmıştır. Bu bağlamda, yazılım geliştirme metotları, modelleri, teknik ve araçları, yazılım geliştirme sürecindeki temel aktiviteler, proje yönetim süreçleri (proje yönetim fonksiyonları, insan yönetimi, risk yönetimi, konfigürasyon yönetimi, kalite yönetimi, ölçüm yönetimi), yazılım geliştirme sürecinin iyileştirilmesi ve modelleri, proje başarısızlık sebepleri, başarı faktörleri ele alınmıştır. Son aşamada ise Türkiye'de yazılım geliştiren şirketlere yazılım geliştirme süreçlerini anlamaya ilişkin bir anket sunulmuş ve elde edilen bulgular değerlendirilmiş, aynı zamanda eksik bulunan, iyileştirilmesi gereken noktalar saptanmıştır.

SOFTWARE PROCESS IMPROVEMENT AND IMPLEMENTATIONS IN TURKEY

SUMMARY

Software systems are now in too many different fields such as manufacturing industry, education, health, communication, finance, electrical equipments and entertainment. Even, it is noticed that most of the sectors cannot go on living without required software systems. So, and the result of increased competition, developed technology and increased capability of software organizations, requirements of developed software systems are increasing from day to day. Changes on technology and demands make software systems more complex. Implementations of these software systems are dependent on achieving these complex projects.

Researches show that many software projects produce software with faults, less reliable, low degree of user satisfaction, not completed on schedule and within over estimated budget. So, what are the reasons of failure of software projects? For success, many techniques, tools have been developed such as CASE tools, but now it has been noticed that the main reason of this failure lies in a process of project management. Without processes that support software development, the degree of probability of succeeding in these projects is too low.

In this study, software development process was observed with main aspects. On this way, software development methods, models, techniques and tools, main activities in software development process, project management process (functions of project management, people management, risk management, configuration management, quality management, measurement management), software process improvement and models, reasons of failure of projects and success factors were observed. At last to gain insight to understand software development processes, a questionnaire was sent to a number of organizations in Turkey and findings were evaluated. At this time also, rooms for improvement were found.

1. GİRİŞ

1.1. Çalışmanın Amacı ve Yöntemi

Fiziksel kısıtlamalar olmamasına rağmen yazılım, karmaşık ve anlaması 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 yazılım geliştirme yollarına ihtiyaç duyulmuştur. Bu çalışmada, bu karmaşık yapının geliştirme sürecinin incelenmesi amaçlanmıştır.

Araştırmanın 2. ve 3. bölümlerinde yazılımın zorlukları, sistem içindeli rolü ve yazılım projelerinin başarısı için geliştirilen metotlar, teknik ve araçlar incelenmiştir. 4. bölümde, yazılım geliştirme süreci modelleri (disiplinsiz, şelale, evrimsel, çevik modeller) birbirleriyle karşılaştırılarak incelenmiştir. 5. 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.

6. bölümde proje yönetimi üzerinde durulmuştur. Bu bölümde; proje yönetim fonksiyonları, insan yönetimi, risk yönetimi, konfigürasyon yönetimi, kalite yönetimi ve ölçüm yönetimi süreçlerinin uygulanma biçimleri ve başarılı bir yazılım projesi için önemleri anlatılmaya çalışılmıştır.

7. ve 8. bölümlerde yazılım süreç iyileştirme süreci, iyileştirme modelleri (CMM,CMMI, ISO9001, SPICE), süreç iyileştirmenin kalite ve verimliliğe etkisi incelenmiştir. 9 ve 10. bölümlerde de proje başarısızlıkları ve yazılım geliştirmedeki yeni gelişmeler hakkında bilgi verilmiştir.

Çalışmanın uygulama bölümünde ise Türkiye'de yazılım geliştiren şirketlerin yazılım geliştirme süreçlerini anlamaya ilişkin bir anket çalışması yapılmıştır. Sonuç bölümünde de, anket sonuçlarından elde edilen bulgular değerlendirilmiş, geliştirilmesi gereken alanlar saptanmıştır.

2. 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 diğ., 2003).

2.1. 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).

2.2. 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

1. 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.

2. 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, finanssal yönetim sistemleri,.. gibi.

3. 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).

2.3. 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).

3. YAZILIM GELİŞTİRME

3.1. Yazılım Geliştirmede Genel Durum

Yazılım geliştirme ilk safhalar:

- Programlar küçük,
- Program tek bir kişi tarafından yazılmakta,
- Programcı programın kullanıcısı,
- Yazılım geliştirme bir sanat olarak görülmekte.

Şimdiki durum:

- Programlar büyük,

  UNIX işletim sistemi 3.7 milyon satır koddan oluşmaktadır.

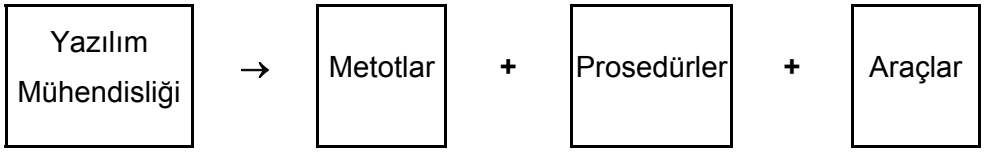
  NASA uzay mekikleri yazılımı 40 milyon satır koddan oluşmaktadır.
- Programlar bir takım tarafından birkaç yıl birlikte işbirliği yaparak geliştirilmekte,
- Programcılar geliştirdikleri sistemin kullanıcısı değiller, uygulamaya ait uzman bilgileri yoktur,
- Yazılım geliştirme mühendislik disiplini olarak görülmektedir.

3.2. Basit Tanımlar

Yazılım Mühendisliği: Mühendislik, bilimsel ve matematiksel prensiplerin, metotların ve araçların ekonomik şekilde kaliteli yazılım geliştirilmesi için kontrollü bir şekilde uygulandığı bir bilim dalıdır. Yazılım mühendisliği ilk kez 1968 yılında NATO'nun bilim komitesi sponsorluğundaki bir konferansta Fritz Bauer tarafından tanımlanmıştır. Bauer, bu terimi sistem mühendisliği yaklaşımlarının yazılım geliştirme ve bakımına uygulanması anlamında kullanmıştır.

Yazılım mühendisliğinin önemi, onun yazılım yaşam süresince yazılım geliştirmeye, gerçekleştirmeye, bakımına sistematik bir yaklaşım getirmesidir. Yazılım mühendisliği, kalite ve maliyet faydalı yazılımlar için kullanılan yapısal metotlar

(planlama, analiz, tasarım, kodlama, test entegre, bakım), prosedürler (proje takibi, kontrolü, kalite güvence, konfigürasyon yönetimi) ve araçları içerir (Sodhi, 1991).



Şekil 3.1 : Yazılım Mühendisliği Yapısı

Yazılım mühendisliği genç bir disiplindir. İlk olarak 1968'de, daha sonradan "yazılım krizi" olarak adlandırılacak bir konunun tartışıldığı bir konferansta ele alınmıştır. Kriz o zamanlar için güçlü, 3. jenerasyon bilgisayar donanımının devreye girmesiyle baş göstermiştir. Onların gücü, o güne kadar fark edilmeyen bilgisayar uygulamalarının girişimini sağlamıştır. İstenilen yazılımlar, önceliklere göre daha karmaşık ve büyük boyutlu olmuştur.

İlk deneyimler, bu sistemleri oluşturmada, formal olmayan yazılım geliştirmenin yeteri kadar iyi olmadığını göstermiştir. Çoğu proje bazen yıllarca gecikmiş, maliyeti planlananı aşmış ve güvenilir olmayan, bakımı zor projeler olarak ortaya çıkmışlardır. Yazılım geliştirmede kriz dönemine girilmiştir. Donanım fiyatları düşmekte, yazılım maliyeti artmaktadır. Büyük yazılım sistemlerindeki doğal karmaşıklığı kontrol etmek için yeni teknik ve araçlara ihtiyaç vardır.

Bu teknikler, yazılım mühendisliğinin bir parçası olmuşlardır ve şimdi evrensel olmasa da geniş bir kulanım alanına sahiptirler. Fakat yine de müşterinin ihtiyacını tam olarak karşılayan, planlanan bütçe ve süre içerisinde kalınarak gerçekleştirilen yazılım üretmede birkaç problem vardır. Bir çok yazılım projesi problemlidir ve bu da birkaç eleştirmeni yazılım mühendisliğinin "kronik sorun" durumunda olduğu önerisine sürüklemiştir.

Yazılım üretme yeteneği arttıkça, istenilen yazılımın karmaşıklığı da artmaktadır. Gelişen yeni teknolojiler yazılım mühendisliğinde yeni taleplere yol açmaktadır. Bu nedenden ve birçok şirketin yazılım mühendisliği tekniklerini kullanmamasından ötürü hala problemler vardır. 1968'den bu yana yazılım mühendisliğinde iyileşmeler, ilerlemeler kaydedilmiştir. Yeni geliştirilen notasyonlar, araçlar, kompleks sistemlerde gereken eforu azaltmaktadır. Fakat daha iyileştirilmesi gereken bir çok alan vardır.

Diğer tanımlara bakacak olursak;

Müşteri: Ürün gelişimini isteyen kişi veya organizasyonlar.

Geliştirici: İhtiyaçlara göre ürünü geliştirecek organizasyon üyeleri.

Kullanıcı: Yazılım tamamlandığında ürünü kullananlar, faydalananlar.

Müşteri ve geliştirici aynı organizasyonun birimleri olabilir. Örneğin; müşteri bir sigorta şirketinin baş uzmanı olabilir, geliştiriciler ise şirketin yönetim bilişim sistemlerine ait bir takım olabilir. Bu "dahili (internal)" yazılım geliştirme olarak adlandırılır. Diğer yandan " sözleşmeli (contract)" yazılım geliştirmede ise müşteri ve geliştiriciler iki farklı organizasyonlardır (Schach, 1993).

Genelde, ürünleri satıcılardan almak daha etkilidir. Fakat satıcıların ürüne kattıkları her zaman optimal değildir. Satıcılar kullanıcılara her zaman sistemlerinin kullanıcı istekleriyle tamamen örtüştüğünü anlatırlar, ayrıca ürüne değerinden fazla şey yüklerler. Gerçekte bu böyle değildir.

Ürünü kendimiz üretmemiz ise daha fazla risk ve zamanla uğraşmayı gerektirir. İstenilen uzmanlık yoksa, dışarıya yaptırmak daha mantıklıdır. Çoğunlukla, dış kaynak kullanımı daha hızlı ve daha ucuzdur. Problem çoğunlukla varolan sistem ve istenilen hizmetin birbiriyle uyuşmamasıdır.

Avrupa'da teklif verenler arasında kazanan taraf, onlardan istenilenin ne olduğunu bilerek akıllıca hesaplanmış ortalama fiyatlı tekliflerdir. Fiyatın yanında diğer unsurlar; maliyetinin bütçeye uygun olması, teklifi verenin gerçekten projeyi yapabilme yeteneğinin olması, benzer projelerdeki durumları, kaliteli işlerdeki pozisyonlarıdır (Olson, 2001).

3.3. Metodoloji (Yöntembilim)

Metodoloji, yazılım geliştirme sürecinin aşamalarını çeşitli basamaklara böler. Her adımda ne yapılması, hangi sırada yapılması gerektiğini, hangi dokümanların oluşturulması gerektiğini tarif eder. Gerçekte detaylı bir plan sunar (Britton ve Doake, 1993). Kısaca, sistem geliştiricilerinin yazılım ve bilgi sistemi oluşturabilmesi için gerekli olan aktiviteleri, modelleri, rotaları, süreç rehberlerini ve otomatikleştirilmiş araçları içerir (Sommervile, 2000).

Birçok yazılım sistemi süreç üzerinde çok az düşünülerek kurulmaktadır. Yazılım takımı yeterli bilgiyi toplar toplamaz kodlamaya geçmektedir. Sürece olan bu dikkatsizlik de projeyi öldürmektedir. Sürecin ileriki safhalarında, kodlamanın büyük parçası müşteri gereksinimlerini karşılamadığı için programın büyük bir bölümü

tekrar yazılmak zorunda kalınmaktadır. Süreç üzerine düşünülmeden yapılan projelerin tamamlanma şansı çok azdır. Proje tamamlansa bile, final önemli tekrar kod yazmalarla ve bütçe aşımları ile gerçekleşmektedir (Dorsey, 2005).

Geliştiriciler, gelişim süreci için anlaşılmış bir yapı sunan bir çerçeve (metodoloji) içinde çalışmak durumundadırlar. Bu çerçeve, geliştiricilere iş başlamadan önce belirtilmiş, dokümante edilmiş standart bir yaklaşım sunmaktadır. Aynı çerçeve (metodoloji) genellikle tüm projeler için kullanılabilecektir. Fakat yine de farklı uygulamalar, farklı yaklaşımlar gerektirebilmektedir. Bu yaklaşım, geliştirilen sistem, geliştiriciler, kullanıcılar, kullanılan teknikler için uygun olmalıdır. Geliştiriciler, tek bir metodolojinin karşılaştıkları çok farklı durumlarla uğraşmak hususunda yeterli olmayacağının farkındadırlar. Açıktır ki, çok farklı yazılım uygulamalarının olması da, yazılım sistemi geliştirmek için tek bir ideal yolun olmayacağını göstermektedir (Britton ve Doake, 1993).

3.3.1. Niçin İhtiyaç Var?

Metodolojiler, yazılım geliştiricilerin deneyimlerine göre edinilen bilgileri bir bütün halde toplamakta, somutlaştırmaktadır. Örneğin, ihtiyaç analizinin ne kadar önemli olduğu ve bu aşamaya önem verilmezse daha sonraki maliyetlerin kötü etkileneceği bilindiğinden bu aşamada ihtiyaçların doğru toplanması için birçok görev yüklenmiştir.

Metodolojiler, deneyimsiz geliştiricilere takip etmeleri gereken bir çerçeve sunmaktadır. Buna örnek olarak, herhangi bir aşamada gereken doküman ve çıktıların ne olması gerektiği gibi bir örneği verebiliriz.

Yazılım geliştirme, özellikle karmaşık ve büyük bir sistem geliştirmek, birçok görevin yapılmak zorunda olunduğu uzun ve karmaşık bir süreçtir. Bu durumda, geliştiricilerin daha koordineli çalışması gerekmektedir. Metodoloji, süreci küçük görevlere ayırarak, ne yapılması gerektiğini belirterek proje yönetimine, planlamaya, çizelgelere, ilerlemeyi izlemeye yardım etmektedir (Britton ve Doake, 1993).

3.4. Farklı Metodolojiler

Yourdan (1978) ve JSD (Jackson System Development, 1983) ilk olarak 1970'lerde geliştirilmiştir. Bu metotlar, sistemin temel fonksiyonel bileşenlerini belirlemeye çalışmışlardır ve fonksiyon-tabanlı metotlar hala geniş bir kullanım alanına sahiptir. 1980 ve 90'larda fonksiyon tabanlı metotlara nesne tabanlı metotlar (Booch 1994,

Rumbough 1991) eklenmiştir. Bu farklı yaklaşımlar daha sonra tek bir UML (Unified Modelling Langauage) çatısı altında toplanmıştır (Sommervile, 2000).

Farklı sistemler için farklı metodolojiler geliştirilmiştir. Farklı metodolojiler farklı bölgelere vurgu yapmaktadır. Bazıları sistem boyunca data akışına odaklanırken, bazıları data mimarisine, ilişkisine odaklanmaktadır. Hepsi farklı araç kullanmaktadır. Metodolojiler, yeni teknolojileri ve fikirleri uygulamak için devamlı değişmektedirler (Britton ve Doake, 1993).

İdeal metot yoktur, çünkü çok fazla metot vardır (Sommervile, 2000). Fakat bir metodolojinin seçilmesi, hiç olmamasından iyidir. Önemli olan birbirini tutan, odaklanılmış bir yolda, süreçleri dikkatlice düşünerek ilerlemektir.

Şunu unutmamak gerekir ki hiçbir araç, metodoloji projenin başarısını garantilemez. Proje daha pahalı veya daha ucuz olabilir, fakat kullanılan metodoloji ve araçlara göre başarılı veya başarısız olmaz (Dorsey, 2005).

Şunu da eklemek gerekir ki; sabit bir metodolojiyi uygun olmayan bir projeye zorlamaktansa (tek bir metodoloji esnekliği de götürebilmektedir), geliştiriciler tekniklerin, araçların ve süreç yaklaşımlarının olduğu bir araç-kutusuna sahip olmanın daha uygun bir araç olacağına inanmaktadırlar. Bu şekilde, bu kutudan her yeni uygulama için uygun tekniği, aracı, model yaklaşımını seçebileceklerdir. Teknik seçim, problemin doğası ile belirlenebilmekte, metodoloji ile önceden belirlenememektedir (Britton ve Doake, 1993).

3.4.1. Yapısal Yöntemler

Yapısal teknikler, teorik temelli değildir. Örneğin, data akış diyagramında problemin anlamını açıklamak için tek doğru bir yol yoktur. Ayrıca bu diyagramların mantıksal olarak birbirini tuttukları ve tam olduklarını kanıtlamanın yolu yoktur.

Yapısal teknikler ve doğal diller yazılım geliştiriciye engin faydalar sağlar, fakat kesinliğin, tutarlılığın esas olduğu yelerde sıklıkla yetersiz kalmaktadırlar (Britton ve Doake, 1993).

Yapısal yöntemlerle modelleme teknikleri:

3.4.1.1. Data Akış Diyagramları

Sistem boyunca verinin akışını modellerler. Diğer modelleme teknikleri gibi uğraşılması gereken detayları sınırlandırmak için ayrıştırma (bir parçaya yoğunlaşmak için) ve soyutlama kullanır. Faydalıdır, çünkü sistemin ne yapacağını

anlaşılabilir bir yolla, grafikle göstermektedir. Fakat birçok soruyu kasıtlı olarak cevapsız bırakmakta, sistemin birçok yüzü diğer tekniklerle modellenmektedir.

En çok analiz aşamasında, kullanılan ve istenilen sistemi tarif etmek için ayrıca fizibilite çalışma esnasında modelleme yapmak için ve ayrıca istenilen sistemi modellerken kısıtları göstermek için kullanılmaktadır. Data akış diyagramlarını çizme aşamasında ve bunları kullanıcıyla tartışma aşamasında tutarsızlıklar, gözden kaçanlar, anlaşmazlıklar gözlenebilmekte ve doğrulanabilmektedir .

3.4.1.2. Data Sözlüğü

Datalar için merkez bellektir. Geliştiriciye veri akışını ve belleğini tanımlamasına basit isimlerle, data akış diyagramlarını okunabilir tutarak izin verir. Birçok iletişim problemini çözer. Proje üzerinde çalışan herkes kullanılan kelimelerin ve terimlerin tam anlamını bilir. Ayrıca farklı kullanıcıların aynı şey için farklı isimler kullanma problemini de çözer.

3.4.1.3. Varlık-Bağıntı Modelleme

Modellemeye data depolamak için hangi sisteme ihtiyaç duyulduğu konusunda sezgisel olarak nesneleri belirlemekle başlar. Uygun özellikler daha sonra her varlığa ayrılmaktadır.

3.4.1.4. Data Standartlaştırılması (Data Normalization)

Sistemdeki küçük anlamlı data birimleri ile başlar ve onları iyi formlu varlıklara çevirir. Varlık-bağıntı modellemede belirlenen varlıkların mantıksal veri grupları ve gereksiz data olmadığını kontrol etmek açısından faydalıdır (Britton ve Doake, 1993).

3.4.2. Nesne Yönelimli Yöntemler

Bütün nesne-yönelimli metodolojiler, geliştirme yaşam döngüsünün başından sonuna kadar çoğunlukla kullanılan “nesne-object” paradigması üzerine kurulmuştur. Bu yöntemde, yazılım geliştirme işi bir değişim işlemi olmak yerine bir evrimleşme süreci olarak nitelendirilmekte ve algılanmaktadır. Bu şekilde yazılım geliştirme yaşam döngüsünün değişik evrelerinin birbirlerinden kopmaması sağlanarak, evreler arasında süreklilik temin edilmektedir. Gereksinimleri toplanıp, nesne-yönelimli analiz süreci sırasında tanıtılan nesne, model güncelliği devam ettirilip evrimleşme geçirerek yazılım ürünü haline gelmesi ile sonuçlanan bütün ara evrelerde ortak temel kaynak olarak kullanılmaktadır (Britton ve Doake, 1993).

Nesne-yönelimli modelleme sistemlerinde, sistemin parçaları, kendi "bilgilerini (datalar)" ve bu bilgilerin "davranışlarını (metotlar)" içeren, sistem varlıklarına dönüştürülürler. Sistemin bu varlıkları nesne olarak adlandırırlar. Her nesne kendine ait bilgileri içerir. Her nesnenin kendi bilgisini diğer nesnelerden bir çerçeve ile ayırması "kapsülleme" olarak adlandırılmaktadır.Kapsülleme sayesinde sistemin her nesnesine diğer nesnelerden bağımsız olarak müdahale edilebilmektedir. Bu durum nesnelerin güncellenmesine olanak sağlar. Sistemin parçaları olan nesneler arasındaki iletişim ve etkileşim ise "mesajlar" ile yapılır. Bir mesaj; gönderilecek mesaj nesnesi, yapılacak metodun adı, metodun ihtiyaç duyduğu herhangi bir parametre parçalarından oluşmaktadır. Nesnelerin etkileşimi sayesinde programcılar daha yüksek tertipteki fonksiyonları ve karmaşık davranışları modelleyebilmektedirler. Ayrıca, sistemin nesneleri alt sınıf/üst sınıf hiyerarşisi içinde kalıtsanarak, bir sınıf şeklinde oluşturulurlar. Buna ek olarak, nesne yönelimli modelleme; nesnelerin diğer nesnelerden bilgi ve prosedürleri miras alabildiği (kalıtsallık) gerçeği ile karakterize edilir. Bu da yazılımın tekrar kullanımına yardımcı olur (Chep ve diğ., 1998).

NYP kavramları ilk kez Simula 67 (1967) ile tanınmıştır. Bugün NYP, çok sayıda etkileşimli yazılım bileşenlerini içeren karmaşık programların gerçekleştirilmesinde tercih edilen üstünlüğünü kanıtlamış bir yöntemdir. En çok bilinen ve kullanılan nesne-yönelimli metodoloji Rational Unified Process (RUP) tur.

90'ların en popüler yazılım geliştirme akımı olan nesneye-yönelim ve onlar üzerine geliştirilen yönetim ve geliştirme süreçleri, hızlı üretime izin vermeyecek kadar çok bürokrasi içermektedir. Müşteri, sadece kaliteli ürünlerin beklememekte, aynı zamanda hızlı üretim de istemektedir. Bu değişen müşteri ve sektör ihtiyaçları, daha çabuk ve kaliteli üretim yöntemleri arayışlarını doğurmuştur. Çevik yöntemler, bu ihtiyaçlara cevap vermeye çalışmaktadır.

3.4.3. Formal Notasyon

Formal notasyon, geliştirme üzerinde önemli bir rol oynar. Yazılım geliştirme aşamaları arasında hangi dili veya notasyonu kullanacağımız sorusuyla ilgilenmektedir. Data akış diyagramları, data modelleme (farklı veri nesnelerini ve aralarındaki ilişkiyi gösterir) gibi yapısal teknikler problemlerin farklı yüzlerini tanımlamak için kullanılan notasyonlardır. Formal notasyon, problemleri belirlemede kullanılan farklı bir dildir. Formal ile anlatılmak istenen; dillerin kesin, açık, iyi tanımlanmış kurallarla yönetilmesidir. Kurallar matematik ve mantık üzerine kuruludur (Britton ve Doake, 1993).

Yazılım sistemi, analistler tarafından müşteriye, tasarımcıya veya testçiye hepsinin anladıkları ortak bir dille modellenirse, çok karmaşık anlatımlar basitleşebilmektedir. Böylece müşteri sistem gereksinimleri raporunu okurken, teknik yazarlar kullanıcı kılavuzu yazarken veya test senaryoları hazırlanırken bu ortak dilden faydalanılmış olunmaktadır (Tokgöz, 2004).

3.4.3.1. UML (Unified Modelling Language)

1989-1994 yılları yazılım mühendisliğinde “Metot Savaşları” olarak bilinen dönemdir. Modelleme dillerinin çoğu yazılım yaşam döngüsünün bazı adımlarını tanımlamakta yetersiz kalmaktaydı. 90'lı yılların ortalarına doğru en çok tercih edilen 3 yöntem ön plana çıktı: Booch, OMT (Object Modelling Technology) ve OOSE (Object Oriented Software Engineering). Bunlar, zamanla birbirleri ile uyumlu birleşme göstererek UML(Unified Modelling Language) modelini oluşturdular. Her bir metodun eksi ve artıları vardı. Örneğin OMT analiz aşamasında güçlü fakat tasarım aşamasında zayıf, Boach tam tersine tasarımda kuvvetli fakat analizde zayıf kalırken OOSE ise davranımsal analizde kuvvetli fakat diğer alanlarda zayıftı. Metot savaşlarının sonunu notasyon açısından UML’in ortaya konması getirdi.

UML (Unified Modelling Language) yazılım mühendisliğinde nesne tabanlı sistemleri modellemede kullanılan açık standart olmuş bir görsel modelleme dilidir. Görsel modellemeler, var olan problemlerin gerçek dünya etrafında şekillenmiş halleridir. UML geliştirilen nesneye yönelik sistemlere ait birimlerin, özelliğini belirtmekte, görselleştirmekte ve dokümante etmekte kullanılır (Pooley, 2004).

Yazılım geliştirme işinde rol alan kişiler, farklı UML diyagramlarından faydalanırlar. Sistem gereksinimlerinin anlatılmasında Use-Case diyagramları, kavramsal (conceptual) model dediğimiz Class diyagramları, Colloboration Diyagram dediğimiz nesneler arası etkileşim ve işbirliği diyagramları, işbirliği diyagramının ifade ettiği şeyin aynısını bir zaman çizgisi üzerinde farklı bir gösterimle ifade eden Sequence Diyagramı, nesnelerin durum değişikliklerini ifade etmekte kullanılan State Diyagramları (Durum Diyagramları) , sistem mimarisinin paket yönünü özetleyen Package Diyagramları (Paket Diyagramları), yazılımın kurulacağı bilgisayarların konfigürasyonu, ağ ve yazıcı bağlantıları gibi detayları kapsayan Dağıtım Diyagramları.

Bu diyagramlar sayesinde büyük bir yazılımın yaşam döngüsü içinde her adımı kapsayan ve sisteme çeşitli gözlerden bakış açısı sunan bir modelleme tekniği olarak UML bir yazılım projesinin paydaşlarını konuşturan en kapsamlı ve en basit ortak dildir (Tokgöz, 2004).

3.4.4. Çevik Yöntemler

Yazılım projelerinin çoğu kullanıcı gereksinimlerini tanımlamada, değişiklikleri uyarlamada başarısızdırlar. Bu durum için 1990'ların sonunda 2000'lerin başında çeviklik ile ilgili "adapte edilebilir yazılım metodolojisi (adoptive software methodology)" geliştirilmiştir.

Bu yöntemle, bürokrasi mümkün olduğunca azaltılmakta, kısa aralıklarla yazılım parçalar halinde kullanıcıya teslim edilmektedir. Çabuk prototip oluşturmak için tekrarlamalı yaşam döngüsü kullanmaktadır. Küçük-orta boyutlu projeler (30-50 insandan az takımlar) için uygundur. Bu projeler, ihtiyaçların tam tanımlanamadığı, müşterinin başarılı bir proje için geliştiricilerle çalıştığı projelerdir.

Çevik yöntemler şunları farz etmektedir:

- Gereksinimler, tam olarak önceden bildirilemez ve değişime açıktırlar. Tasarım, değişiklikleri sezmelidir.
- Programcılar profesyoneldirler. En iyi sonuçlar, iyi bir takım ve güçlü teknik takım liderlerinden alınır.

Süreç, her projenin ihtiyacına göre adapte edilir (Kossiakoff, 2003).

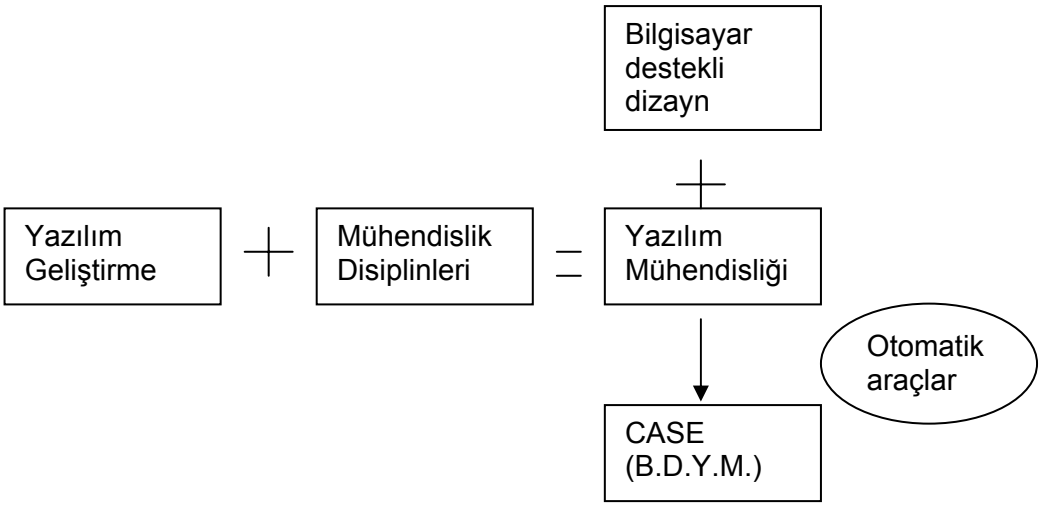
3.5. Yazılım Destek Araçları

Yazılım destek araçları, yazılım programlarının bakımına, gelişimine yardım eden yazılım programlarıdır. Bu araçların uygunluğu ve kalitesi başarı ve başarısızlık arasındaki farkı belirleyebilir. Destek araçları, tüm süreçte kullanılmaya ve ticari marketlerde daha fazla yer almaya başlamışlardır. Bu nedenle araçlar belli bir yatırıma gereksinim duymakta, proje yöneticileri ve sistem mühendisleri için ilgilenilmesi gereken bir konu olmaktadır (Kossiakoff, 2003).

Araçlar insanların yerini almazlar, ancak geliştirme sürecinde yardımcı olurlar. Bu tür araçlar önce makine mühendislerine yönelik olarak 'Bilgisayar Destekli Tasarım (CAD-Computer Aided Design)' adı ile yaygınlaştı. Genellikle üç boyutlu katı maddelerin çizimi, değişik açılar ve kesitler tanımlanarak görüntünün oluşturulması temelinde, başarılı bir geçmişe sahip oldular. Bunu elektronik konusundaki araçlar izledi ve günümüzde birçok mühendislik alanında tasarım artık büyük ölçüde bir yazılım çabası şekline dönüştü. Bu araçlar üzerinde, geliştirilecek sistemler yazılan programlar ile tanımlanmaktadır. Gerektiği yerde grafik ortamda yapılan girdiler araç tarafından otomatik olarak bir programa çevrilmektedir. 'Mum dibine ışık vermez' sözünü doğrularcasına yazılım mühendisleri bu gibi bir araç desteğine diğer

mühendisliklerden daha geç kavuştular. Ancak hızla yetişip değişik boyutlarda gelişen Bilgisayar Destekli Yazılım Mühendisliği (CASE) teknolojisi, diğer alanlardaki benzer teknolojiler ile bir karşılıklı etkileşime girdi ve sonra yazılım konusunda bulunan teknikler diğer alanlara da uygulanmaya başlandı (Kurnaz ve diğ., 2003).

Bilgisayar Destekli Yazılım Mühendisliği Araçları (CASE Tools); Yazılım geliştirme sürecini standardize etmek için tasarlanmış araçlar topluluğudur (Kossiakoff, 2003) . CASE (Computer Aided Software Engineering), yazılım mühendisliğine yazılım geliştirmede yardımcı olan otomatik araç ve yöntemleri kapsar. CASE araçları, yazılım geliştirme metodunun yerine geçmez, fakat bu metotlara bir ektir ve kaliteli ürünler üretmek için kaliteyi artıran araçlardır. CASE araçları, yazılım mühendisliğine yeni ve disiplinli bir mühendislik yaklaşımı getirmiştir. CASE araçları, yazılım yaşam sürecinde mühendislik prensiplerini uygulayan bir yaklaşımla maliyet ve zaman tasarrufu sağlar ve verimli, kaliteli ürünler ortaya çıkmasını sağlar. CASE araçları, yazılım yaşam sürecini otomatikleştirme kapasitesine sahiptir. CASE araçları, tek başına değer ifade etmeyen araçlardır. Değeri onu kullanan kişiye ve araçların durumuna bağlıdır. Böylece CASE araçları şu şekilde ifade edilebilir: Başka bilgisayar programlarının ve dokümantasyonlarının gelişimine, testine, analiz edilmesine ve bakımına yardımcı olan bilgisayar programlarıdır. CASE araçları, yazılım ve gereksinim mühendislikleri arasında bir köprüdür. Gereksinim mühendisliği; müşterinin bir yazılımdan beklentilerini, operasyonel ve geliştirme esnasındaki kısıtları da dikkate alarak ortaya koyma ve gereksinimleri belgeleme işlemlerini sistematik olarak yapabilme disiplinidir (Sodhi, 1991).



Şekil 3.2 : Bilgisayar Destekli Yazılım Mühendisliği genel bakış

CASE araçları, 5 temel bileşeni vardır:

- Veri Sözlüğü Deposu
- Yazılım Mühendisliği
- Proje Yönetimi Desteği
- Kalite Güvencesi Desteği
- Yazılım Yaşam Süreci Desteği

Bazı araçlar;

Modern CASE araçları: Tasarımcıya yapıyı, programı, data akışını, modülleri, birimleri tanımlamasını sağlayan grafik tabanlı diyagramlama araçları etrafında döner. Bu araçlar, yazılım süreçlerinden gereksinim analizi ve tasarım aşamalarında temeli oluşturur (Kossiakoff, 2003).

Gereksinim Yönetim Araçları: Birçok bilgisayar tabanlı araç, organize veri tabanının oluşturulmasına yardım eder ve otomatik olarak verilerin tutarlılık kontrollerinin yapılmasına, takip edilmesine, rapor hazırlanmasına ve diğer faydalı servislerin verilmesine destek verir.

Yazılım ölçüm araçları: Bilgisayar programlarının çeşitli teknik karakteristiklerinin yapısı ve karmaşıklığına bağlı olarak otomatik ölçülmesini sağlar.

Entegre gelişim destek araçları: Birçok araç uygun entegre destek fonksiyonlar setini sağlamak için mevcuttur. Örneğin bazı araçlar proje yönetimini, UML diyagramlama, gereksinim analizi, ölçü toplama yeteneklerini entegre eder. Böyle araçlar yazılım gelişiminin ilgili bölgelerindeki bilgilerin birbiriyle tutarlılığının sürdürülebilirlik problemini basitleştirir (Kossiakoff, 2003).

Yazılım mühendisleri arasında tartışılan konu “CASE araçlarının gerçekten etkin olarak kullanılıp kullanılmadığı”dır. Gün geçtikçe piyasada artan CASE araçlarından hangisi daha iyidir? CASE araçları kötü bir mühendisi, iyi mühendis yapmayacaktır. Önemli nokta, kullananın araç üzerindeki hâkimiyetidir.

Organizasyonların bilinçsizce yaptığı CASE yatırımları zararlara sebep olabilmektedir. Organizasyonların gerek duyduğu aktivitelerin doğru tespit edilip, bu doğrultuda CASE araç yatırımlarının yapılması isabetli olacaktır.

3.6. Yazılım Ölçümleri

Yazılım ölçümleri, süreç veya ürün karakteristiklerinin ölçümüne denir. Ürün ölçümleri yarı veya bitmiş ürün hakkında bilgi verir. Ölçümler; boyut, karmaşıklık, güvenilirlik,.. şeklinde tanımlanır. Örneğin; her ürün için boyut ölçümü tanımlanabilir. İhtiyaç sayısı, dizayn edilen parça sayısı, kodlanan modül sayısı, test senaryoları sayısı alınan ölçümler olabilir. Bazı geliştiriciler, boyutu fonksiyonlara göre, bazıları kodlama satırına göre ölçmektedirler (Pfleeger, 1991).

i. Proje ölçümleri

Proje yönetiminin (gereksinimlerin tutarlılığı, proje planının kalitesi, proje denetlemelerinin kalitesi, çizelgelere bağlılık,..) başarısının ölçümü ile ilgilidir. Yeni projeler için ölçmek zor olduğundan önemlidir. Bu metrikler projenin formalitesine, boyutuna, diğer özelliklerine uyarlanabilmektedirler (Kossiakoff, 2003).

ii. Süreç ölçümleri

Süreç ölçümleri tüm süreci ölçer. Yazılım prosesi hakkındaki sayısal verilerdir. Bassili ve Rombach (1988) GQM (goal question metric) paradigmasını önermişlerdir.

Goal (hedef): Programcının verimliliği, sürenin kısalması, üründen beklentilerin artması,,,,

Questions (sorular): Amaçlara yönelik cevaplanması beklenen sorular (nasıl süre azaltılabilir, hatası ayıklanan satırlar nasıl artırılabilir,..)

Metrics (ölçümler): Soruların cevaplanması ve hedefe ulaşılması için toplanan ölçümler. Örneğin; test sayıları, her ihtiyaç değiştikçe iletişim sayısı,...(Sommervile, 2000).

Örneğin; hataların yüzdesi bulunarak prosesin etkinliği ölçülebilinmektedir (Pfleeger, 1991).

iii. Teknik ölçümler

Yönetimden ve süreçten ziyade ürünün kalitesine odaklanır. Bu metrikler ürünün gelişimi için kullanışlıdır (Kossiakoff, 2003).

4. 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 ve Laanti, 2005). "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 ve Laanti, 2005).

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).

4.1. 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 ve Laanti, 2005) :

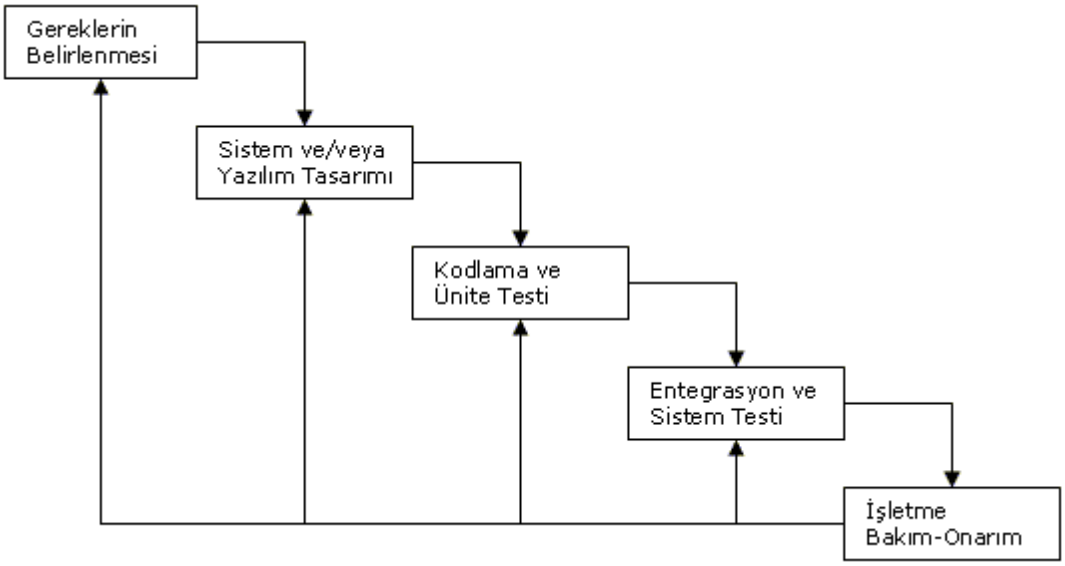
1. Disiplinsiz (Ad hoc)
2. Şelale(Waterfall) Modeli
3. Evrimsel Süreç Modelleri
    a. Artımsal (Incremantal) Model
    b. Spiral Modeller
    c. Rasyonel Bütünleştirme Süreci (RUP - Rational Unified Process)
4. Çevik (Agile) Metodolojiler
    a. Özelliğe Dayalı Geliştirme Modeli (FDD - Feature-Driven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

4.1.1. 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).

4.1.2. Şelale (Waterfall) Modeli (Royce 1970)

İlk yayınlanan yazılım geliştirme süreç modelidir. Şekil 4.1'de gösterildiği gibi her evre, ihtiyaç analizi, tasarım, kodlama ve ünite testi, entegrasyon ve sistem testi, işletme ve bakım - onarım safhalarını içermektedir.



Şekil 4.1 : Şelale Model işleyişi

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).

4.1.3. 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:

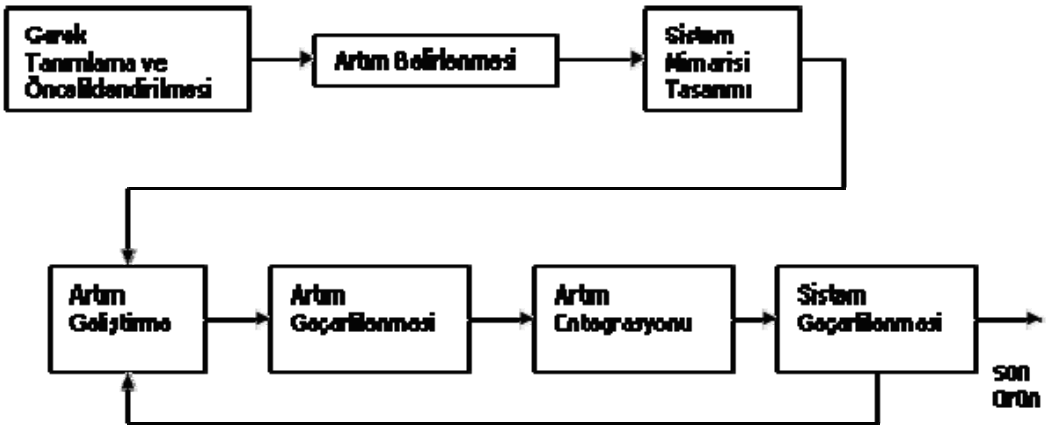
- Yöneticiler ilerlemeyi görmek için rapor istemekte, fakat sistem çabuk geliştirildiğinden her versiyonu için doküman hazırlamak maliyetli olabilmektedir.
- Sürekli değişim yazılım mimarisini bozmakta, değişiklikleri birleştirmek zor ve maliyetli olabilmektedir.
- Özel araçlar ve teknikler gerektirebilmektedir. 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.

❖ **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 4.2 'de de bu modelin süreç şeması görülmektedir.



Şekil 4.2 : Artımsal Model İşleyişi

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).

❖ **Spiral Model (Boehm 1988)**

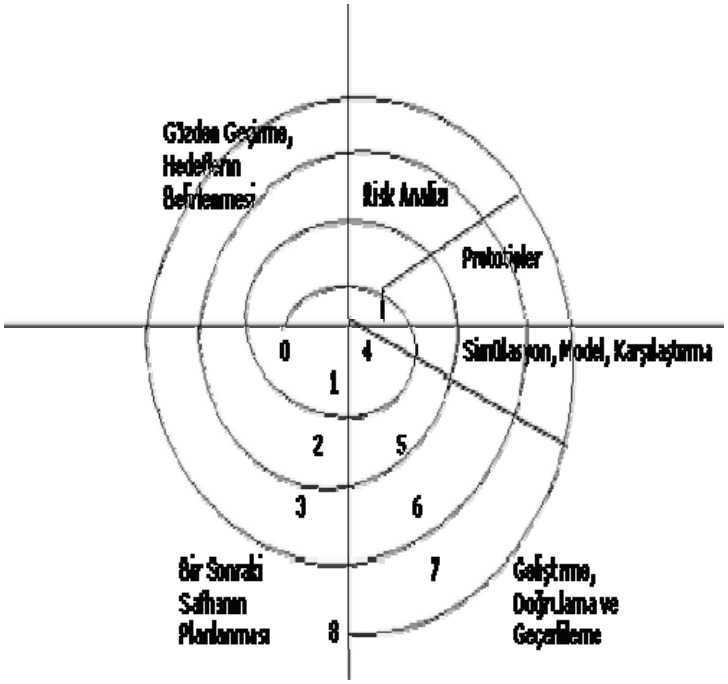
Modeli oluşturan her döngü (sistemin olabilirliği, ihtiyaç tanımlama, tasarım,...)

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



0. Yaşam Döngüsü Planı
1. Gerek Planı
2. Geliştirme Planı
3. Entegrasyon ve Test Planı
4. Kapsam
5. Gereklerin Belirlenmesi ve Geçerlileşmesi
6. Tasarım ve Geçerlileşme
7. Entegrasyon, Ünite Testleri ve Kabul Testleri
8. Son Ürün

Şekil 4.3 : Spiral Model işleyişi

❖ **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).

4.1.4. 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 ve Laanti, 2005).

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).

- **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).

- **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). 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 yöntemle 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 modellemeden faydalanırken, daha sabit öz parçaları ise şelale modelini izleyebilir (Kettunen ve Laanti, 2005).

5. 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.

5.1. 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).

5.1.1. 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.

5.1.2. 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).

5.1.3. 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).

5.1.4. 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 (Pfleeger, 1991).

5.1.5. 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 (Pfleeger, 1991).

5.1.6. 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 (Ollson, 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).

5.1.7. 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 (Pfleeger, 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).

5.1.8. Kullanılan Metot ve Araçlar

Her metot ihtiyaçların net belirlenmesi yolunda organize ve standartlaşmaya yardımcıdır (Pfleeger, 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-process-output): 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 (Pfleeger, 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).

5.1.9. 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 ve Doake, 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.

5.2.Tasarım

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 (Pfleeger, 1991).

5.2.1. 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 (Pfleeger, 1991).

5.2.2. 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).

5.2.3. 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 (Pfleeger, 1991).

5.2.4. 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 (Pfleeger, 1991).

5.2.5. 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 ve Doake, 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 ve Doake, 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).

5.2.6. 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).

5.2.7. 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).

5.3. 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 (Pfleeger, 1991).

5.3.1. 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).

5.3.2. Programlama Dilleri

5.3.2.1. 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 ve Doake, 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.

5.3.2.2. Programlama Dili Seçimi

Hangi dilin seçileceği belirlenirken tüm kriterlerin göz önünde bulundurulması gerekir. 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).

5.3.3. Kullanılan 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).

5.4. 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).

5.4.1. 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 (Pfleeger, 1991) .

Kodlama aşamasında, yazılım sistemlerinin çok sayıda karmaşık formül, aktivite, algoritma içermesinden ve bazen müşterilerin isteklerindeki belirsizliklerden dolayı hatalar olabilmektedir (Pfleeger, 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

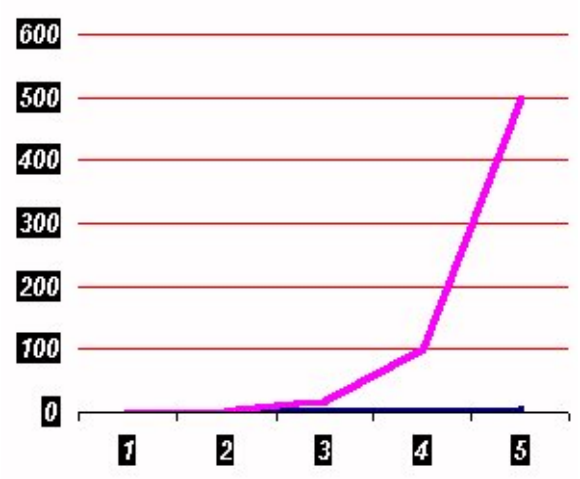
a. **Fonksiyon testi:** Entegre sistemde istenilen fonksiyonların testi yapılır. Bu safhada "doğru yazılımın üretilip üretilmediği" incelenir. Bu test "kara kutu" testi olarak da adlandırılır. Yazılımın gereksinme duyulan şeylere yanıt verip veremediği ve işlevselliği sınanır.

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 (Pfleeger, 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).



Şekil 5.1: Bir Hatayı Düzeltmenin Proje Safhasına Bağlı Olarak Değişen Masrafı

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.

5.4.2. 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 (Pfleeger, 1991).

5.4.3. Test Dokümantasyonu

Testin kompleks ve zorluklarını kontrol etmek için dikkatlice hazırlanmış test dokümantasyonu kullanılır (Pfleeger, 1991).

5.4.4. 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,1991).

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).

5.5. 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 (Pfleeger, 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).

5.6. 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 (Pfleeger, 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).

5.6.1. 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.

5.6.2. Kullanılan Araçlar

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

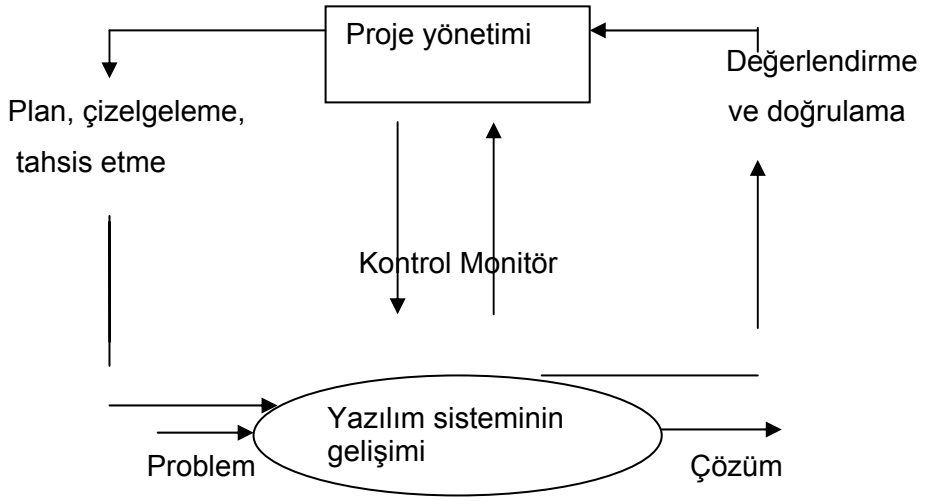
5.6.3. Kullanılan Ölçümler

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).

6. PROJE YÖNETİMİ

1969 yılında Amerika'da kurulmuş olan PMI tarafından geliştirilen rehber PMBOK (A Guide to the Project Management Body Of Knowledge) bugün konusunda ANSI ve IEEE tarafından standart olarak kabul edilmekte ve birçok sektörde proje yönetimi referans başvuru kitaplarından biri olarak kullanılmaktadır.

PMBOK'ta Proje Yönetimi şöyle tanımlanmaktadır: Bir projenin gereksinimlerini karşılamak üzere bilgi, beceri, araç ve tekniklerin tüm aktivitelere uygulanması (Karadağ, 2002).



Şekil 6.1: Sistem gelişiminin ve proje yönetiminin ayrı ayrı fakat birbiriyle ilişkilendirilerek gösterilmesi (Britton ve Doake, 1993).

Tüm yönetimler için genel prensipler; amaçları, hedefleri anlamak, kısıtları bilmek, kısıtlar altında hedefe ulaşmak için plan yapmak, planı izlemek, gerekli yerlerde düzeltmeler yapmak, çalışma ortamını verimli, huzurlu, pozitif tutmaktır (Blum, 1992). Planlama, izleme ve kontrol çerçevesiz ve yazılım takımının işleri takibindeki dikkatleri olmaksızın proje kaosa doğru sürüklenebilmektedir (Britton ve Doake, 1993). Projenin iyi yönetilmesi, yazılım projelerinin başarılı olması için gerekli önemli faktörlerden biridir

Yazılım yönetimi birkaç yönden diğer mühendislik yöntemlerinden ayrılmaktadır. Bunlar:

- Ürün soyuttur: Teftiş etmek için diğerleri tarafından oluşturulan dokümanlara inanılmak zorunluluğu vardır.
- Standart yazılım süreçleri yoktur: Diğer mühendislik dallarında standart süreçler vardır (örneğin köprü yapımı). Yazılım süreçleri ve ürün tipleri arasındaki ilişki tam olarak anlaşılmaz.
- Büyük projeler "one-off" projeler: Farklı projeler, bilgi ve komünikasyondaki hızlı teknolojik gelişmeler, bir önceki deneyimi geçersiz kılabilmektedir (Sommervile, 2000).

Proje yönetiminin görevleri, planlama, ilerlemeyi takip etme, rapor hazırlama, zaman ve maliyet açısından yük getirmektedir. Şu bilinmelidir ki: proje yönetimi sihirli bir değnek değildir. Birçok yazılım projesi hala geç teslim edilmekte, planlanan bütçeyi aşmakta ve kullanıcı ihtiyaçlarını tam karşılayamamaktadır. Bu sadece yazılım projelerine ait bir özellik değildir, diğer endüstrilerde de aynı problemler vardır. Bu nedenle, proje yönetim metotlarını sunarken bunların her zaman harcanan zamana ve maliyete değip değmeyeceği sorusunun akla gelmesi sürpriz olmamalıdır (Britton ve Doake, 1993). Unutulmamalıdır ki; projenin iyi yönetilmesi, yazılım projelerinin başarılı olması için gerekli önemli bir faktördür, fakat tek faktör değildir.

6.1. Proje Yönetim Fonksiyonları

6.1.1. Proje Planlama

Proje yönetiminin en önemli süreçlerinden biri planlama sürecidir. "Plan yapmada başarısız olursan, başarısızlığı planlamalısın" sözü planlamanın önemini vurgulamaktadır (Natarajan, 2004).

Plan, uygun olan kaynakla sistemin ne yapmak zorunda olduğunu uzlaştırmalıdır. Projenin amaçlarına göre kaynak dengeleri bir projeden diğerine değişecektir. Bazen, müşteri çok para harcamayı kabul edip, fakat bu doğrultuda sistemin çabuk teslimini isteyebilir, bazen geliştirici herhangi bir uzmanlık alanında yetersizdir. Proje yöneticisinin çalışan bir plan üretmesi için tüm bu çeşit bilgileri elinde olmalıdır (Britton ve Doake, 1993).

Planın başlıca parçaları; müşterinin ne alacağı, kilometre taşları (ne zaman alacağı) ve bütçedir (maliyetinin ne olacağı). Süreci tüm detaylarıyla tarif eder (Kullanılacak CASE araçlarını, yönetim nesnelerini, süreç modelini,..) (Schach, 1993).

6.1.2.Tahmini Hesaplama

Hiçbir müşteri, projenin ne kadar süreceğini ve ne kadara maliyet edileceğini bilmeden projeyi onaylamaz (Schach, 1993). Yazılım geliştirme projelerinde müşteri projenin maliyetini ve sürecin uzunluğunu hesaplamak ister. Yazılım mühendisliği tam doğru hesaplayamayacaklarının bilincindedir. Asıl amaç, projenin ilerleme aşamasında daha fazla kontrole sahip olabilmektir (Pfleeger, 1991). Doğru hesaplar yapmak proje planının başarısı için önemlidir (Britton ve Doake, 1993).

Yazılım geliştirme projesinin toplam maliyeti içinde 3 parametre vardır:

- Donanım, yazılım maliyetleri, bakım dahil,
- Ulaşım, eğitim masrafları,
- Efor maliyeti (yazılım mühendisleri giderleri ve diğer elektrik,su,… giderleri) (Sommervile, 2000).

6.1.3. Görevleri Belirleme

Görev tanımlamaları proje planının önemli bir parçasıdır, fakat tek başına proje yöneticisine çizelgeleme yapmak için yeterli bilgi sağlamaz. Takım üyelerinin ve gerekli kaynakların proje için tam olarak ne zaman uygun oldukları da ayrıca önemlidir (Britton ve Doake, 1993).

6.1.4. Çizelgeleme

Yazılım istenilen ihtiyaçları karşıladığı takdirde faydalıdır. Proje çizelgesi projenin safhalarını tek tek ayırıp, aralarındaki ilişkiyi kurup, bu aktivitelerin ne kadar süre alacağını hesaplar (Pfleeger, 1991).

Proje yöneticisinin, uygun zaman, kaynak miktarını ve tam olarak ne yapılması gerektiğini bildiği zaman, görevleri bölüştürmesi ve çizelgeleme yapması mümkündür. Pazarda bu iş için yardımcı birçok otomatik proje yönetim araçları vardır. Bu araçlar projenin belirlenen zaman ve bütçe limitlerinde teslimi için bir iş planı çıkarmakta ve kaynakların optimum kullanımını sağlamaktadır. Araçlar proje yöneticisinin iş yükünü hafifletmektedir. Örnek verecek olursak; **PERT Chart** bir grafik tabanlı ağdır. Proje içindeki görevleri ve bu görevler arasındaki ilişkileri betimler. **Gantt Chart** ise basit bir yatay çubuk grafiğidir. Bu grafikte proje

görevlerine karşılık takvimler betimlenmektedir. Her çubuk isimlendirilmiş bir proje görevini temsil etmektedir. Görevler liste şeklinde grafiğin sol dikey çizgisinde yer almakta, takvim zamanları da grafiğin x ekseninde gösterilmektedir (Britton ve Doake, 1993).

6.1.5. İzleme ve Kontrol Etme

Yazılım geliştirme sırasında proje yönetiminin asıl görevi projenin ilerleyişini izlemek ve ilerlemeyi kontrol etmektir. Bunu başarmak için, ilerlemenin zaman, maliyet ve kalite yönünden nasıl gerçekleştiğini ölçmek önemlidir. Proje yöneticisi, plana göre projenin durumunu, plandan herhangi bir sapmanın olup olmadığını, bunun nedenlerini, alınması gereken önlemleri ve bunların proje üzerindeki sonuçlarının farkında olmalıdır. Proje kontrolü için iyi bir iletişim gereklidir. Yazılım geliştirme ilerleyişini izleyebilmek için bir mekanizma olması gereklidir. Mekanizma genellikle şunları içerir: belli aralıklarla veya her tamamlanan görevin sonunda takım üyeleri tarafından raporlar sunulması ve durumu değerlendirmek, geleceğe dair kararlar verilmesi için önceden planlanan toplantılarla, gözlemlerin yapılması.

Tüm yazılım geliştirme projeleri planlanan zaman ve maliyette teslim için yönetilmeye ihtiyaç duyarlar. Teknik yeteneğin yanında iyi bir yönetim de yüksek kalite sağlayan anahtarlardan biridir. Bugün otomatik araçlar, yazılım ölçümleri proje yönetiminin görevlerini azaltsa da sürecin etkinliğinden ve ürünün kalitesinden sorumlu kişi yine proje yöneticisidir (Britton ve Doake, 1993).

6.2. İnsan Yönetimi

Yazılım yönetimi esas olarak insan yönetimi ile ilgilenmektedir. Bu nedenle yöneticiler, insan faktörlerini anlamalıdırlar ki kendilerinden ve çalışanlarından gerçekçi olmayan taleplerde bulunmasınlar.

Yazılım organizasyonunda çalışan kişiler ortaya bilgilerini sunmaktadır. Yöneticiler de insanlara yapılan yatırımdan en iyi geri dönüşü almakla görevlidirler. Başarılı organizasyonlarda bu, organizasyonun çalışanlarına saygı gösterdiği zamanlar başarılabilmektedir. Çalışanlara çeşitli düzeyde sorumluluklar verilmekte ve yetenekleriyle orantılı bir şekilde ödüllendirilmektedirler.

Proje yöneticileri, teknik veya teknik olmayan problemleri, takımındaki çalışanları en efektif yoldan kullanarak çözmek, insanları motive etmek, işleri plan ve organize etmek ve işin doğru yapılmasını sağlamak zorundadırlar. Yazılımcıların sosyal (iş arkadaşlarıyla toplanma ortamları sağlanması,…), saygı (değerli olduklarının

gösterilmesi), kendilerinin farkına varma (kendi işlerine ait sorumluluklar verilmesi, görev talep etmeleri sağlanması, yeteneklerini geliştirecekleri eğitimlerin verilmesi) ihtiyaçları sağlanmalıdır (Sommervile, 2000).

6.2.1. Grup Çalışmaları

Yazılım geliştirme grupları küçük ve uyumlu olmalıdır. Küçük gruplar, iletişim problemlerini azaltmaktadır. Her grup projeyi oluşturan alt projeden sorumludur. Tüm grup ise yuvarlak bir masa etrafında birbirlerinin ofislerinde toplanarak tüm proje için iletişim kurabilirler.

Grup içindeki iletişim şu faktörlere bağlıdır:

Grup üyelerinin statüleri: Yüksek statülü üyeler, iletişim kurmada veya kritik durumlarda geri planda duran kişilerle yapılan iletişimlerde daha baskın olma eğilimindedirler.

Grup büyüklüğü: Grup büyüdükçe iletişimdeki etkinlik azalmaktadır. Normal şartlarda alt grup 8 kişiden fazla olmamalıdır.

Grup yapısı: Formal yapılı olmayan gruplar formal-hiyerarşik yapılı gruplara göre daha efektif iletişim kurmaktadırlar. Hiyerarşik gruplarda aynı seviyedeki kişiler birbirleriyle konuşmamakta, sadece yöneticileriyle iletişim kurmaktadırlar. Bu da çok gruplu büyük projelerde gecikmelere ve yanlış anlamalara sebep olabilmektedir.

Grup bileşimi: Aynı tip kişilikli kişilerin oluşturdukları gruplardaki iletişim de kolay olmamaktadır.Tüm cinsiyetlerden personelin oluşturduğu grup iletişimi daha iyidir. Kadınlar daha etkileşim-tabanlı olma eğilimindedirler. Grup için etkileşim kontrolcüsü ve kolaylaştırıcısıdırlar.

Fiziksel çevre: İletişimde büyük faktördür. Çalışanların işlerine konsantre olabilecekleri, doğal aydınlatmalı, kişiye özel düzenlenmiş ortamlar çalışanı her zaman motive edecektir.

Yazılım mühendisliği, bilmeye, kavramaya ilişkin ve sosyal yönlü bir aktivitedir. Dolayısı ile insan faktörleri insanların nasıl yazılım gerçekleştirdiklerini anlamada önemlidir. İnsan yönetimindeki zayıflık, proje başarısızlığına sebep olan en belirgin hususlardan biridir (Sommervile, 2000).

6.3. Risk Yönetimi

Risk yönetimi, riskleri belirleyip, proje üzerindeki etkilerini minimize etmeye denir (Sommervile, 2000).

Birçok yazılım projesi kaçınılmaz şekilde çeşitli tip ve derecede belirsizlikler içermektedir. Doğru risk değerlendirmeleri yapılamadığı sürece, projeler kolaylıkla kontrolden çıkabilmekte ve belirgin bir şekilde ek kaynak harcayabilmektedir. Denver Uluslararası Havaalanı'nın bagaj dağıtım sistemi örneğini vermek gerekirse, bu yazılım projesi havaalanının açılmasını 16 ay geciktirmiş, bütçenin 2 milyar dolar aşımına yol açmıştır.

Bilgi sistemleri literatüründe çoğu yazılım geliştirme zorlukları yetersiz bilgiden kaynaklanan belirsizliklere atfedilmiştir (Na ve diğ., 2004). Yazılım geliştirmede oluşan belirsizlikler yüksek proje başarısızlığı oranlarına ve bütçe aşımının %50 fazla olmasına neden olmaktadır (Gibbs, 1994). Risk analizleri; yazılım takımına, bu tür belirsizlikleri anlamasına ve yönetmesine yardım eder (Natarajan, 2004).

6.3.1. Yazılım Risk Kategorileri

Belirsizlikler olabilir ve iyi yönetilemezlerse istenmeyen sonuçlar ortaya çıkabilir (Natarajan, 2004). Yazılım risklerini şu şekilde gruplayabiliriz:

- **Proje Riskleri:** Proje planını tehdit etmektedir. Proje risklerinin gerçekleşmesi durumunda çizelgeler değişmekte, maliyetler artmaktadır. Bütçe, müşteri, kaynak problemlerini içermektedir (Natarajan, 2004). Zamanı, kaynakları etkilemektedir.
- **Ürün Riskleri:** Kalite ve zamanlamayı tehdit etmektedir. Gerçekleşirse uygulama çok zor olmakta ya da mümkün olmamaktadır. Tasarım, uygulama, ara yüz doğrulaması, bakım problemlerini içermektedir. Kaliteyi ve performansı etkilemektedir (Sommervile, 2000).
- **İş riskleri:** Projeyi ve ürünü tehlikeye atmaktadır (Natarajan, 2004). Organizasyon gelişimi, yazılımı elde etmeyi etkilemektedir .

Olası Yazılım Riskleri:

- Deneyimli elemanın proje bitmeden projeyi bırakması (proje riski)
- Farklı önceliklerle organizasyonel yönetimde değişikliğin olması (proje riski)
- Proje için gerekli donanımın zamanında teslim edilmemesi (proje riski)
- Gereksinimlerde beklenenden fazla değişikliğin olması (proje+ ürün riski)

- Gerekli ara yüz tanımlamalarının gecikmesi (proje+ ürün riski)
- CASE araçlarından beklenen performansın alınamaması (ürün riski)
- Kullanılan teknolojinin değişmesi, yenisinin eskinin yerine geçmesi (iş riski)
- Ürün tamamlanmadan rakip ürünün pazara sunulması(iş riski) (Sommervile, 2000).

6.3.2. Risk Yönetim Süreçleri

1. Risk tanımlama: Olası proje, ürün, iş riskleri belirlenir. Risk tanımlama, takım üyeleriyle beyin fırtınası yaparak oluşturulabileceği gibi yöneticinin deneyimlerine göre de oluşturulabilir. Risk tipleri; teknoloji, insan, organizasyon, araçlar, ihtiyaçlar, hesaplamalar,...

2. Risk analizi: Tanımlanan risklerin ihtimal ve sonuçları değerlendirilir.

Risklerin olasılığına göre değerlendirilmesi: çok düşük (< 10%), düşük (10-25%), normal (25-50%), yüksek (50-75 %), çok yüksek (>75%) şeklinde yapıla bilinir. Risk sonuçları da şu şekilde değerlendirilir: facia, ciddi, tolerans gösterile bilinir, önemsiz.

Bu bilgilerle bir tablo oluşturulur. Tablo bilgiler oluştukça, planlar yapıldıkça değiştirilir. Analizden ve sıralamadan sonra proje sırasında düşünülmesi gereken en önemli riskler üzerinde karar verilir. Genellikle facia, çok ciddi riskler her zaman incelenmelidir.

Örnek vermek gerekirse;
Proje riski: Proje gereksinimlerine göre eleman bulmak zor.
Olasılığı: Yüksek
Etkisi: Ciddi.

3. Risk planlaması: Riskten kaçınma ya da riskin proje üzerindeki etkilerini minimize etmek için planlar hazırlanır.

Stratejiler 3 gruba ayrılabilir:

- Kaçınma stratejileri: Bu stratejiyi izlemek, riskin oluşma olasılığı azaltılacak anlamına gelir. Örnek vermek gerekirse;

  Önemli risk: hatalı, kusurlu bileşenler.

  İzlenmesi gereken strateji: Potansiyel hatalı bileşenleri, güvenilirliği bilinenle değiştirmek.

- Minimize etme stratejisi: Riskin etkisi azaltılacak anlamına gelir.

  Örneğin; personel hastalığı riskine karşı alınması gereken stratejiye çalışanların birbirlerinin işinden anlayacak şekilde işlerin organize edilmesi stratejisi verilebilir.

- Olasılık planları: Eğer en kötüsü olursa, onunla uğraşmak için plan yaparsın.

  Örneğin; organizasyonel finanssal problemler riskine karşı alınabilecek strateji, yönetime özet bir dokuman hazırlanarak projenin hedeflere ulaşmada ne çok katkısı olduğunun gösterilmesi olabilir.

4. Risk izleme: Risk sürekli değerlendirilir, riski hafifletmek için yapılan planlar risk hakkında bilgiler ortaya çıktıkça revize edilir. Riskleri direk izlemek zor, bu nedenle risk olasılığı ve sonuçları hakkında ipucu veren diğer faktörlere bakmak zorunluluğu vardır. Bu faktörler, risk tiplerine bağlıdır.

Risk tiplerini değerlendirmek için birkaç faktöre örnek Tablo 6.1'de verilmiştir.

Tablo 6.1: Risk tipleri ve potansiyel göstergelere örnek

Risk tipi	**Potansiyel göstergeler**
İnsan	Çalışanın moral bozukluğu, üyeler arası iletişim zayıflığı, mevcut işler
İhtiyaçlar	Birçok ihtiyaç değişikliği isteği, müşteri şikayetleri
Araçlar	Araç kullanmaya isteksizlik, şikayetler, daha yüksek seviyeli araç istekleri

Risk izleme sürekli bir süreç olmalı ve riskler toplantılarla ele alınmalı ve tartışılmalıdır (Sommervile, 2000).

6.3.3. Risk Yönetim Stratejilerinin Yazılım Ürün ve Süreç Performansı Üzerindeki Etkileri – Nidumolu Modeli

Risk yönetimi teknikleri projelerde gömülü olan risklerin negatif etkisini minimize etmek için tasarlanmıştır. Örneğin; standartların geliştirilmesi, kullanıcı gereksinim analizi önemli risk yönetim teknikleridir (Na ve diğ., 2004).

Nidumolu (1996), 2 risk yönetimi tekniğinin süreç ve ürün performansı ile ilişkisi üzerine bir model sunmaktadır. Bu model, standardizasyon ve gereksinim belirsizliğinin performansı etkilediğini öne sürmektedir. Modele göre gereksinim belirsizliği sonraki safhalardaki performans riskini artırmakta, yazılım geliştirme standartları bu riski azaltmaktadır. Nidumolu, diğer süreçlerde görülecek performans riskinin artmasının ürün ve süreç performansı üzerinde direk negatif etkiye yol

açacağını ileri sürmektedir. 64 US yazılım firmasından toplanan datalar, Nidumolu modelini desteklemektedir. Modele göre tanımlamalara bakacak olursak;

Proje performansı: Hem süreç performansını hem de ürün performansını kapsamaktadır.

Süreç performansı: Yazılım geliştirme süreci için performans ölçüsüdür ve şu şekillerde tanımlanır:

* Proje süresince öğrenmek (geliştirme, anahtar tekniklerin kullanımı için bilgi toplama),
* Yönetimin projeyi ne derecede kontrol edebildiği (proje maliyeti, çizelgesi, kontrol standartlarına ve incelemelere uyma),
* Gelişim süresince geliştirici takım ile kullanıcılar arasındaki etkileşimin kalitesi (eğitim desteği, kaliteli iletişim, kullanıcıların düşünceleri, katılımları),

ile tanımlanır.

Ürün performansı: Yazılımın teknik performansı, yazılımın kullanıcının ihtiyaçlarını ne derecede karşıladığı, değişen kullanıcı ihtiyaçlarına ve yeni ürünlere ne kadar uyumlu olduğu ile tarif edilir.

Gereksinim belirsizliği ile; kullanıcı ihtiyaçlarını belirlemek için gerekli bilgi ile geliştiriciler için uygun olan bilgi arasındaki farklılık kastedilmiştir. 3 boyutla ilişkilidir:

- Süreç boyunca ihtiyaçların değişmesi,
- İhtiyaçların kullanıcılar arasında farklılaşması,
- Kullanıcı ihtiyaçlarının ürün ihtiyaçlarına çevrilmesi (objektif bir prosedür çerçevesinde gerçekleşme derecesi).

Standardizasyon; organizasyonlarda farklı seviyelerde gerçekleşmekte, bu modelde düşünülen, standardize edilen kontroldür. Takımdan beklenen çıktıların, gelişim sırasında kullanılacak teknik ve araçların kontrollerinin standardize edilmesidir.

Birçok çalışma etkin risk tanımlama ve kontrolü için çeşitli risk yönetim teknikleri önermişlerdir. U.S. yazılım şirketleri üzerinde yapılan çalışmalar, gelişim standardizasyonun ve gereksinim analizinin, daha sonraki safhalardaki risklerin azaltılmasında etkin teknikler olduğunu ve dolayısı ile yazılım proje performansı üzerinde etkilerinin büyük olduğunu göstermiştir.

2002 de yapılan bir çalışma, Na ve diğ. (2004) Nidumolu modelini CMM sınıflandırmasının US'den düşük olduğu IT'nin geliştiği bir ülkede Kore'de test etmiştir. Fakat US'de alınan sonuçların aynısı alınamamıştır. Analizler, kalan

performans riskinin ve onun proje performansı üzerine etkilerinin iki ülkede farklı sonuçlar verdiğini ortaya çıkarmıştır. Yazarlar, bu farklılığı anlamaya IT yeteneklerinin yardım edeceğini ileri sürmektedirler. Kore gibi ülkelerde sistematik risk yönetimi henüz olgunlaşmamıştır. Bu nedenle, sonraki aşamalar için risk kontrol çözümlerini kontrol etmeden projenin kalan performans riskini modellemek farkı oluşturan en önemli sebeptir. Ancak, gelişim standardizasyonunun ve gereksinim analizinin, tüm yazılım proje performansı üzerinde olumlu etkisinin olduğu gözlenmiştir (Na ve diğ., 2004).

6.4. Konfigürasyon Yönetimi

Sistemi oluşturan tüm yapılan şeyleri bilmeyi kapsar. Bunların yönetimi, sistemin sürümleri sunmakla gerçekleştirilir (Natarajan, 2004). Konfigürasyon yönetimi, sisteme yapılan tüm değişiklikleri dokümante etmekten ve tüm sürümleri izlemekten sorumludur.

Bu, sistemin tüm yaşam döngüsü boyunca uğradığı değişiklikler karşısında kalitesinin korunmasını sağlar (Britton ve Doake, 1993).

Konfigürasyon yönetiminin, gelişim sürecinde ve sonrasında en kritik aktivitelerden biri olmasının nedenlerinden bazıları;

- Yazılımın soyut olması ve iyi tanımlanmış bileşenlerin azlığı yazılımın anlaşılmasını zorlaştırır.
- Daha fazla ara yüzün olması, etkilerinin fazla olması takip etme zorluğu verir.
- Herhangi bir değişiklik sistemin derinliklerine kadar yayılır.
- Herhangi bir değişiklik tüm sistemin testinin gereksinimini sağlayabilir.
- Hataların oluşması ani olur.
- Değişiklik için kodları eklemek basit ve hızlı fakat büyük riskler taşımaktadır (Kossiakoff, 2003).

Konfigürasyon yönetimi, gelişen sistem ürününün yönetimi için standart ve prosedürlerin geliştirilmesi, uygulanmasını gerektirir. Yazılımın birçok versiyonu çıkarıldığından, gelişen sistem yönetilmek zorundadır. Bu versiyonlar, değişiklik, hata düzeltme gereklerini ve farklı donanım, işletim sistemlerine adaptasyon önerilerini bir araya getirebilmektedir. Uygulanan bu değişikliklerin ve bu değişikliklerin yazılımda nasıl yer ettiğinin izlenmesi gerekmektedir.

Konfigürasyon yönetimi prosedürleri, önerilen sistem değişikliklerinin nasıl kayıt edileceğini ve işleneceğini, bu değişikliklerin sistem bileşenleriyle nasıl ilgili olduklarını ve sistemin farklı versiyonlarını tanımlamak için kullanılan metotları tanımlar.

Konfigürasyon yönetimi araçları, sistem bileşenlerinin versiyonlarını saklar, bu bileşenlerden sistemi kurar ve müşteriye sunulan versiyonları takip ederler.

Konfigürasyon yönetimi, yazılıma yapılan değişiklikleri kontrol ettiğinden kalite yönetiminin bir süreci olarak görülebilir. Konfigürasyon yönetimi prosesleri ve ortak dokümanları standartlara dayalı olması iyidir. IEEE 828-1983 bu standartlara bir örnektir. Bu, konfigürasyon yönetimi planları için standartları tanımlar. Daha sonra bu standartlar, konfigürasyon yönetimi el kitabında ya da kalite el kitabında yayımlanır.

6.4.1. Konfigürasyon Yönetimi Aktiviteleri

Konfigürasyon yönetimi aktiviteleri arasında konfigürasyon yönetimi planlama, değişim yönetimi, versiyon ve piyasaya sürme yönetimi gelmektedir.

Konfigürasyon yönetimi planlama sürecinde ileriki bakım safhalarında gerekli olacak tüm dokumanlar kontrol edilmelidir.

Entegre CASE araçları, dokuman ve değişikliklerden etkilenen bileşenler arasında değişiklikleri ilişkilendirmeyi mümkün kılar. Dokümanlar (tasarım dokumanı) ve program kodu arasındaki bağlara bakım yapılabilir, böylelikle bir değişiklik öngörüldüğünde değiştirilmesi gereken her şeyi kolayca bulmak mümkün olur. Buna rağmen, birçok organizasyon konfigürasyon yönetimi için entegre CASE araçları kullanmazlar.

Değişim Yönetimi; yazılım sistemleri için değişim hayatın gerçeğidir. Organizasyonel ihtiyaçlar ve gereksinimler yazılım yaşam döngüsü boyunca değişmektedirler. Bu da yazılım için değişiklikleri gerektirmektedir. Tanımlı değişim yönetim süreci ve ilişkili CASE araçları değişikliklerin kaydedilmesini ve sisteme mali açıdan faydalı bir yoldan değişikliklerin uygulanmasını sağlar.

Değişim yönetim prosedürleri, değişikliklerin maliyetleri ve faydalarının analiz edilmesini ve yapılacak değişikliklerin kontrollü yoldan yapılmasını sağlayacak şekilde tasarlanmalıdır.

Versiyon ve piyasaya sunma yönetimi; büyük sistemlerde birçok farklı versiyonda yüzlerce yazılım bileşeni vardır. Versiyon yönetim prosedürleri, her bileşen versiyonunu tanımlamanın tek, belirli bir yolunu sunmalıdır (Sommervile, 2000).

6.5. Kalite Yönetimi

Yüksek kalitede yazılım üretmek her geçen gün daha da önem kazanmaktadır. Bu nedenle öncelikle kaliteli sistemi doğru tanımlamamız gerekmektedir. Fakat bu görüldüğü kadar kolay değildir. Kalite, kullanıcı ve yazılım geliştirici gözüyle farklıdır. Kullanıcı için sistemden iyi bir fiyata zamanında istediğini almaktır. Kullanıcı için sistemin neler yapabileceği, ne kadar kullanıcı dostu olduğu, güvenilirliği, maliyeti önemlidir. Geliştirici için ise ayrıca bunların başında iyi program, iyi tasarım, kaynakların etkin kullanımı, iyi yapılandırılmış veri tabanı önemlidir (Britton ve Doake, 1993). Görüldüğü gibi, yazılımda kalite karmaşık bir kavramdır, basit bir yolla ifade edilememektedir. Klasik olarak kalite; geliştirilen ürünün, spesifikasyonlarını karşılaması olarak tanımlanmaktadır.

Yazılımda karşımıza çıkan birkaç problem vardır; tanımlanan ihtiyaçların dışında ihtiyaçların çıkması, kesin kalite karakteristiklerinin bilinmemesi, tüm yazılım spesifikasyonlarını dokümante etmenin zor olmasından dolayı bu spesifikasyonları sağlıyor olsa da kullanıcının bunun farkında olmaması ve yüksek kalite olduğunu düşünmemesi,..gibi (Sommervile, 2000).

Kalite yönetim sistemi etkili değil ise son ürünü test etmek için metotların nasıl kullanıldığının, dokümanların nasıl tamamlandığının, geliştirme planının, proje gözden geçirmelerinin, konfigürasyon kontrolünün, araç ve tekniklerin gelişmişlik düzeylerinin hiçbir önemi olmayacak ve proje başarısızlıkla sonuçlanacaktır. Kalite standartları, ancak etkili kalite yönetim sistemi ile yerine getirilebilmektedir. Dolayısı ile kalite; kalite, zaman, bütçe güvencelerini sağlamak amacıyla tanımlanan ve uygulanan yönetim ve tekniksel prosedürler doğrultusunda yazılım ürünlerine yerleştirilmektedir (Gill, 2005).

Kalite Yönetimine bağlı birimleri şu şekilde sıralayabiliriz:

6.5.1. Kalite Güvence

Ürün ve süreç standartlarının belirlenmesi önemlidir. Çünkü bu bilgiler birçok deneyim ve hatanın sonucunda hesaplanarak edinilmiş standartlar olduğundan en uygun çözümü sunmakta, kontrolü kolaylaştırmaktadırlar.

Standartları ürün ve süreç standartları olarak incelersek;

Ürün standartları: Bunlar geliştirilen yazılım ürünlerine uygulanan standartlardır. Doküman standartları(gereksinim dokümanları yapısı,…gibi), programlama dilinin nasıl kullanılacağını tanımlayan kodlama standartları,… gibi.

Süreç standartları: Gelişim aşamasında izlenen sürece uygulanan standartlardır. Gereksinim belirleme, tasarım, onaylama süreçlerini ve bu süreçlerde oluşturulan dokümanları belirlemektedir.

Ürün ve süreç standartlarının birbiriyle yakın ilişkisi vardır. Ürün standartları çıktılara, süreç standartları süreç aktivitelerine uygulanmaktadır.

Yazılım standartlarının önemli olmasının birkaç sebebi vardır:

- En iyiyi ya da en azından en uygun olanı sunar. Bu bilgi, büyük uğraşlı deneyimlerden, hatalardan toplanmaktadır. Bunları standartlara çevirmek geçmiş hataların yapılmasını engellemektedir.
- Kalite güvencenin uygulanacağı alana bir çerçeve sunmaktadır. En iyi pratikleri içermekte, kalite kontrol de bu standartların izlenmesini sağlamaktadır.
- Devam eden bir işin bir kişiden alınıp diğer kişiye verilmesi durumunda sürekliliğe yardım etmektedir. Standartlar, organizasyondaki tüm mühendislerin aynı pratikleri benimsemesini sağlar. Sonuç olarak, yeni işe başlarkenki öğrenme eforunu azaltır.

Yazılım mühendisliği proje standartlarının gelişimi zor ve zaman alıcıdır. US DoD, ANSI, NATO, IEEE standartların gelişiminde aktif rol almışlardır (Sommervile, 2000).

Birçok organizasyonda ayrı bir bölüm olarak kalite güvence ekibi vardır. Kalite güvence ekibi yazılım geliştiriciler tarafından bazen şüphe ile hatta düşman gözüyle bakılabilmektedirler. Aslında onlar geliştiricilere yardım için oradadırlar, polis kuvveti olarak değil (Britton ve Doake, 1993). Yazılım mühendisleri, standartları yazılım gelişiminin tekniksel aktiviteleri karşısında bürokratik ve ilgisiz olduğunu düşünmektedirler. Bu durum, özelikle can sıkıcı form doldurma ve iş kayıtlarında olasıdır. Bu problemlerden kaçınmak için standartları koyan kalite yöneticileri şu basamakları izlemelidir:

- Yazılım mühendislerini ürün standartlarının gelişimi aşamasına katmalıdır. Standartların arkasında yatan gerçekleri, motivasyonu anlamalı, bunu standartlara işlemelidir. Standart dokümanları sadece standartları göstermemeli, aynı zamanda standart kararlarının niçin alındığının akıllıca bir açıklamasını içermelidir.
- Değişen teknolojiye göre standartlar gözden geçirilmeli, değiştirilmelidir.

- Yazılım araçlarının mümkün oldukça standartları desteklemesi sağlanmalıdır. Bu şekilde, büyük uğraşlar gerektirmeden standartların gelişimi sağlanabilir (Sommervile, 2000).

Kalite güvenceye harcanan zaman ve efor sistemin doğasına göre değişmektedir. Sonuçları hayati önem taşıyan sistemler (hastane hasta takibi, uçak kontrol, nükleer alan kontrolü yazılımları,… gibi) daha sert kalite güvence prosedürleri gerektirmektedir (Britton ve Doake, 1993).

6.5.2. Kalite Planlama

Arzu edilen ürünün kalitesi tanımlanır. Bu yapılmazsa her mühendislik zıt bakış açısı kullanır ve farklı ürün özellikleri optimize edilir. Plan belli kalite kriterlerini içermelidir (Sommervile, 2000).

6.5.3. Kalite Kontrol

Kalite güvence standartlarının izlenip izlenmediği kontrol edilir. Üretimde bir takım amaçlara (ürünler) bir takım süreçleri uygulayarak ve bir takım kaynakları (girdiler) kullanarak ulaşılır. Üretim işinde ortaya çıkan ürünlerden bazıları bir takım süreçlerin uygulanmasında aynı zamanda kaynak görevi görürler. Bu etapta ürünlerin ve onları oluşturan süreçlerin kalitesini kontrol altında tutmak es geçilmemesi gereken çok önemli bir etkinlik konusudur. Süreç kalitesi olmadan ürün kalitesine ulaşmak sadece şans eseri olabilir (Kolarik, 1995).

Kalite gözden geçirmeleri, kalite kontrol için tamamlayıcı bir yaklaşım olarak düşünülebilir. Ürün bileşenlerinin veya dokümantasyonların gereksinimlerle, tasarım, kod dokümantasyonlarının bileşenleriyle uyuşmazlıklarının analizinin yapılmasını ve standartların izlenmesini sağlar.

Dokümanların, sürecin gözden geçirilmesi bir grup insan tarafından yapılır. Bu grup, standartların takip edildiğini, yazılım ve dokümanların bu standartları doğruladığını kontrol etmekle sorumludur. Standartlardan sapmalar not edilmekte ve proje yöneticisinin dikkatine sunulmaktadır (Sommervile, 2000).

Fakat günümüzde, tüm takım üyeleri kendi işleri üzerindeki kaliteden sorumlu olmalı ve her üye birbirlerinin işlerini olumlu, yapıcı yönde eleştirmesi için desteklenmelidir (Britton ve Doake, 1993).

6.5.4. Yazılım Kalitesinin Hesaplanması

Zaman ve maliyet sayısal değerlerdir, bu nedenle ölçülebilmektedirler; diğer yandan kalite çok boyutlu bir kavramdır ve tanımlanması, ölçülmesi zordur.

Yazılım kalite faktörleri “Yazılım Kalite Güvencesi El Kitabı (The Handbook of Software Quality Assurance, Prentice Hall, 1998)“ na göre 3 ana başlık altında gruplaşarak şu şekilde tanımlanmıştır:

Yazılım kalite faktörleri;

- **Tasarım kalitesi**

Doğruluk: Yazılımın belirtilmiş tanımlamaları ve gereksinimleri karşılaması.

Bakım kolaylığı: Belirli bir zaman içinde yazılımda hata bulma ve onarma için gereken çaba.

Doğrulanabilirlik: Tanımlamalar doğrultusunda yazılımın özelliklerini ve performansını onaylama için gereken çaba.

- **Performans kalitesi**

Etkinlik: Daha az bir sistem (donanım, işletim sistemi, iletişim,..) kaynağı ile yazılımın ne kadar daha fazlasını yapabildiği.

Bütünlük: Yazılım ve bilgilerine yetkisiz insanlarca ulaşımın ne derece engellenebildiği, yazılımın değişmeden, etkilenmeden amacı için kullanılabilirliği.

Güvenirlik: Belirlenen zaman diliminde yazılımın gereksinim duyulan işlevleri ne hassasiyet ile yerine getirebileceği.

Kullanılabilirlik: Yazılımı öğrenim ve kullanım kolaylığı.

Test Kolaylığı: Belirlenen fonksiyonları yerine getirip getirmediğini doğrulamak için programın test edilmesi için gereken çaba.

- **Adaptasyon kalitesi**

Genişletebilirlik: Var olan fonksiyonları büyüterek veya yeni fonksiyonlar ekleyerek yazılım yeteneğinin ve/veya performansının arttırılması için gereken çaba.

Esneklik: Değişen ihtiyaçlara göre yazılımın misyonunu, fonksiyonlarını veya datasını değiştirebilme kolaylığı.

Taşınabilirlik: Bir yazılımın belli bir yazılım/donanım ortamından diğerine taşınması için gereken çaba.

Tekrar Kullanılabilirlik: Yazılımın ve parçalarının başka bir yazılım sisteminde ve uygulamalarında ne derecede kullanılabileceği.

Birlikte Çalışabilirlik: Bir yazılım sisteminin diğerleri ile bağlantı sağlaması kolaylığı.

Çalışabilirlik: Aynı yazılım sistemindeki parçalar arasındaki haberleşme için gereken çaba (Ashrafi, 2003).

Geçmiş dönemlerde bir ürünün son kontrolünde aldığı onay, onun kalitesini belirlerken, yazılımın kod satırlarında program geliştiriciye yönelik olarak yer alan ve "*rem" diye anılan program açıklamalarının varlığı da yazılımın kaliteli olduğunun belirtisiydi. Ardından yazılımın bitiminde uygulanan inceleme (review), teftiş (inspection), gözlem (walkthrough), denetim (audit) diye tanımlanan değişik kalite çalışmaları geldi. Bugün ise yazılım sektörü, yazılımın her aşamasında kalite olgusunu tam anlamıyla gerçekleştirebilmek için değişik modelleri, metodolojileri, resmi ve gayri resmi standartları kullanmaya çalışıyor, bunların dünya çapında kullanılır olması için çaba veriyor (Özkan, 2001).

"Yazılım Süreç İyileştirme" metodolojisi de öne çıkan yaklaşımlar arasında gelmektedir. Burada şu noktanın da gözden kaçırılmaması gerekmektedir; bu yaklaşımlar pahalı ve sonuçlarını görmek için biraz zaman gerektirmektedirler.

Tüm yazılım geliştiren organizasyonlar için yazılım kalitesinin iyileşmesi için bir yol bulmak mümkün değildir. Bir organizasyonda kalite için temel hedef olarak düşünülen bir şey diğer organizasyon için önemli olmayabilir. Örneğin; kritik misyonlu yazılımlar için güvenlik en önemli faktör iken, farklı platformlar için üretilen yazılım için taşınabilirlik öncelikli faktör olabilir. Açıkça görülmektedir ki, bazı faktörlerin değerinin artması diğerlerini düşürmektedir (Ashrafi, 2003).

6.6. Ölçüm Yönetimi

Yazılım ölçüm yönetimi, yazılım metrikleri verimliliği ve kaliteyi iyileştirmek için faydalıdır. Buna rağmen, bu ölçümler proje ve yazılım takımı için kötüye de kullanılabilir. Bu nedenle ölçüm yönetimi için birkaç prensibi incelemek gereklidir:

- Her ölçümün amacı toplanmaya ve analiz edilmeye değer olması için tam olarak anlaşılmalı,
- Projede toplanan ölçümler projenin özelliğine uygun olmalı,
- Ölçüm sonuçları, öncelikle projenin başarı olasılığını artırmak için kullanılmalı,
- Hiçbir zaman takım veya kişileri değerlendirmek için kullanılmamalı,
- Yeni ölçümlerin tanıtımı için dataların kullanılmadan önce bir geçiş süresi olmalı (Kossiakoff, 2003).

7. YAZILIM SÜREÇ İYİLEŞTİRME

Süreç içindeki aktiviteler (gereksinim analizi, tasarım, ...) sistemin verimliliğini ve kalitesini iyileştirebilmektedir. Her aktiviteye bu şekilde bakılmıştır. Şimdi, süreç iyileştirmeye daha geniş açıdan bakılacaktır.

7.1. Süreç İyileştirme

Yazılım geliştirme krizi yıllardır önemli bir tartışma konusu olmuştur. Yazılım endüstrisinde çalışanlar proje iptali, bütçe aşımı, geç teslim gibi problemleri çözmek için birçok yol aramıştır. Günümüzde yazılım boyut, karmaşıklık olarak gitgide daha da büyümekte ve çözümler bulmak, kaliteye odaklanmak daha da önem kazanmaktadır (Ashrafi, 2003). Yazılımın modern iş dünyasında hatta günlük hayatta aldığı rolün etkisinin bunda payı büyüktür. Artan bir şekilde, yazılım endüstrisinde yazılım kalite problemlerine çözüm bulma konusunda alarmlar çalmaktadır.

Başarısızlıkla savaşmak için bir yaklaşım, organizasyonlara yeni tasarım metotları getirmek yolu ile tekniksel olmuştur. Metotsal yaklaşımlarla (CASE araçları, RAD,..) oran azaltılmaya çalışılsa da çok başarılı olunamamıştır (Jiang ve diğ., 2004). CASE araçları, RAD,… gibi birçok araç ve teknik gelişmelere karşı ürünler hala fonksiyonel, tekniksel ve güvenilirlik amaçlarında başarısızlığa uğramakta, planlanan zaman ve bütçe hesaplarının üstüne çıkmaktadırlar. Bu hızlı çözümlere karşı yazılım geliştirme topluluğu iyileşme için daha kapsamlı bir sistem arayışı içine girmiştir (Ashrafi, 2003). Kaliteyi iyileştirmek için uzun yıllardır çalışılmakta ve şimdi organizasyonlar, en temel problemlerden birinin yazılım süreçlerinin etkin yönetilmemesi olduğunun farkına varmışlardır.

Yazılım süreçlerinin etkin yönetimi için farklı yaklaşımlar geliştirilmiştir. Yazılım süreç iyileştirme (SPI - Software Process Improvement) 1990'ların şüphesiz en belirginiydi. Şimdi de en fazla kullanılanıdır (Niazi ve diğ., 2005).

Yazılım süreç iyileştirme; var olan prosesleri anlayıp, bu prosesleri ürün kalitesini geliştirmek, maliyeti ve süreyi azaltmak için değiştirmek anlamına gelmektedir. Ölçüm aktivitesi üzerine kuruludur. De Marco'nun "Ölçemediğiniz bir şeyi kontrol edemezsiniz." sözü durumu açıklamaktadır (Blum, 1992).

Ürün kalitesiyle sürecin kalitesi arasında güçlü bir ilişki vardır. Süreç iyileştirme metodolojisinin altında iyi tanımlanmış, dokümante edilmiş sürecin kaliteli ürünle sonuçlanacağı düşüncesi yatmaktadır (Ashrafi, 2003).

Süreç iyileştirme her organizasyona göre değişen bir olgudur. Belirli metotların, araçların, başka bir projede kullanılan modellerin uygulaması demek değildir (Sommervile, 2000).

Proseslere sürekli iyileştirme standartlarının uygulanması gittikçe daha da önem kazanmaktadır. ISO 9000, SEI CMM, CMMI, SPICE yazılım geliştirme süreci iyileştirilmesine rehberlik eden süreç odaklı metodolojilerdir. Bu uygulamalar yazılım geliştiriciye analizin, tasarımın, kodlamanın, testin ve dokümante etmenin nasıl olacağını anlatmaz, aksine standartlar sunarak yazılım sürecini değerlendirmeyi ve sürecin iyileşmesini sağlar (Ashrafi, 2003). Yazılım geliştiren organizasyonlarda bu standartlar bazı müşteriler tarafından aranmaktadır (Olson, 2001).

7.2. Süreç İyileştirme Süreci

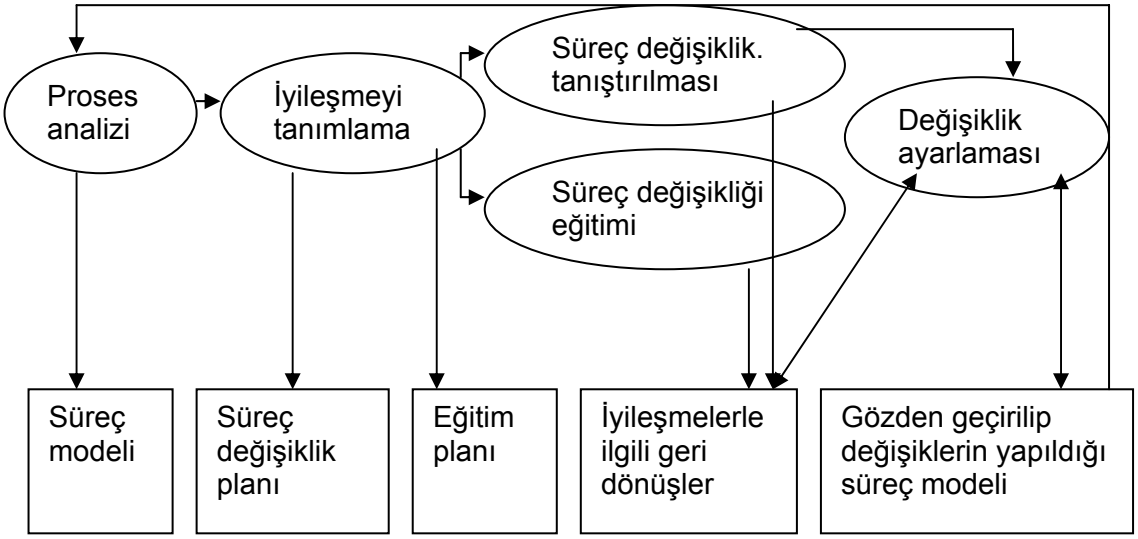
Proses analizi: Var olan prosesi incelemek, anlamak ve dokümante edebilmek için proses modeli üretmek gereklidir.

İyileşmeyi tanımlama: Analiz sonrasında darboğazlar belirlenir. Yeni prosedürler ve metotlarla bu darboğazları yok etme üzerine yoğunlaşılır.

Süreç değişikliklerinin tanıştırılması: Yeni prosedürler ve araçlar süreç aktivitelerine entegre edilir.

Süreç değişikliği eğitimi: Eğitim verilmeden tam bir fayda sağlanamaz.

Değişiklik ayarlaması: Değişikliklerin uygulanmasıyla verimliliğe bakılıp, sonuçlar incelenir (Sommervile, 2000).



Şekil 7.1 : Süreç İyileştirme Süreci

7.3. Süreç İyileştirmenin Kalite ve Verimlilik Üzerindeki Etkileri

Yazılım ürünlerinde 4 faktör ürün kalitesini etkilemektedir:

- Süreç kalitesi
- Kullanılan teknoloji
- İnsan kalitesi
- Maliyet, zaman ve çizelgeleme

Büyük projelerdeki önemli problemlerin başında entegre sorunu, proje yönetimi ve iletişimi gelmektedir. Küçük projelerde ise takımın kalitesi sürecin kalitesinden daha önemlidir. Takım üyeleri yeteneksiz ve deneyimsiz ise prosesin kalitesi ürünün kalitesiz olmasını engelleyemeyecektir. Yetenek ve deneyim varsa ürünün kalitesi de yüksek olacaktır.

Takım küçük ise kullanılan teknoloji özellikle önemlidir. Küçük takımlarda mühendisler çoğu zamanını tasarıma ve programlamaya vermektedirler. Bu nedenle iyi araçlar (tools) onların verimliliğini etkilemektedir.

Büyük projelerde; temel (basic) seviyede teknoloji yeterlidir. Komplike, gelişmiş araçlar daha az önemlidir. Takım üyeleri geliştirme aktivitelerine daha az zaman harcamakta, iletişime ve sistemin alt parçalarını anlamaya daha fazla zaman harcamaktadırlar. Verimliliklerini etkileyen faktör de budur. Araçlar (tools) fazla bir fark yaratmamaktadırlar.

Aslında, yazılım kalite problemlerinin gerçek nedeni kötü yönetim değil, yetersiz süreçler ve kalite eğitiminin kötü olmasıdır.

Organizasyonlar ayakta kalabilmek için yarışmak durumundadırlar. Birçok yazılım firması projeyi alabilmek için maliyetin altında (under-budgeted) fiyat sunmaktadır. Kazanmak için fiyatlandırma (pricing to win) rekabetçi sistemin kaçınılmaz sonucudur. Bu sistemde yazılımın kalitesini kontrol etmenin ne kadar güç olduğu da sürpriz değildir (Sommervile, 2000).

8. YAZILIM SÜREÇ İYİLEŞTİRME MODELLERİ

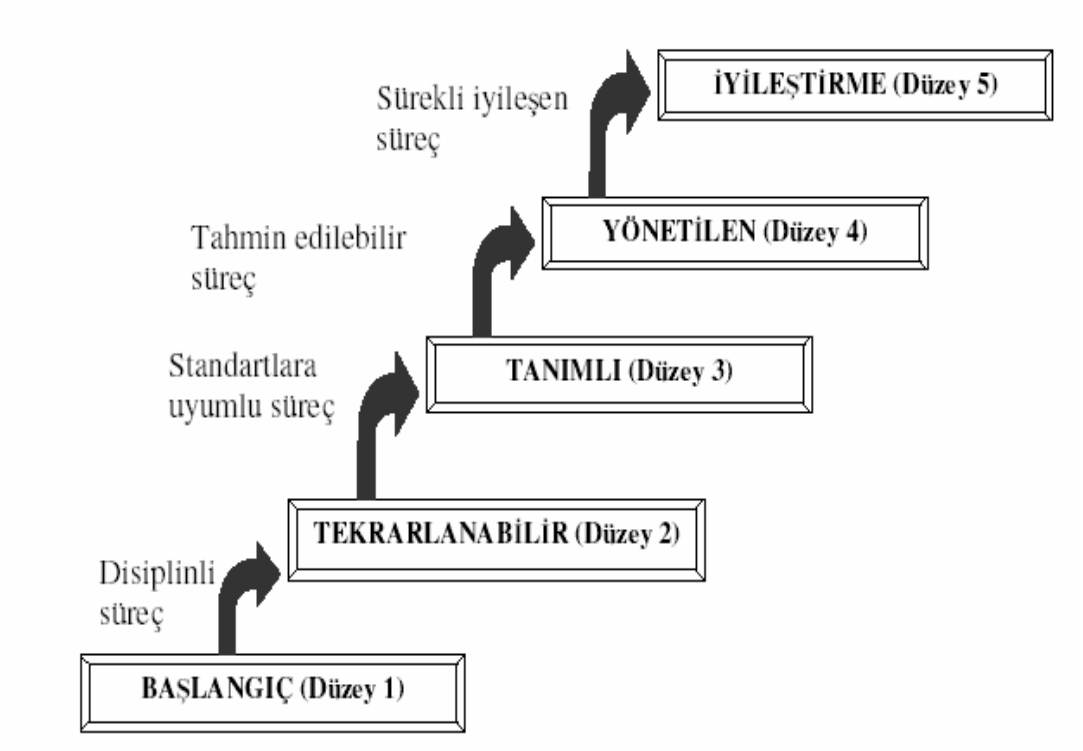
Yazılım geliştirme projelerinin yönetim tarafı çoğunlukla yetersiz planlama, geliştirme proseslerinin tam anlaşılmaması, iyi bir yönetim çerçevesinin olmayışı gibi problemlerle karşı karşıyadır (Jiang ve diğ., 2004). Bu çerçevede daha disiplinli geliştirme süreçleri için standartlar geliştirilmeye başlanmıştır. Bu modellerden başlıcaları CMM, CMMI, ISO 9000, SPICE dır.

8.1. CMM (Yetenek Olgunluk Modeli)

1990'larda organizasyonlara daha disiplinli geliştirme süreçleri için yönetim pratiklerine eklemeler yapıldı. SEI (Yazılım Mühendisliği Enstitüsü), DoD (ABD Savunma Departmanı) için yazılım süreç iyileştirme (SPI) anahtarı sundu. Daha sonra bu aktiviteler CMM adı altında toplandı (Jiang ve diğ., 2004).

Tanım ve anketler doğrultusunda etkili yazılım sürecinin ana elemanlarını yazılıma uyarlayan CMM'in, ilk sürümü 1991 yılında sunulmuştur. Güncel ve son sürümü ise 1993 yılına aittir. CMM, yazılım geliştirme alanında çalışan kuruluşların iş süreçleri olgunluğunu belirlemeyi ve iyileştirmeyi amaçlar. Burada "olgunluk" sözcüğü, "Bu kez ne kadar iyi yapılacak?" sorusunun daha az sorulduğu ve bilinmeyenin (riskin) düşük olduğu bir çevrenin oluşturulması anlamında kullanılmıştır. Süreçlerde olgunluk düzeyinin artması, belirsizliklerin azalması anlamına gelmektedir (Arifoğlu ve Gür,2005).

Bu model hayli vakit almakta, uygulama için çaba ve ayrıca organizasyonun kültüründe, davranışında büyük değişiklikler gerektirmektedir (Jiang ve diğ., 2004).



Şekil 8.1 : CMM süreç olgunluk düzeyleri

CMM yazılım geliştirmedeki zorlukları yönetimsel kontrol perspektifinden kırmak için yönetim proseslerini 5 olgunluk seviyesinde tanımlar (Jiang ve diğ., 2004) . Her seviyede süreç yetenekleri tanımlanmıştır.

Düzeyden düzeye geçişte, olgunlaşmamış, gelişigüzel, süreçlerin doğru düzgün belirlenmediği ve yazılım gelişimi için sabit yöntemlerin oluşturulmadığı düzeyden (Düzey 1) olgunlaşmış, yordamların yazılı hale getirildiği, izlendiği ve belirlenmiş kriterlerle kalitenin ölçümlendirildiği ve sürekli iyileştirmelerin yapıldığı düzeye doğru yol alınır.

Düzeyler şu şekildedir:

Düzey 1 - Başlangıç Düzeyi: Efektif prosedürler ve proje planı yönetimi yoktur. Başarının sadece bireylere bağlı olduğu seviyedir.

Düzey 2 - Tekrarlanabilir Düzey: Yazılı olmayan ve kısmen tutarlı süreçlerin olduğu seviye (proje planlaması, proje takibi, taşeron yönetimi, kalite güvencesi, konfigürasyon yönetimi). CMM felsefesine ilk adım bu seviyede atılır.

Düzey 3 – Tanımlı Düzey: Prosesler tanımlanmıştır. Şirket kültürü yazılı hale getirilmiştir (süreç odaklaması, süreç tanımı, eğitim programı, bütünleşik yazılım yönetimi, yazılım ürün mühendisliği, gruplar arası koordinasyon, ayrıntılı

değerlendirme). Proje yapısı ve süreçler organizasyon içinde kurumsallaşmış bir yapı oluştururlar.

Düzey 4 – Yönetilen Düzey: Performans ölçümleri yapılır. Tanımlı hale gelen süreçlerin artık ölçülebildiği, performans göstergelerinin değerlendirilebildiği seviye (nicel süreç yönetimi, yazılım kalite yönetimi).

Düzey 5 – İyileştirme Düzeyi: Kurumsallaşmanın gerçekleşip, geri beslemelerin sistematik bir şekilde değerlendirilmeye başlandığı seviye (hata önleme teknoloji değişim yönetimi, süreç değişim yönetimi). Bu seviyede, tam olgunluğa erişilir (Karadağ, 2002).

Bu beş geliştirme seviyesi, organizasyonun yazılım süreçlerinin gelişmişliğini ölçen ve işlem yeteneklerini değerlendiren bir yapıya sahiptir. Seviyeler, bir organizasyonun geliştirme çabalarının önceliklerinin belirlenmesini sağlar.

Organizasyon. yazılım süreci iyileşmesi için hiçbir çalışma yapmıyorsa seviye 1'e girer.

1 den 2 ye yükselmek için yönetim aktivitelerine temel disiplinler işlenmelidir.

2 den 3 e çıkmak için proseslerde gereksinim duyulan yeterlilik belirlenmeli ve bu aktivitelere katılan insanların bu yeterlilik verileri olduğuna emin olunmalıdır.

3 den 4 e çıkmak için sayısal hesaplamalar yapılmalı, proje ölçümleri, analizleri gerçekleştirilmelidir.

4 den 5 e çıkmak için sürekli iyileşme gereklidir (Olson, 2001).

Yapılan bir çalışma, 1 seviye atlamak için ortalama 21-37 ay gerektiğini göstermiştir (Jiang ve diğ., 2004) .

CMM, bir organizasyonun şu anda nerede, nerede olmak istiyor ve oraya geldiğini nasıl bilecek sorularını sorarak hangi olgunluk düzeyinde olduğunu ve bir sonraki düzeye geçmesi için hangi adımları takip etmesi gerektiğini belirler.

2003 yılının Ocak ayında Gartner Inc. tarafından yayınlanan rapora göre, CMM öngören değil tanımlayan bir standarttır, yani model, yazılımı geliştirirken *nasıl* değil, *ne* yapılması gerektiğini söyleyen bir araç olarak ele alınmalıdır.

SEI tarafından 13 organizasyonda süreç iyileştirme üzerine yapılan vaka analizinde, CMM'in uygulanması sonucunda, hataların önceden fark edilmesinde %6-%25 iyileşme; üretimde yıllık %9-%67 artış; teslim tarihinde %15-%23 oranında azalma; üretim sonrası hatalarda %10-%94 azalma ve iyileştirme programının yatırım getirisinin %400-%880 arasında olduğu saptanmıştır (Arifoğlu ve Gür, 2005).

Dünyada yaklaşık 120 kadar kurum (bunlardan yaklaşık 100 kadarı Hindistan'da faaliyet göstermektedir.) Düzey 5'te bulunmaktadır (Jiang ve diğ., 2004). SEI'nin tahminlerine göre bilgi teknolojisi organizasyonlarının %73'ü birinci düzeydedir.

Tüm bunların yanı sıra, problemler (yüksek seviyelerde) SEI modelinin kullanışsızlığını inkâr edemez. Birçok organizasyon alt seviyedir. CMM ile ilgili 3 problem vardır:

1. Model proje yönetimine ürün gelişiminden daha fazla odaklanır. Teknoloji kullanımını (formal, yapısal metotlar, araçlar) hesaba katmaz.
2. Risk analizini (problemleri önceden görmemizi sağlar) hariç tutar.
3. Tüm organizasyonlar için uygun olmadığı halde uygulama alanı tanımlanmamıştır.

SEI model; savunma modelleri için geliştirilmiştir. Bunlar büyük ve uzun ömürlü, komplike ve büyük takımlarla oluşturulmaktadır. Bu nedenle CMM önemli fakat tüm yazılım süreçlerinde kesin CM modeli olarak alınmamalıdır (Sommervile, 2000).

8.1.1. CMM Aktiviteleri ile Proje Performansı Arasındaki İlişki

Proje performansı, verimlilik (belirlenen zaman ve maliyete bağlı olarak kalitesi), etkinlik (uygulanabilirliği, adapte edilebilirliği) ve organizasyon kontrol, iletişim bilgilerini içerir.

Performans; ürün ve süreç olarak 2 grupta incelenebilir. Ürün yüksek kalitede olabilir fakat proje zaman ve maliyeti aşmış olabilir. Maliyet ve zaman yönünden iyi yönetilmiş projeler de kötü ürünler ortaya çıkarabilir.

IEEE Bilgisayar Derneği'ne bağlı 154 deneyimli proje geliştiricisi üzerinde yapılan bir çalışma, yazılım süreç yönetimi olgunluk seviyesinin proje performansı ile pozitif ilişkisi olduğunu göstermiştir.

Nidumolu modelinde tanımlanan performans kriterleri göz önüne alınarak bu çalışma yapılmıştır ve bulunan sonuçlar diğer raporlarla tutarlı çıkmıştır.

Sonuçlar, proje performansının en çok süreç mühendisliği ve CMM'in organizasyonel destek aktiviteleriyle (CMM seviye 3) ilişkisi olduğu, fakat ürün ve süreç kalite aktiviteleri (CMM seviye 4) ile de pozitif bir ilişkisi olduğunu göstermiştir. Diğer yandan, temel proje yönetim süreci aktivitelerinin (seviye 1-2) belirgin, gözlemlenebilir faydalar sağlamadığı görülmüştür. Makalede çıkarılan sonuç ise, organizasyonlar, seviye 3 e kadar proje performansında fayda alamayacaklarının

farkında olmalıdırlar ve aktivitelerin planlanması ve uygulaması aşamalarında daha fazla dikkatli olmalıdırlar (Jiang ve diğ., 2004).

8.2. CMMI (Bütünleşik Yetenek Olgunluk Modeli)

Organizasyonların ihtiyaçlarını karşılamak için tek bir model bulmada zorlanmaya başladıkları, birden fazla model uygulamanın ekonomik olmaması, kısacası hangi süreç iyileştirme modelini kullanacakları konusunda sorunlar yaşamaya başlamalarının üzerine bütünleşik CMM (CMMI) modeli geliştirilmiştir (Arifoğlu ve Gür, 2005).

CMMI, üretim süreçlerini, yazılım mühendisliği, yazılım, tümleşik ürün süreç gelişimi ve yazılım edinimi olmak üzere dört boyutta ele almaktadır. Buna karşın CMM, süreç ve olgunluk düzeylerini tek boyutta incelemektedir. CMMI, değişik disiplinleri sistem analiz ve tasarımı, yazılım mühendisliği ve yönetim konularını da içerisine alarak bir araya getirmektedir. CMMI modeli, aşamalı ve sürekli olmak üzere iki sürüm olarak üretilmiştir. SEI'ye göre gelecekte CMMI, CMM modelinin yerini alacaktır (Kossiakoff, 2003).

8.3. ISO 9000 (Üretim ve Yönetim Prosesleri için Avrupa Standardı)

Tüm endüstri alanında kalite yönetim sisteminin gelişiminde kullanılan uluslararası standarttır. Organizasyonların ürettikleri ürünün kalitesinden ziyade organizasyonun proseslerine odaklanmaktadır. Model, prosesler doğru olursa kaliteli ürün çıkacaktır mantığına dayalıdır. Standartlar, proseslerin tanımlanıp dokümante edildikten sonra prosedürlerin uygulanması, ölçülmesi ve sürekli iyileşmesi aşamalarını gerektirmektedir (Olson, 2001).

8.4. ISO 9001

Bu standartların en genelidir. Kalite Sistemi Standardı, tasarım, geliştirme, kurma ve servis için bir model tanımlamaktadır (Gill, 2005). Destekleyici doküman (ISO 9000-3) yazılım gelişimi için ISO 9000 açıklamalarını yapar. Yazılım için kalite sistemlerinin uygulayıcıları ve denetimcileri için özel rehberlik bilgileri verir.

ISO 9001, kalite süreci için genel bir modeldir. Sürecin farklı yüzlerini tarif etmekte ve organizasyonda hangi standartların, prosedürlerin olması gerektiğini tanımlar. Herhangi bir endüstriye özel olmadığından, bunlar detaylı tanımlanmaz. Spesifik bir

organizasyon için uygun kalite süreçleri kalite el kitabı ile tanımlanır ve dokümante edilir (Sommervile, 2000) .

ISO 9001 standartları şu 20 gereksinimi içermektedir:

- İstatistiksel teknikler
- Yönetim sorumluluğu
- Kalite sistemi
- Sözleşme gözden geçirmeleri
- Tasarım kontrolü
- Doküman ve veri kontrolü
- Satın alma
- Tedarikçilerin kontrolü
- Ürün tanımlaması ve izlenebilirliği
- Süreç kontrolü
- Kontrol ve testler
- Teftiş, ölçüm ve test araçlarının kontrolü
- Teftiş ve test durumları
- Uygun olmayan ürünün kontrolü
- Doğrulayıcı ve engelleyici eylemler
- İşleme, depolama, paketleme, saklama ve teslim
- Kalite denetçilerin kontrolü
- Eğitim
- Servis

(Gill, 2005).

Paulk (1995), CMM ile ISO 9000'i karşılaştırmıştır. CMM'in her seviyesindeki anahtar süreçlerle ISO 9000 in kalite yönetimi için gerektirdiklerine bakmıştır. CMM, daha detaylı, daha emredici ve süreç iyileştirme için bir çerçeve sunmaktadır. Bu özellikler, ISO 9000'de yoktur. CMM seviye 2 veya 3'deki organizasyonlar, ISO 9000 kriterlerine uymaktadır. Yine de, ISO' nun soyut tanımlamalarından dolayı, bazı seviye 1'deki organizasyonlar, ISO 9000 standartlarını sağlayabilmektedir (Paulk, 1995).

8.5. ISO'nun SPICE Modeli (ISO 15504)

SPICE nedir? Yazılım Süreci İyileştirme ve Yeterlilik Belirleme (Software Process Improvement and Capability dEtermination). ISO/IEC JTC1/SC7 WG10 tarafından

1991'de çerçeve üst model oluşturma kararı alındı. Çalışmalar 1993'te başladı. İlk versiyon Haziran 1995'te çıktı.

SPICE, SEI modelinden daha esnektir. ISO standartları süreç iyileştirme çerçevesi amaçlıdır. Olgunluk düzeyleri içermekte, fakat ayrıca düzeylerdeki anahtar süreç alanlarını da tanımlamaktadır (müşteri-tedarikçi ilişkisi gibi). Olgunluk düzeyi arttıkça, bu anahtar süreçlerin performansı da düzelmektedir (Sommervile, 2000).

SPICE Kapsamı :

- Yazılım satın alma
- Yazılım geliştirme
- İşletim
- Bakım ve destek için

planlama, yönetim, icra, denetim ve iyileştirme aracıdır.

SPICE ilkeler

- Ortak, modeller üstü, uluslararası kapsamlı çerçeve
- Diğer modelleri destekleyen, uyum sağlayabilen
- Her organizasyona, endüstriye, duruma uyabilen
- Büyük küçük projelere, gruplara, firmalara uygun
- İyileşmeyi, gelişmeyi ölçebilen
- Nesnel, tutarlı ve tekrarlanabilir
- Sertifikasyon amacı taşımayan
- Bir süreç standardı, ürün standardı değil.

SPICE, iki boyutlu bir modeldir. Birinci boyutta süreçler, ikinci boyutta yetenek düzeyleri vardır. SPICE'ın amacı, süreç iyileştirme (içe dönük), yetenek belirleme (dışa veya içe dönük) dir.

Süreç Grupları

Müşteri-satıcı süreçleri

- CUS.1 Yazılım edin
- CUS.2 Müşteri ihtiyaçlarını yönet
- CUS.3 Yazılım sağla
- CUS.4 Yazılımı işlet
- CUS.5 Müşteri hizmeti sağla

Mühendislik süreçleri

- ENG.1 Sistem isterlerini ve tasarımını geliştir
- ENG.2 Yazılım isterlerini geliştir
- ENG.3 Yazılım tasarımını geliştir
- ENG.4 Yazılım tasarımını uygula
- ENG.5 Yazılımın entegrasyonu ve testini yap
- ENG.6 Sistem entegrasyonu ve testini yap
- ENG.7 Sistemin ve yazılımın bakımını yap

Yönetim süreçleri

- Man.1 Projeyi yönet
- Man.2 Kaliteyi yönet
- Man.3 Risk yönetimi yap
- Man.4 Taşeronları yönet

Destek süreçleri

- SUP.1 Dokümantasyon geliştir
- SUP.2 Konfigürasyon yönetimi yap
- SUP.3 Kalite güvencesi sağla
- SUP.4 İş ürünlerini doğrula (verify)
- SUP.5 İş ürünlerinin geçerliliğini sağla (validate)
- SUP.6 Ortak gözden geçirmeler yap
- SUP.7 Denetimler yap
- SUP.8 Problemleri çözümle

Örgüt süreçleri

- ORG.1 İşin mühendisliğini yap
- ORG.2 Süreci tanımla
- ORG.3 Süreci iyileştir
- ORG.4 Kaliteli iş gücü temin et
- ORG.5 Yazılım mühendisliği altyapısı sağla

Değerlendirme modelinde her sürecin eksiksiz tanımı şu bilgileri içerir:

- sürecin amacının tanımı (Referans Model ile aynı)
- amacın tanımını güçlendirici açıklayıcı notlar
- sürecin amacına ulaşması için gerekli aktiviteler ve görevler
- her süreç ile ilgili girdi/çıktı iş ürünleri
- her iş ürününün karakteristikleri

Yetenek Düzeyleri

0- Eksik (incomplete) düzey

1- Varolan (performed) düzey

YD 1 : 1.1 Süreci Yerine Getirme (process performance) niteliği

2- Yönetilen (managed) düzey

YD 2 : 2.1 Başarım Yönetim (performance management) niteliği

2.2 İş ürünü Yönetim (work product management) niteliği

3- Yerleşmiş (established) düzey

YD 3 : 3.1 Süreç Tanım (process definition) niteliği

3.2 Süreç Kaynak (process resource) niteliği

4- Kestirilebilir (predictable) düzey

YD 4 : 4.1 Süreç Ölçüm (process measurement) niteliği

4.2 Süreç Denetim (process control) niteliği

5- En iyileşen (optimizing) düzey

YD 5 : 5.1 Süreç Değişim (process change) niteliği

5.2 Sürekli İyileşme (continuous improvement) niteliği

SPICE, yetenek düzeyini daha ayrıntılı, tekrarlanabilir ve nesnel notlama amacıyla, alt düzeyde, **süreç nitelikleri** tanımlamıştır.

- Yetenek Düzeyi boyutunda temel pratikler ve dokuz süreç niteliği vardır.
- Her yeni nitelik yükselen yetenek düzeyini gösterir.
- Süreç Yeteneği Düzeyi başarılan niteliklerle belirlenir.

Süreç nitelikleri, süreç yeteneğinin ölçümünü veren ve bir başarı skalasında değerlendirilebilen, özelliklerdir. Her süreç niteliği, sürecin amacına ulaşması için, o sürecin etkinliğini iyileştirme ve yönetme yeteneğinin bir yönünü tanımlar. Bir yetenek seviyesi, bir süreci yerine getirme yeteneğinde önemli artış sağlayan, nitelikler kümesidir.

SPICE Belgeleri; Part 1 : Kavramlar ve Giriş, Part 2 : Referans Modeli (zorunlu model), Part 3 : Değerlendirme İsterleri, Part 4 : Değerlendirme Rehberi, Part 5 : Uyumlu Model, Part 6 : Değerlendiricilerin Nitelikleri Rehberi, Part 7 : Süreç İyileştirme Rehberi, Part 8 : Yetenek Belirleme Rehberi, Part 9 : Sözlük içerir (İnce, 1998).

8.6. Modellerin Karşılaştırılması (İnce, 1998)

Tablo 8.1 : Süreç iyileştirme modellerinin karşılaştırılması

	ISO 9001	**CMM**	**SPICE**
YAPI	2 düzey	5 düzey	2 boyut, çok düzey
ANA AMAÇ	sertifikasyon	yetenek belirleme ve sertifikasyon	iyileştirme ve yetenek belirleme
BELGELER	kısa soyut genel kökenli (ISO 9000)	uzun somut yazılıma özgü ABD bağımsız gelişme	uzun somut ve ayrıntılı yazılıma özgü ISO üst model
DEĞERLENDİRME	denetleme soyut kriter dış uzman kısa,öz hata arayan kanıt isteyen negatif tutum kapsamlı değerlendirme tüm organizasyon tek not alır	değerlendirme somut kriter iç - dış uzman uzun ayrıntılı kapsamlı değerlendirme tüm organizasyon için tek değerlendirme yapılır	değerlendirme somut kriter iç - dış uzman uzun ayrıntılı gerçek arayan gösterge isteyen pozitif tutum küçük çapta yada kapsamlı olabilir her süreç için, her grup için, her proje için ayrı yapılabilir
İYİLEŞTİRME AMAÇLI OLARAK	rehber değil doğrultucu (corrective) hareket	rehber olma niteliği var iyileşme yolu var	rehber olma niteliği var esnek iyileşme yolları, amaca göre iyileşme

8. 7. Yazılım İyileştirme Standartlarının Uygulanması

Yazılım süreç iyileştirme modelleri (CMM, CMMI, ISO SPICE) kaliteli yazılım için proseslere odaklanmaktadır. Araştırmalar, bu modeller üzerinde çaba harcamanın, yüksek kalitede yazılım üretmeye, maliyetleri ve zamanları azaltmaya yardım ettiğini göstermiştir. Buna rağmen, bu standartların nasıl uygulanacağına da önem verilmesi gerekmektedir. Etkin kullanmak başarıyı etkileyecektir.

Çalışmalar yazılım süreç iyileştirme SPI yöneticilerinin %67'sinin bu modellerden hangi aktivitelerin uygulanacağı hususunda değil bu modellerin nasıl uygulanacağı konusunda danışmanlık istemekte olduğunu göstermiştir (Niazi ve diğ., 2005).

Konunun önem arz etmesine rağmen, bu konuda deneysel araştırma az vardır. Deneysel çalışmalar, yazılım geliştirme sürecini pozitif ya da negatif etkileyen faktörleri araştırmışlardır.

85 UK kampanyası ile yapılmış bir anket çalışmasını Rainer ve Hall (2002), bu çalışmalardan birine örnek olarak verebiliriz. Bu çalışmada SPI uygulamasını etkileyen başarı faktörleri belirlenmiştir. Sonuçlar; 4 faktörün başarılı SPI uygulamasında büyük etkisi olduğunu göstermiştir. Bunlar; yeniden incelemeler (reviews), standart ve prosedürler, eğitim ve danışmanlık, deneyimli personel.

SPI literatürü, vaka çalışmaları, deney raporları, yüksek seviyede yazılım proses metinleri içermektedir. Çalışmaların çoğu SPI uygulamalarına ait gerçek deneyimleri anlatmakta ve SPI uygulamaları için yol gösterici tavsiyeler vermektedir. Literatürün tavsiyelerini doğrulamak için ampirik çalışmalar da yapılmıştır. SPI üzerinde olumlu etkisi olan faktörleri inceleyen 50 yayınlanmış deney raporu, makale incelenmiş ve;

- En çok vurgulanan faktörün %66 gibi bir oranla üst yönetimin taahhüdü, desteklemesi olduğu sonucuna varılmıştır, Bu, SPI' yı uygulayan kişilerin düşüncesinde desteğin hayati rolü olduğunu göstermektedir.
- Literatürde sık bahsedilen diğer faktör, personelin iyileştirme sürecine dâhil edilmesi (%51), eğitim ve danışmanlık (%49). Bu da uygulayıcıların, kendilerinin sürece dâhil edilmesinin, eğitim ve danışmanlığın başarılı bir SPI programının uygulanması için zorunlu olduğunu düşündüklerini göstermektedir.
- Sonuçlar, ayrıca personel zaman ve kaynaklarının ve süreç eylem takımlarının yaratılmasının önemli faktörler olduğunu göstermiştir.

- Bu çalışma, aynı zamanda incelenen literatür kaynaklarının 1/4 inin yeniden incelemeleri, deneyimli personeli, açık ve net SPI amaçlarını, sorumlulukları paylaşmayı kritik başarı faktörleri arasında gördükleri sonucunu vermiştir.

Makalede incelenen literatüre göre;

- Kaynak kısıdı,
- Zaman baskısı,
- Deneyimsiz personel

SPI programının uygulamasına zarar vermektedir. Uygulayıcılar, uygulama sırasında organizasyon politikasından kaçınmayı tercih etmektedirler.

Avustralya'da 20 kampanyada (3 tanesi CMM seviye-3, 2 tanesi CMM seviye-2, 1 tanesi CMM kullanma planı yapmakta, 3 tanesi ISO 9001 sertifikalı, 10 tanesi formal kalite güvence programları uygulamakta; çoğu 5 seneden fazla bir süredir SPI programı uygulamaktalar) yapılan ampirik çalışmada ise süreç iyileştirmede organizasyonel politikaların en kritik faktör olduğu görülmüştür. Formal metodolojilerin eksikliği ve bilinçsizlik ise literatürde olmamasına karşın bu ampirik çalışmada bulunan faktörlerdir. 2. kritik engel ise destek eksikliğidir. Kaynak eksikliği de bir diğer başarısızlık faktörüdür (Niazi ve diğ., 2005).

SPI uygulamaları, daha fazla iyileşmeye ve değerlendirilmeye ihtiyaç duymaktadır. Unutulmamalıdır ki, yazılım süreç iyileştirme organizasyonun boş zamanında yapılacak bir iş değildir, başlı başına bir projedir. Destekleyen vizyon ve liderlik oluşmadan süreç iyileştirme gerçekleşemeyecektir.

Yazılım geliştiricilerin deneyimleri, algıları SPI modellerinin uygulamalarının pozitif ya da negatif rollerini belirlediği gibi, dolaylı yoldan kaliteyi de etkileyecektir (Niazi ve diğ., 2005).

9. PROJE BAŞARISIZLIKLARI

Yazılım yeni bir konudur. 1950'ler 1960'larda "Yazılım Mühendisliği", programcılık olarak karşımıza çıkmaktaydı. 1968'lerde "Yazılım Krizi" ilk olarak belirdi. 1980'lerde yoğun kullanımlar başladı. 1990'lara kadar büyük projelerde başarı oranı çok düşüktü.

ABD'de 1970'lerde kamu tarafından sipariş yolu ile satın alınan yazılım ürünlerinin ancak %5'i kabul edilip kullanılabilmiştir. %40'ı hiç kullanılamamış, geri kalanı ise maliyet artışı, zaman artışı, performans eksikliği veya özelliklerden taviz verilerek kullanılabilmiştir (İnce, 1998).

Datamation (1994), yılında yaptığı araştırmada yazılım projelerinde sık sık yapılan hataları şu şekilde sıralamıştır:

- Projenin kuruluşun stratejik hedeflerine uymaması
- Yazılımın büyüklüğüne göre değil, yönetimin beklentilerine göre belirlenmiş gerçekçi olmayan son tarihler
- Yetersiz planlama ya da sabırsız üst-yönetim
- Kritik kadro değişiklikleri
- Talep edilen ürün özelliklerinde meydana gelen değişiklikler
- Geç kalmış projeye yeni eleman eklenmesiyle projelerin daha da gecikmesi
- İşini iyi yapan insanların ödüllendirilmeyişi/ayırt edilmeyişi
- Müşterinin ne istediğinin tam olarak anlaşılmadan işe başlanması
- Projenin gidişatını izleyecek bir sistemin olmayışı
- Proje sonrası görüşmelerin yapılmayışı ve deneyimlerden kazanç sağlanmayışı

"Standish Grup" unun 1994'ten bu yana yaptığı araştırmalar IT sektörü hakkında en fazla istatistiksel bilgi aktaran, hatta hükümet kararlarında değişikliklere neden olabilen araştırmalardır. 50,000'den fazla tamamlanmış projeyi temsil eden bu araştırmalar projelerin neden başarılı veya başarısız olduklarına dair 10 yıllık verileri

sunmaktadır ve yapılan bu araştırmalar, IT proje yönetiminde iyileşmeler olduğunu göstermektedir.

CHAOS sonuçları, proje istatistik sonuçlarında global bir bakış sağlamaktadır, fakat araştırmada US ve Avrupa ağırlıklı bir saha incelenmiştir. %58'lik dilimi US, %27'lik bir dilimi Avrupa, kalan %15'i ise diğer bölgeler temsil etmektedir. Bu firmaların %45'i "Fortune 1000" grubunda, diğer %35'i orta ölçekli, %20'si de küçük firmalardır.

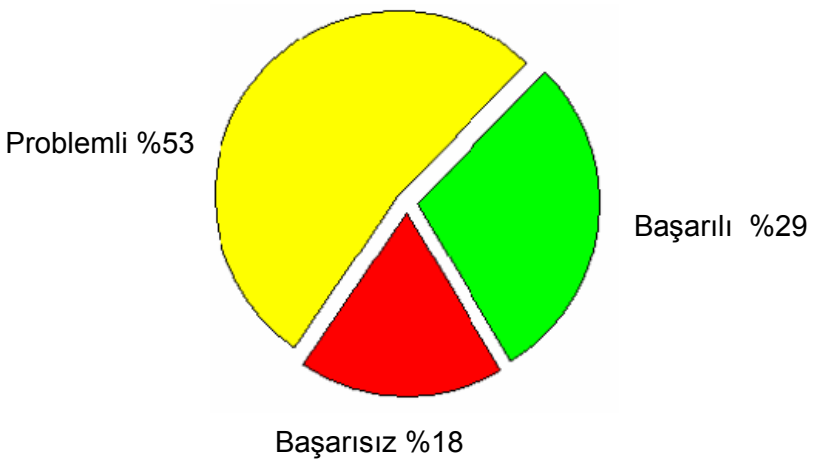
Araştırmalarda projeler 3 tipe ayrılmıştır:

Tip1: Başarılı projeler - yazılım projesinin tüm özellikleri ve fonksiyonları tanımlandığı şekilde, zamanında ve hesaplanan maliyette tamamlanmaktadır.

Tip2: Problemli projeler - proje tamamlanmakta, fakat hesaplanan zaman ve maliyetin üstünde ve belirlenen özelliklerden birkaçını sağlamayacak şekilde tamamlanmaktadır.

Tip3: İptal edilen projeler - yazılım projesi sürecin herhangi bir yerinde iptal edilmektedir.

2004'de Şekil 9.1'de gösterildiği gibi, proje başarı oranı %29, tamamlanmış fakat zaman, maliyet aşımına uğramış, gereksinimleri tam karşılamayan proje oranı %53, iptal edilen proje oranı %18 'dir (The Standish Group, 2004).



Şekil 9.1 : Proje başarı oranları (The Standish Group, 2004)

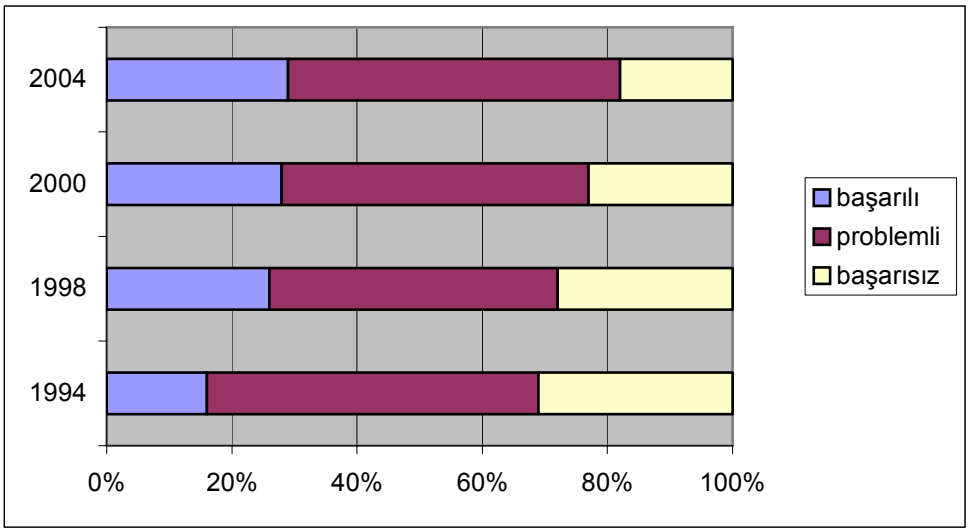
1994-2004 yılları arasındaki proje başarı oranlarına baktığımızda ise Tablo 9.1'deki gibi bir durum karşımıza çıkmaktadır:

Tablo 9.1 : 1994-2004 proje başarı oranları (The Standish Group,1994-2004)

	Başarılı	**Problemli**	**Başarısız**
1994	16%	53%	31%
1998	26%	46%	28%
2000	28%	49%	23%
2004	29%	53%	18%

Tablodan yıllara göre proje başarı oranlarına baktığımızda başarısız proje oranının giderek azalmakta, diğer yandan başarılı proje oranının artmakta olduğunu görüyoruz. Problemli proje oranında ise yıllar arasında fazla bir değişiklik görülmemektedir. Bu, başarısız proje oranındaki azalmanın başarılı proje oranına yansıdığını göstermektedir. Tablo 9.2'de bu durum grafikle gösterilmiştir.

Tablo 9.2: Proje başarı oranlarının yıllara göre dağılımı (The Standish Group,1994-2004)



2001 yılında yayınlanan sonuçlara göre proje başarısızlığında ilk sırayı yönetim desteğinin eksikliği almıştır. Sağlam iş vizyonlarına sahip kuvvetli bir proje kahramanı yoksa proje teknik ve politik uçurumlara sürüklenebilmektedir. Proje yöneticileri, müşteri hizmetlerini iyileştirerek, açık ve net iş planları ortaya koyarak, rekabetçi avantajları geliştirerek değer yaratmalıdır.

1994 yılında yapılan araştırmada, yazılım projelerinde 10 başarı kriteri belirlenmiştir. 2000 yılında bu başarı faktörleri güncellenerek yeni bir tablo oluşturulmuştur.1994 ve 2000 yıllarında yapılan araştırma sonuçları Tablo 9.3 ve Tablo 9.4'de gösterilmiştir. Hiçbir proje başarılı olmak için bu 10 faktörün tümüne gereksinim duymasa da, bu faktörlerden ne kadar çoğu proje stratejilerinde yer ederse, güvenilirlik de o kadar artacaktır (The Standish Group, 2004).

Tablo 9.3 : 10 başarı faktörü (Standish Group, 1994)

BAŞARI KRİTERLERİ (1994)	**PUAN**
Kullanıcı katılımı	19
Yönetim desteği	16
Gereksinimlerin açık ve net belirtilmesi	15
Doğru planlama	11
Gerçekçi beklentiler	10
Küçük kilometre taşları	9
Yetenekli çalışan	8
Ortaklık	6
Açık vizyon ve amaçlar	3
Çalışkan ve odaklanmış çalışan	3
TOPLAM	**100**

Tablo 9.4 : 10 başarı faktörü (Standish Group, 2000)

BAŞARI KRİTERLERİ (2000)	**PUAN**
Yönetim desteği	18
Kullanıcı katılımı	16
Deneyimli proje yöneticisi	14
Açık ve net iş hedefleri	12
Minimize edilmiş kapsam, alan	10
Standart yazılım alt yapısı	8
Sağlam, basit gereksinimler	6
Formal metodoloji	6
Gerçekçi hesaplamalar	5
Diğerleri	5
TOPLAM	**100**

Tablo 9.3 ve Tablo 9.4'de her faktör, proje başarısı üzerindeki etkisine göre ağırlıklandırılmıştır. Fazla puan, daha az proje riski demektir.

Harvard Business Okulu profesörlerinden Alan MacCormack ve arkadaşlarının yaptığı ve yaklaşık 2 yıl süren bir araştırmada da başarılı yazılım projelerinin şu 4 özelliğe sahip oldukları görülmüştür:

- Başarılı projeler özyinelemeli (iterative) şekilde yazılım geliştirirler. Yazılımın müşteri için anlamlı bir parçası erken bir yayımla teslim edilir ve yazılım teslimi diğer yinelemeler ile devam eder. Müşteriden yayımlar sonrası sürekli geri beslenim alınır. Yazılım bir evrim süreci sonucunda oluşur.

- Yapılan değişiklikler sonrası günlük tümleştirme yapılır. Tümleştirme sonucu yazılımın durumu hakkında bağlanım (regression) testleri sayesinde hızlı bir şekilde geri beslenim alınır.
- Yazılım geliştirme ekibi deneyimlidir.
- Projenin başından itibaren yazılım mimarisine ve sistemin birbirinden bağımsız bileşenlerden oluşturulmasına dikkat edilir (MacCormack, 2001).

Diğer yandan, John S. Reel'e (1999) göre yazılım projelerindeki iyileşme için gelişmiş teknolojiler, 5 esas faktörden daha az önemlidir. Bu faktörler:

1. Doğru adımla işe başlamak
2. Momenti sürdürmek
3. Gelişmeyi takip etmek
4. Akıllı kararlar almak
5. Proje bitimi sonrası analiz

1. Doğru adımla işe başlamak

Bu faktörler arasında en önemli olanı saptamak zor olsa da projeyi kurmak ve doğru adımlarla işe başlamak bu faktörler arasında öne geçecektir. Zayıf toprakta güçlü bitkiler yetiştirmek ne kadar zorsa, yanlış kurulan bir projeyi başarıyla geliştirmek de mümkün değildir.

Tom Field, yazılım geliştirmedeki tuzakları analiz etmiş ve proje başarısızlığı üzerinde 10 işarete dikkat çekmiştir. Bunlardan 7 tanesi tasarım geliştirilmeden veya kodlama başlamadan gelişmektedir. Bu nedenle yazılım gelişiminin kaderini belirleyen işlemlerin %70'inin yapı kurulmadan oluştuğu gözlenmektedir.

Proje başarısızlığına sebep olan 10 madde şu şekildedir:

1. Proje yöneticileri, kullanıcı ihtiyaçlarını anlamamaktadırlar.
2. Proje kapsamı kötü tanımlanmaktadır.
3. Proje değişiklikleri zayıf yönetilmektedir.
4. Seçilen teknoloji değişmektedir.
5. İş ihtiyaçları değişmektedir.
6. Proje bitiş tarihi gerçekçi değildir.
7. Kullanıcılar direnç göstermektedirler.

8. Sponsorlar çekilmektedirler.
9. Projede uygun nitelikli çalışanlar azdır.
10. Yöneticiler, en iyi uygulamalara ve alınması gereken derslere önem vermemektedirler (Field, 1997).

Bu verilen bilgilerden yola çıkılarak projeye nasıl başarılı bir başlangıç yapılır?

Herkes için gerçekçi amaçların ve beklentilerin oluşturulması: Müşteriye bu sistemin tüm problemleri çözemeyeceği, belki yeni sorunlar ortaya çıkaracağı hatırlatılmalıdır. Geliştirici, ayrıca müşterinin tam olarak ne istediğini, nasıl istediğini ve bu konuda nasıl yardım edebileceğini bilemediğini anlamalıdır. Hatta çoğu zaman müşteri, ne kadar harcayacağını da bilememektedir. İleride karşılaşılabilecek olumsuz durumlardan kaçınmak için her iki tarafın da son kararlarına çok sıkı bir şekilde dikkat edilmesi gerekmektedir.

Doğru takımın kurulması: İşi yapmak için yeterli kaynak sağlanmalıdır ve bilinmelidir ki düşünüldüğünden daha fazla kaynağa ihtiyaç olacaktır. Bu konuda fazla iyimser olunmamalıdır. Doğru takımı kurmak iyi kişileri seçmektir. Tüm tasarımcıların, geliştiricilerin en iyisi olması gerekli değildir. Deneyimler, takımın %20'sinin en iyilerden oluşmasının iş için en uygun olduğunu göstermektedir. Çok fazla stara sahip olmak da ego ve karışıklık yaratabilmektedir.

Aynı zamanda, mümkün oldukça gelişime kullanıcılar ve müşteri de dahil edilmelidir. Bu, iki taraf arasında güveni arttıracağı gibi, kullanıcı gereksinimlerini karşılayan bir ürün ortaya çıkarma şansını da arttıracaktır (Reel, 1999).

Kullanıcıların sisteme dahil edilmesinin 2 önemli sebebi vardır:

- Sistem analistleri sistemi ve problemleri tanımlayabilmek için kullanıcıların girdilerine ihtiyaç duyarlar.
- Proje, kullanıcılar sistemi kullanmazlarsa çalışmayacaktır.

Problemleri belirlerken katılırlarsa, önerilen çözümlerde danışmanlık verilirse projeyi kabul etme ihtimalleri artacaktır (Olson, 2001).

Takımın ihtiyaç duyduğu şeylerin temin edilmesi: Güçlü bir takım oluşturduktan sonra, sıra onlara verimliliklerini maksimize edecek, dikkatlerini dağıtmayacak bir ortam sunmaya gelir. Rahat bir ofis çok iyi sonuçlar almayı sağlayabilir.

Takım, araçlara da ihtiyaç duyar. Araçların güvenilir yerlerden alınması tavsiye edilir. Desteklenmeyen araçları kullanmaktan başka hiçbir şey, projeyi raylardan daha hızlı çıkartamaz. Takım, ayrıca bu araçlar üzerinde eğitime de ihtiyaç duyar. Dosyaları,

broşürleri bilgisizlik yüzünden kaybetmek ve deneyimsizlik, zahmetli ve maliyetlidir. Araçlar arasında; analiz, tasarım, konfigürasyon, test, doküman üretimi gibi araçlar yer almaktadır. Bu, paranın çoğunu araçlara yatırmak anlamına gelmemelidir. Dikkatli satın almaya, bir çok opsiyonu değerlendirmeye ve tüm takımı karar aşamasına dahil etmeye dikkat edilmesi gerekmektedir (Reel, 1999).

2. Momenti korumak

Takımın kurulup, uygun bir ortamın hazırlanmasından ve araçların sağlanmasından sonra diğer kritik faktör bu momenti devam ettirmek ve artırmaktır. Devam ettirebilmek için 3 anahtara odaklanılmalıdır:

- Yıpranmalar- küçük tutulmalı.
- Kalite- erken izlenmeli ve üstünlük beklentileri tespit edilmelidir.
- Yönetim- insandan çok ürün yönetilmelidir. İnsanlar rahat bırakılmalı, işlerini son dakika vermelerine izin verilmelidir. Daha sonra işleri kritik edilmelidir. Eğer ürün kabul edilebilir değil ise onların ürününü iyileştirmek için onlarla bireysel çalışılmalıdır. Amaç, kişiye ait konuları tüm takımın problemi yapmamaktır.

3. Gelişmeyi takip etmek

Yazılım gelişimi yönetiminin en büyük problemi çizelgede nerede olduğunu tam bilememektir. Çizelgeye göre nerede olunduğu bilinmezse ayarlama yapılamamaktadır. İlerlemeyi izlemek için bir çok metodoloji geliştirilmiştir, bunlardan biri seçilmeli ve düzenli bir şekilde kullanılmalıdır.

4. Akıllı kararlar almak

Akıllı kararlar almak, çoğunlukla başarılı proje liderlerini başarısızlardan ayırmıştır. Bir kararı almadan onun kötü bir karar olacağını teşhis etmek zor olmamalıdır.

Doğru kararlar almak için, müşteri gereksinimlerini müzakere etmek gibi birçok fırsat olacaktır. Çoğunlukla kullanıcılar, karmaşıklığı anlamadan marjinal değerli ihtiyaçlar istemektedirler. Onlara bu karmaşık ihtiyaçların kolları anlatılmalı ve gereksinim değişikliklerinin zaman ve maliyeti nasıl etkilediği anlatılmalıdır. Onlara yazılımcılara yardım etmeleri için yardım edilmelidir.

5. Proje bitimi sonrası analiz

Çok az şirket, hatalarından edinimler sağlamayı bir süreç olarak âdet haline getirmiştir. Şirketler proje sırasında iyi ya da kötü olan kavramları anlamaya zaman ayırmazlarsa onları tekrarlamaya mahkum olmaktadırlar.

Proje bitimi sonrası yapılan analizden neler öğrenilebilir?

İlk olarak, çizelge hedeflerinin neden tutturulamadığı öğrenilebilir. Bu faktörleri gelecek projede hesaba katmak, dramatik bir şekilde hesaplama tekniklerini iyileştirecektir. Bu analiz, ayrıca takımın ve şirketin yazılım sistemini nasıl geliştirdiklerine dair bir profil geliştirmeye yardım eder. Çoğu şirketin ve takımın gelişim döngüsünü güçlü etkileyen kişilikleri vardır. Bu analiz yapılarak, bu kişilikler olaylardan izole edilmiş bir şablon olarak ortaya çıkar.

Sistemi ölçmek, bu ölçümleri analiz etmek sistemin gelişimini iyileştirecektir. Şirketin yazılım geliştirme için kullandığı metotlar, alışkanlıklar bir sistem oluşturur. Montaj hattı sisteminden daha az tanımlanmıştır, fakat yine de bir sistemdir. Bu analiz, bize ileriki "ürün hattı" için sistemde değişiklik yapmaya müsaade etmektedir.

Bu 5 kritik faktör, tasarım, geliştirme metodolojilerini, kullanılan dilleri dikkate almasa da bu faktörler uygulanırsa, yazılımcılar zamanında, istenilen bütçede ve kullanıcı isteklerini karşılayan projelerini tamamlamada büyük adımlar atacaklardır (Reel, 1999).

10. İLERİYE BAKIŞ

Artan taleplere cevap vermek ve proje başarısızlık risklerini azaltmak için yazılım teknolojilerini iyileştirme üzerinde büyük bir baskı vardır. Bu yönde yapılan çalışmalara bakacak olursak;

10.1. Süreç İyileştirme

Yazılım süreç standartları yazılımda disiplini güçlendirmiştir. SEI tarafından yeni gelişmeler PSP (Personal Software Process) ve TSP (Team Software Process) kişi ve takım seviyesinde pratiklere yoğunlaşmaktadır. Çalışanların yeteneklerini ve yeterliliğini iyileştirmeyi amaçlamaktadırlar.

Büyük iyi tanımlanmış projelerde bu yaklaşımların başarısızlıkları azalttığı gözlenmiştir. Küçük projelerde ise (iyi tanımlanmamış gereksinimler) çevik (agile) metotlar tavsiye edilmektedir (Kossiakoff, 2003).

10.2. Programlama Araçları

Visual Basic gibi, bilgisayar destekli programlama araçları daha iyi otomatik hata kontrol etmek, veri tabanı desteği sağlamak, programlamayı daha hızlı ve hataya daha az eğilimli olmak için iyileşmelere devam etmektedir.

Form kontrolü, hata ayıklama ve diğer programlama dillerinin destek fonksiyonlarının entegrasyonu verimliliği ve doğruluğu iyileştirmeye devam etmektedir (Kossiakoff, 2003).

10.3. Entegre CASE Araçları

Gereksinimler ve konfigürasyon yönetim araçları büyük projelerin gelişimini, bakımını kolaylaştırmak için diğer fonksiyonlarla entegre edilmiştir. Bu araçların entegrasyonu program modüllerinin takip edilebilirliğini sağlamaktadır. Pahalı olsalar da, sonuçlarının verimliliği artırması, zaman ve maliyeti azaltması kullanımları için yeterlidir (Kossiakoff, 2003).

10.4. Yazılım Tekrar Kullanımı

Bir önceki projeden şimdiki veya bir sonraki proje için alabileceğimiz parçalar var mı?

Yazılım geliştirme maliyeti artmaktadır. Donanımda, ortak parçalarda standardizasyonun gelmesi ile maliyet giderek azalmaktadır. Aynı fayda, yazılım mühendisliğinde de "tekrar kullanılabilirlik (reuse)" ile sağlanabilmektedir. Bu nedenle, farklı bir modüle yazmaya gerçekten ihtiyaç olup olmadığı incelenmelidir (Pfleeger, 1991).

Tekrar kullanım sadece kodun tekrar kullanımını değil, tasarımın, test ortamının, dokümanların ve benzeri bilgilerin kullanımını kapsamaktadır. Yazılım, esnek bir ürün olma özelliği kullanılarak, önceden belli bir disiplin ile üretilirse, bir formdan diğerine geçme kolaylaşabilir. Bir başka deyişle yeniden kullanılabilirlik özelliği ile önceden belli bir gereksinime göre üretilmiş olan bir yazılım, farklı bir gereksinime göre yeniden düzenlenebilir (Özkan, 2001).

Tekrar kullanım, belirgin bir şekilde verimlilik iyileşme potansiyeli sunmaktadır. Ancak, yeni teknolojiler entegre edildikçe tekrar kullanım faydaları yavaşlayacaktır. Çünkü oluşturulan yeni, ilk projelerin bir şeyler çekeceği kütüphaneleri olmayacak, çok az tekrar kullanım olacaktır (Blum, 1992).

10.5. Yazılım Sistem Mühendisliği

Yazılım yoğun sistemlerdeki gelişmede belki de en belirgin ilerleme, sistem mühendisliği prensiplerinin ve metotlarının etkin bir şekilde yazılım sistem tasarım ve mühendisliğine uygulanması olmuştur (Kossiakoff, 2003).

Sistem mühendisliği; sistemi bir bütün olarak belirlemek, tasarlamak, uygulamak, geçerliliğini kılmak, bakımını yapmak aktivitelerinin toplamıdır. Sistem mühendisliği yalnızca yazılımla ilgili değil, yazılım, donanım, sistemin kullanıcı ve çevresiyle olan ilişkisi ile de ilgilidir. Sistem mühendisleri, sistemin sağladığı hizmetleri, içinde bulundukları kısıtları bilmek zorundadırlar. Yazılım mühendislerinin de sistem mühendisliğini anlamaya ihtiyaçları vardır. Çünkü yazılım mühendisliği problemlerinin çoğu sistem mühendisliği kararlarının sonucudur ve yazılımın sistem içinde önemli büyük bir rolü vardır.

Sistem mühendisliği prosesleri ile yazılım geliştirme süreçleri arasında önemli farklar vardır:

Sistem mühendisliği süreci, disiplinler arası bir süreçtir. Sistem mühendisliği, bir çok farklı mühendislik disiplinini kapsamaktadır. Farklı mühendisliklerce farklı terminolojinin kullanılması sonucu ortaya çıkan anlaşmazlıklar çok büyük olabilmektedir.

Sistem geliştirmede tekrarlı yapılan işlerin (rework) azaltılması söz konusudur. ATC (hava trafiği kontrolü – Air Traffic Control) sisteminde radarın yerleştirildikten sonra değişiklikler yapılması gibi sistem mühendisliği kararları çok pahalıdır. Sistem tasarımı için yapılacak tekrar işler neredeyse imkansızdır. Yazılımın sistemde bu kadar önemli olmasının nedeni, sistem gelişimi sırasında yeni ihtiyaçlara cevap verebilecek esnekliğe sahip olmasıdır.

Yazılımın esnek olmasından dolayı, beklenmeyen problemler çözülmek üzere yazılım mühendislerine bırakılır. Radarın yeri değiştirilemediğinde yazılımcı devreye girer. Bu da sistemde sağlaması zor yüksek dereceli işlemci gerektirir. Yazılım mühendisliği, donanım maliyeti artırmadan yazılımın yeteneğini artırmak durumuyla karşı karşıyadır. Birçok “yazılım hatası” doğal yazılım problemi sonucu değildir. Bunlar değişen sistem mühendisliği ihtiyaçlarını sisteme yerleştirmek için yazılım değiştirme çalışmalarının sonuçlarıdır. Buna iyi bir örnek olarak Denver havaalanı bagaj sistemini (1996) verebiliriz (Sommervile, 2000).

Hem yazılım hem sistem mühendisliğini ortak çerçevede ele alan CMMI’nın gelişimi bu bakışa yardım etmektedir. Kompleks yazılımlar için taleplerin devam etmesi ile sistem mühendisliği metotlarının yazılım gelişimine uygulanması hızlanacaktır (Kossiakoff, 2003).

11. UYGULAMA: TÜRKİYE'DE YAZILIM GELİŞTİREN ŞİRKETLERDEKİ YAZILIM GELİŞTİRME SÜRECİNİN GENEL DURUMUNU ANLAMAYA İLİŞKİN BİR ANKET ÇALIŞMASI

11.1. Araştırma Metodolojisi

Öncelikle araştırmada kullanılan tanımlamalardan söz etmekte fayda vardır. Ankette kullanılan, yazılım mühendisliği uygulamaları ile yazılım geliştirme sürecinde kullanılan metotlar, modeller, teknik ve araçlar kastedilmektedir. Metodoloji, sistemleri kurmak ve bakımını gerçekleştirmek için tamamlayıcı tekniklerin bir araya getirilme faaliyetler dizisi; model, geliştirme yolunu oluşturan teoriler, konseptler, stratejiler topluluğu; teknik, verilen aktiviteyi bitirmek için belirlenmiş faaliyetler topluluğu (data modelleme, veri akış diyagramları,…); araçlar, yazılım mühendislerine yazılım geliştirmede ve bakım esnasında yardım eden ürünler (CASE, test araçları,…) olarak ele alınmaktadır (Fletcher ve Hunt, 1993).

11.1.1. Örneklem Kurulumu

Türkiye'de yazılım geliştiren şirketlerin sayısını tam olarak belirlemek zordur. Ancak ülkemizdeki yazılım sektörünün boyutlarının küçük ve mali yönden sınırlı bir sektör olduğu bilinmektedir. Hazırlanan anket (bkz. Ek.A) 200 civarında şirkete mail yolu ile gönderilmiştir. Ayrıca, İTÜ ARI Teknokent Teknoloji Geliştirme Bölgesi dâhilinde bulunan yazılım geliştiren firmalara da anketler elle teslim edilmiştir. Sonuç olarak 35 şirket anketi cevaplamıştır. Örneklem profili, bulgular bölümünde ayrıca incelenmiştir.

11.1.2. Araştırmanın Amacı

Bu araştırmada şunlar amaçlanmaktadır:

- Organizasyonlarca hangi yazılım mühendisliği uygulamalarının benimsendiğini belirlemek,
- En çok kullanılan yazılım süreci modellerini, teknik ve araçlarını saptamak,
- Yazılım mühendisliği metotlarına ilişkin eğitim durumunu saptamak,
- Yazılım mühendisliği uygulamaları konusundaki yaygın anlayışları saptamak,

- Yazılım mühendisliği uygulamalarını etkileyen faktörler hakkındaki genel anlayışı belirlemek,
- Proje yönetim süreçleri (proje planlama, kalite, risk yönetimi, personel eğitimi/ ödül ceza/ iletişim-uyum-işbirliği/ hedeflerin belirsizliği ve ölçüm yönetimi aktiviteleri) ile ilgili genel durumu incelemek,
- Yazılım süreç iyileştirmesi çalışmalarını ve bu çalışmalar esnasında en sık rastlanan problemleri saptamak,
- Geliştirme sürecine ilişkin en sık rastlanan problemleri belirlemek,
- Ve son olarak yazılım projelerini başarı kriterlerine göre durumlarını incelemek.

Anket sonuçları ile yeteri kadar önem verilen alanların görülüp, iyileştirilmesi, geliştirilmesi ve kaynak ayrılması gereken alanların da belirlenebilmesi amaçlanmıştır.

11.1.3. Anket Tasarımı

Araştırma 54 kapalı uçlu soru, 2 tane de açık uçlu sorudan oluşmaktadır ve şu şekilde yapılandırılmıştır:

1. BÖLÜM: İşletmenin demografik özellikleri ile ilgili soruları içermektedir.

2. BÖLÜM: Yazılım geliştirme sürecinde kullanılan yazılım mühendisliği metodolojilerinin, modellerin, teknik ve araçların genel durumunu incelemeye ilişkin soruları içermektedir.

3. BÖLÜM: Proje yönetim süreçleri ile ilgili genel durumu incelemeye ilişkin soruları içermektedir.

4. BÖLÜM: Geliştirme sürecine ait genel soruları içermektedir.

5. BÖLÜM: Bu bölüm, yazılım geliştirme projelerinin başarı kriterlerini içermektedir. Bu durumların oluşma sıklığına göre şirket değerlendirilmektedir.

Anketin 2. bölümünde Poo ve Chung (1998) kaynağından yararlanılmıştır.

11.2. Bulgular

Anket sonuçları SPSS programı kullanılarak değerlendirilmiştir.

11.2.1. Demografik Veri

Araştırma çoğunluğu proje yöneticisi ve yardımcısı kişiler tarafından cevaplanmıştır. Araştırmaya katılan 35 kişiden 22 tanesi proje yöneticisi, lideri ve yazılım grubu başkanı (3'ü Pazarlama Müdürü) dır. Bu şekilde, yöneticiler %63'lük bir payı oluşturarak cevaplayanlar grubunda büyük çoğunluğu oluşturmaktadır. Anketi dolduranların 13 tanesi yazılım mühendisi ve uzmanıdır. Bu da %37'lik bir dilime tekabül etmektedir.

Araştırmaya katılan şirketlerden bilimsel ve mühendislik yazılımları (istatistik analiz paketleri,...) üreten şirket sayısı 9 (%25,7), mesleki yazılımlar (stok kontrol p., müşteri takip p., muhasebe p.,...) yapan şirket sayısı 28 (%80,0), görüntüsel yazılımlar (oyun ve animasyon yazılımları,..) yapan şirket sayısı 4 (%11,4), sistem yazılımları (derleyiciler, haberleşme programları, işletim sistemi,...) yapan şirket sayısı ise 7 (%20,0)'dir. Yüzdelik paylar 35 şirket üzerinden değerlendirilmiştir. Şirketlerden 12 tanesi birden fazla alanda yazılım geliştirmektedir.

Örneklemi oluşturan şirketlerin 28'i (%80,0) tamamen yerli, 5'i (%14,3) yabancı ortaklı , 2'si (%5,7) tamamen yabancıdır (bkz. Tablo A.1).

Araştırmaya katılan şirketlerin büyük bir çoğunluğunun çalışan sayısı 50'den daha azdır (%48,6). 50-100 arası çalışanı olan şirket yüzdesi de belirgin bir paya sahiptir (%34,3) (bkz. Tablo A.2).

Organizasyonların %31,4'ü son 5 yılda 1 ve 5 arasında yazılım üretmiştir. %25,7'si 6-10 arasında ve 20 den fazla, %8,6'sı ise 11-15 arasında ve 16-20 arasında yazılım üretmiştir (bkz. Tablo A.3).

Yeni uygulamalar geliştirmek ilk aktivite olarak gözlenirken, yapılan yazılımlara bakım yapmak ikinci sıradaki aktivite olarak gözlenmiştir. Son kullanıcılara operasyonel, teknik destek sağlamak ise yazılım aktiviteleri sıralamasında üçüncü sırada yer almıştır.

11.2.2. Yazılım Geliştirme Metodolojisi

Organizasyonların %62,9'u disiplinli, dokümante edilmiş (formal) sistem geliştirme metodolojileri kullanmaktadır. Formal metodoloji kullananların %40,9'u 1-2 yıldır, %27,3'ü 3-4 yıldır, %31,8'i 4 yıldan fazladır disiplinli metodoloji kullanmaktadır (bkz. Tablo A.4 ve Tablo A.5).

Şirket büyüklüğünün formal metodoloji kullanma ile ilişkisini test etmek için yapılan chi-square testi sonrasında şirket büyüklüğü 50'den daha az olan şirketlerle 50'den daha fazla olan şirketler arasında formal metodoloji kullanımı bakımından bir fark olmadığı gözlenmiştir (bkz. Tablo A.6).

Formal veya formal olmayan metodolojiyle en fazla artışlı model izlenmektedir. Artışlı modeli, şelale ve çevik modeller izlemektedir.

Tablo 11.7 : Formal veya formal olmayan metodolojiyle izlenilen yazılım süreci modeli

	Sıralama	Cevaplar	Sıklık	Pay (%) *
Formal veya formal olmayan metodolojiyle izlenilen yazılım süreci modeli	1	Artışlı (incremantal) modeller (aşama aşama teslim)	14	40,0
	2	Şelale modeli	8	22,9
	3	Çevik modeller (XP,FDD)	7	20,0
	4	Spiral model (şelale+risk analizi)	5	14,3
	5	RUP (Rasyonel Bütünleştirme Süreci Modeli)	4	11,4
	6	Diğer	4	11,4
	7	Cevaplamayan	6	17,1

* 35 anket = 100%

Bu konuya ilişkin soruyu cevaplamayanların oranının %17,1 gibi büyük bir sonuç çıkması sürpriz değildir. Bu soru, yazılım geliştirme metodolojileri konusunda biraz akademik bilgi gerektirmektedir ve anketi cevaplayan kişilerin tamamının IT disiplininden gelmeme ihtimali vardır.

11.2.3. Yazılım Mühendisliği Teknikleri

11.2.3.1. Gereksinim Analizi Safhası

Gereksinim analizi safhasında kullanılan teknikler kullanılma sırasına göre Tablo 11.8'deki gibi saptanmıştır:

Tablo 11.8 : İhtiyaç analizi safhasında kullanılan teknik ve araçlar

	Sıralama	Cevaplar	Sıklık	Pay (%) *
İhtiyaç analizi safhasında kullanılan teknik ve araçlar	1	Data akış diyagramları	22	62,9
	2	Nesne tabanlı analiz	17	48,6
	3	Hızlı prototip	15	42,9
	4	Varlık-Bağıntı diyagramları	12	34,3
	5	Veri standartlaştırılması	11	31,4
	6	UML	10	28,6
	7	Veri sözlülüğü	8	22,9
	8	Durum değişikliği diyagramı ve analizi	5	14,3
	9	Diğer	4	11,4

* 35 anket = 100%

Cevaplar, ankete katılan organizasyonların %62,9'unun data akış diyagramları kullandığını göstermektedir. Data akış diyagramları yapısal teknikler grubuna

girmektedir. Tabloda göze çarpan bir diğer nokta ise, ankete katılan organizasyonların yarıya yakının nesne yönelimli teknikleri kullanıyor olmasıdır. Nesne yönelimli teknikler, diğer geleneksel tekniklerle kıyaslandığında yazılım mühendisliğinin yeni kültürünü yansıtmaktadır.

Yapılan bu araştırmada , hızlı prototip, %42,9'lik bir pay alarak 3. sıraya yerleşmiştir. Yazılım geliştirme projelerinin ortak notasyonu olan UML modeli ise alt sıralarda yer almaktadır.

11.2.3.2. Tasarım Safhası

Tablo 11.9'a göre kullanılan 3 önemli tasarım tekniği; ekran tasarlayıcılar (%60,0), nesne tabanlı tasarım (%51,4) ve rapor tasarlayıcılar (%40,0) dır. Prototip tasarım da %34,3'lük bir pay almıştır. Diyalog akış diyagramları, karar ağaçları, işlem hacim analizi teknikleri ise %15-30 arasında bir değere sahiptir.

Tablo 11.9 : Tasarım safhasında kullanılan teknik ve araçlar

	Sıralama	Cevaplar	Sıklık	Pay (%) *
Tasarım safhasında kullanılan teknik ve araçlar	1	Ekran tasarlayıcılar	21	60,0
	2	Nesne tabanlı tasarım	18	51,4
	3	Rapor tasarlayıcılar	14	40,0
	4	Prototip tasarım	12	34,3
	5	Diyalog akış diyagramları	10	28,6
	6	Karar ağaçları	8	22,9
	7	İşlem hacim analizi	5	14,3
	8	Yapı çizelgeleri	3	8,6
	9	Diğer	3	8,6
	10	HIPO (hiyerarşik girdi-süreç-çıktı) çizelgeleri	1	2,9

* 35 anket = 100%

11.2.3.3. Kodlama Safhası

Tablo 11.10'a bakıldığında 4. kuşak dillerin %77,1'lik bir oranla en çok kullanılan diller olduğu görülmektedir. 3.kuşak diller de %22,9'luk bir paya sahiptir. 4.kuşak diller kapsamında RPG, Visual Basic, Oracle Forms, Access. Paradox, Foxpro..., 3.kuşak diller kapsamında ise Cobol, Fortran, Basic, Pascal, C,...dilleri yer almaktadır.

Tablo 11.10 : Kodlama safhasında kullanılan teknik ve araçlar

	Sıralama	Cevaplar	Sıklık	Pay (%) *
Kodlama safhasında kullanılan teknik ve araçlar	1	4.kuşak diller	27	77,1
	2	3.kuşak diller	8	22,9
	3	Diğer….	6	17,1

* 35 anket = 100%

11.2.3.4. Test safhası

Tablo 11.11'e göre kara kutu testi (gereksinme duyulan şeylere yanıt verip veremediği, işlevselliğinin sınandığı test), %65,7'lik bir oranla en fazla kullanılan test tekniği olmuştur. Bu tekniği sırayla yük, zorlanım, performans testi ve birim testi takip etmektedir.

Beyaz kutu testi (yazılım kodundaki deyim, koşulların,... sınanması), güvenlik testi ve değişiklerin entegresinden sonra yapılan bağlanım (regression) testi de %50'ye yakın bir kullanılma oranına sahiptir.

Beyaz kutu testinde, test senaryoları fonksiyonel spesifikasyonlardan ziyade program kodu temellidir. Beyaz kutu testinin farklı formları tüm durumların, dalların en az bir kere test edilmesini sağlar. Testin tüm formlarında durumları izlemek için bir araca ihtiyaç duyulur. Bu nedenle diğer test tekniklerine göre biraz karmaşıktır.

Tablo 11.11 : Test safhasında kullanılan teknik ve araçlar

	Sıralama	Cevaplar	Sıklık	Pay (%) *
Test safhasında kullanılan teknik ve araçlar	1	Kara kutu testi	23	65,7
	2	Yük,zorlanım,performans testi	20	57,1
	3	Birim testi	19	54,3
	4	Beyaz kutu testi	17	48,6
	5	Güvenlik testi	16	45,7
	6	Bağlanım (regression) testi	14	40,0
	7	Walkthrough (gözlem)	12	34,3
	8	Diğer...	2	5,7

* 35 anket = 100%

Test safhasında kullanılan tekniklerin kullanılma yüzdeleri diğer safhalarda kullanılan tekniklere göre daha fazladır.

11.2.3.5.Bakım Safhası

Yazılım sürecinin kontrol ve idaresini sağlamak amacıyla bir çok kuruluş değişiklik ve konfigürasyon yönetim sistemlerini kullanmaya başlamıştır. Konfigürasyon yönetimi, yürütülen işi güvence altına almakta, geliştirme ekiplerinin aynı sisteme ait alt bileşenleri geliştirme gayreti içerisinde birbirleri ile işbirliği içerisinde çalışabilmelerine olanak tanımakta, binlerce münferit dosyanın saklanması ve üstesinden gelinmesi gibi sıradan görevlerin otomatik olarak yapılmasını sağlamaktadır. Değişiklik yönetim sistemleri yazılım üzerinde yapılan değişikliklerin izlenmesini ve kontrol edilmesini sağlayarak faaliyetlerin izlenebilmesine ve ekip içerisinde yer alan herkesin bu değişikliklerin neden yapılmış olduğunu en ufak kod ayrıntısına varıncaya kadar bilebilmesine olanak tanımaktadır (Proya, 2003).

Bakım safhasında en çok kullanılan teknik, yazılım değişim yönetimi prosedürleri olarak karşımıza çıkmaktadır. Değişim yönetimi prosedürleri, konfigürasyon yönetimi aktiviteleri arasında gelmektedir. Bu oranın %50 civarında olması ve konfigürasyon yönetim prosedürleri kullanan organizasyonların da %37,1 gibi küçük bir orana sahip olması bu safhada teknik kullanımda yetersizlik olduğunu çok açık göstermektedir. Oysa ki, konfigürasyon yönetimi yazılım gelişim sürecindeki ve sonrasındaki en kritik aktivitelerden biridir.

Tablo 11.12 : Bakım safhasında kullanılan teknik ve araçlar

	Sıralama	Cevaplar	Sıklık	Pay (%) *
Bakım safhasında kullanılan teknik ve araçlar	1	Yazılım değişim yönetimi prosedürleri	18	51,4
	2	Konfigürasyon yönetim prosedürleri	13	37,1
	3	Diğer….	4	11,4

* 35 anket = 100%

11.2.4. Yazılım Mühendisliği Uygulamalarına İlişkin Eğitim Durumu

Organizasyonların %40'ı yazılım mühendisliği metotları üzerine düzenli eğitim verdiğini belirtmiştir (bkz. Tablo A.13).

Verilen eğitimler 35 anket üzerinden değerlendirildiğinde; formal olmayan eğitimin ağır bastığı görülmektedir. Cevap verenlerin %77,1'i iş üstünde eğitim, %68,6'sı kişilerin kendi çabalarıyla eğitim (video, bilgisayar,… kanalıyla) aldığını, %57,1'i şirket içi eğitimciler tarafından verilen eğitim, %40,0'ı şirket dışındaki eğitimciler tarafından verilen eğitim aldıklarını belirtmektedirler (bkz. Tablo A.14).

11.2.5.Yazılım Mühendisliği Uygulamaları Konusundaki Yaygın Anlayışlar

Bu safhada Likert ölçeği kullanılarak yazılımcıların yazılım mühendisliği uygulamalarına karşı tutumları ölçülmeye çalışılmıştır. Tek ana kütle aritmetik ortalama testi ile değerlendirme yapılmaya çalışılmıştır. Test değerlerine uygun değerler verilerek de, algılanan ortalama değerlerin istatistiksel olarak anlam ifade edip etmediği test edilmiştir.

Ankete katılanlardan anket üzerinde belirtilen faktörlere ne ölçüde katıldıklarına birer derece (1: Hiç katılmıyorum, 2: Katılmıyorum, 3: Ne/Ne, 4: Katılıyorum, 5: Tamamen katılıyorum) tayin etmeleri istenmiştir. Organizasyonlarda yazılım mühendisliği uygulamaları konusundaki yaygın anlayışlara karşı tutumlar, ortalama büyüklüklerine göre sırayla şu şekilde gözlenmiştir:

- Nesne tabanlı yazılım mühendisliği metotları faydalıdır.

Bulgular bu kriter için ortalama değerlendirmenin 4.14 , standart sapmanın 1.19 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- SEP, yazılım geliştirme ve bakım kalitesini iyileştirmektedir.

Bulgular bu kriter için ortalama değerlendirmenin 4.06 , standart sapmanın 1.11 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- SEP, yazılım geliştirme ve bakım sürelerini azaltmaktadır.

Bulgular bu kriter için ortalama değerlendirmenin 3.86 , standart sapmanın 1.22 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- SEP, yazılım geliştirme ve bakımı üzerindeki kontrolü güçlendirmektedir.

Bulgular bu kriter için ortalama değerlendirmenin 3.77 , standart sapmanın 1.33 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- Organizasyon tarafından adapte edilen standart metodolojilerin kullanılması kolaydır.

Bulgular bu kriter için ortalama değerlendirmenin 3.6 , standart sapmanın 1.36 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- Maliyetleri azaltmaya yardım etmektedir.

Bulgular bu kriter için ortalama değerlendirmenin 3.4 , standart sapmanın 1.93 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- SEP, organizasyonun rekabet avantajı kazanmasına yardım etmektedir.

Bulgular bu kriter için ortalama değerlendirmenin 3.29 , standart sapmanın 1.18 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

Bu faktörlerin ortalama değerleri ve standart sapmaları, ayrıca hangi sıklıkla “Katılmıyorum (1,2)”, “ Ne/Ne (3)”, “Katılıyorum (4,5) değerlerleri aldığı, Tablo A.16’ da verilmiştir (bkz. Tablo A.16). Ayrıca algılanan ortalama değerlerin istatistiksel

olarak bir anlam ifade edip etmediği Tablo A.17 ve Tablo A.18'de gösterilmektedir (bkz. Tablo A.17 ve Tablo A.18).

Bulgular, ankete katılanların yazılım mühendisliği uygulamaları hakkındaki "maliyetleri azaltmaya yardım etmesi" ve "organizasyonun rekabet avantajı kazanmasına yardım etmesi" faktörlerine katılma derecelerinin diğer faktörlere göre daha az olduğunu göstermektedir.

11.2.6. Yazılım Mühendisliği Uygulamalarının Başarılı Adaptasyonunu Engelleyen Etmenler

Bu bölümdeki değerlendirmede de bir önceki bölümde izlenen metot kullanılmıştır ve organizasyonlarda yazılım mühendisliği uygulamalarının başarılı adaptasyonunu engelleyen etmenlere karşı tutumlar ortalama büyüklüklerine göre sırayla şu şekilde gözlenmiştir:

- Yazılım mühendisliği metotları, araçları konusunda disiplinli eğitim eksikliği.

  Bulgular bu etmen için ortalama değerlendirmenin 3.94 , standart sapmanın 1.06 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- Yönetim desteğinin eksikliği.

  Bulgular bu etmen için ortalama değerlendirmenin 3.77 , standart sapmanın 1.22 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- Yazılım mühendisliği metotlarını, araçlarını efektif kullanacak deneyimli çalışanların azlığı.

  Bulgular bu etmen için ortalama değerlendirmenin 3.66 , standart sapmanın 1.45 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 4 olduğunu ortaya koymaktadır.

- Büyük finanssal yatırımlar.

  Bulgular bu etmen için ortalama değerlendirmenin 3.40 , standart sapmanın 1.31 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- Adapte etmek için amaçların net, açık olmaması.

Bulgular bu etmen için ortalama değerlendirmenin 3.31 standart sapmanın 1.37 unu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde3 olduğunu ortaya koymaktadır.

- Yazılım mühendisliği metotları ve araçlarının karmaşıklığı.

Bulgular bu etmen için ortalama değerlendirmenin 3.17 , standart sapmanın 1.04 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- Yazılım müh. araçlarının seçiminde, araçları kullanacak mühendislerin, teknik danışmanların etkin olmaması.

Bulgular bu etmen için ortalama değerlendirmenin 3.09 , standart sapmanın 1.25 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- Yazılım mühendisliği metotlarıyla geliştirilen yeni sistemlerin eski sistemlerle uyuşmaması.

Bulgular bu etmen için ortalama değerlendirmenin 3.06 , standart sapmanın 1.24 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- Yazılım müh. metotlarını destekleyecek uygun ortam ve araçların eksikliği.

Bulgular bu etmen için ortalama değerlendirmenin 3, standart sapmanın 1.237 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

- Günümüzdeki projeler için standart metodolojilerin uygun olmaması.

Bulgular bu etmen için ortalama değerlendirmenin 2,71 , standart sapmanın 1.45 olduğunu ve algılanan ortalama değerin istatistiksel olarak anlamlı bir şekilde 3 olduğunu ortaya koymaktadır.

Bu faktörlerin ortalama değerleri ve standart sapmaları, ayrıca hangi sıklıkla “Katılmıyorum (1,2)”, “ Ne/Ne (3)”, “Katılıyorum (4,5)” değerlerleri aldığı, Tablo A.19’da verilmiştir (bkz. Tablo A.19). Ayrıca algılanan ortalama değerlerin istatistiksel olarak bir anlam ifade edip etmediği Tablo A.20 ve Tablo A.21’de gösterilmektedir (bkz. Tablo A.20 ve Tablo A.21).

Yapılan araştırma sonucuna göre, “yazılım mühendisliği metotları, araçları konusunda disiplinli eğitim eksikliği”, “yönetim desteğinin eksikliği”, “ yazılım mühendisliği metotlarını ve araçlarını efektif kullanacak deneyimli çalışanların azlığı” etmenlerinin yazılım mühendisliği uygulamalarının başarılı adaptasyonunu engelleyen en büyük etmenler olduğu görülmektedir.

“Günümüzdeki projeler için standart metodolojilerin uygun olmaması” faktörüne katılmama oranının fazla olması sevindiricidir. Fakat yazılım mühendisliği uygulamaları seçiminde yalnızca bugünkü projelere bakılırsa, bu durum uzun dönem için garanti olmayabilir. Fletcher ve Hunt (1993), yazılım mühendisliği uygulamaları seçiminde “kristal topa uzun uzun bakılması” gerektiğini hatırlatmaktadırlar. Sonucun aynı kalması için organizasyonların önsezili ve 2-5 yıl sonra teknolojiyi nasıl kullanacaklarına karar vermiş olmaları gerektiğine ihtiyaçları olduğunu belirtmektedirler.

11.2.7. Proje Yönetim Süreçleri ile İlgili Genel Durum

Anketin bu bölümünde yazılım şirketlerindeki yönetim süreçleri ile ilgili genel durumu ölçmek amaçlanmıştır.

11.2.7.1. “Proje Planlama, Ara Hedef, İş Bölümü Belirleme” Aktivitelerinin Durumu

- Yetersiz ön planlama,
- Gerçekçi olmayan proje planı,
- Proje kapsamının küçümsenmesi,
- Yeterli sayıda ve sıklıkta ara hedeflerin belirlenmemesi

sıklıkla karşılaşılan problemler arasındadır. Bu problemler, çoğu zaman projeyi başarısızlığa sürükleyebilmektedir.

“Yapılan proje planları (proje kapsamının belirlenmesi, gerekli insan gücü ve kaynakların belirlenmesi) yeterli ve gerçekçidir”, “Yeterli sayıda ve sıklıkta ara hedefler belirlenmektedir” kriterlerinin Tablo 11.22’ye göre, algılanan ortalama değerleri 3,26 ve 3,46’dır. Bu değerler istatistiksel olarak anlamlı bir şekilde 3’dür (bkz. Tablo A.23).

Bu kriterlerin aldığı en sık değerler de olumlu yöndedir, fakat olumlu yöndeki sıklık %50 civarındadır. Bu değerler, organizasyonların planlama ile ilgili çalışmalarında noksanlıklar olduğunu göstermektedir.

Tablo 11.22 : Proje planlama, ara hedef, iş bölümü belirleme aktivitelerinin durumu

Kriterler	Cevap*	Yüzde**	Ortalama	St. sapma
Yapılan proje planları (proje kapsamının belirlenmesi, gerekli insan gücü ve kaynakların belirlenmesi) yeterli ve gerçekçidir	katılıyorum	45,7	3,26	1,172
	ne/ne	28,6		
	katılmıyorum	25,7		
Yeterli sayıda ve sıklıkta ara hedefler belirlenmektedir	katılıyorum	51,4	3,46	1,01
	ne/ne	31,4		
	katılmıyorum	17,2		

* Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

11.2.7.2. "Yazılım Kalite" Aktivitelerinin Durumu

Bu aşamada denetleme, organizasyonlardaki kalite kontrol mekanizmaları değerlendirilmektedir. Yazılım kalite aktiviteleri, sürecin başından sonuna kadar uygulanması gereken aktivitelerdir.

Ankete katılan organizasyonlardaki kalite aktiviteleri durumuna Tablo 11.24'den bakacak olursak;

Tablo 11.24 : Yazılım kalite aktivitelerinin durumu

Kriterler	Cevap*	Yüzde**	Ortalama	St. sapma
Gözden geçirme görüşmeleri etkin ve sistemli bir şekilde yapılmaktadır	katılıyorum	57,1	3,57	1,065
	ne/ne	25,7		
	katılmıyorum	17,2		
Yapılan işler takip edilmekte ve sonuçlar Denetlenmektedir	katılıyorum	68,5	3,91	1,095
	ne/ne	20		
	katılmıyorum	11,5		
Kalite aktiviteleri yerine getirilmektedir	katılıyorum	45,7	3,31	1,051
	ne/ne	34,3		
	katılmıyorum	20		

* Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

Analiz-tasarım süreçleri ve ara ürünlerinin kalitesiz oluşunun en temel sebebi bu ara ürün ve süreçlerin belirli standartlarının tanımlanmış olmaması ve bu süreçleri ve ara ürünleri denetleyecek bir pozisyonun ya da birimin bulunmayışından kaynaklanmaktadır. Bu da sorunların birikerek çığ etkisi yapmasına sebep olmaktadır.

"Gözden geçirme görüşmelerinin etkin ve sistemli bir şekilde yapılması", "yapılan işlerin takip edilmesi ve sonuçlarının denetlenmesi" için algılanan ortalama değer, istatistiksel olarak anlamlı bir şekilde 4'dür (bkz. Tablo A.25). Yapılan işlerin takip edilmesi, denetlenmesi faktörünün aldığı en sık değer olumlu yönde %68,5'dir.

Kriterler arasında, kalite aktivitelerinin (kalite güvence standartlarının belirlenmesi, kalite planının yapılması, ürün ve sürecin denetlenmesi, izlenmesi, sapmaların belirlenip dokümante edilmesi, düzeltmelerin yapıldığına emin olunması) yerine getirilmesine ilişkin genel durumu anlamaya ilişkin bir kriter de vardır. Bu kriterin aldığı en sık değer olumlu yönde, fakat ne yazık ki %45,7 gibi düşük bir orandır. Tüm bu kriterlere toplu bakıldığında kalite aktivitelerinin tam olarak yerine getirilmediği görülmektedir.

11.2.7.3. "Risk Yönetimi" Aktivitelerinin Durumu

Risk yönetimi aktivitelerinin durumuna baktığımızda ise planlama ve kalite aktivitelerine göre daha kötü bir tablo karşımıza çıkmaktadır. Problemlerin önceden saptanabilmesi ve proje üzerindeki etkilerinin minimize edilmesi için risk analizi, risk planlaması, risk izleme yapılması gerekmektedir.

Fakat Tablo 11.26'ya bakacak olursak problemlerin her zaman çoğunlukla önceden saptanamadığını, aynı şekilde riskten korunma, risk izleme ve yönetme planının sistemli bir şekilde yapılmadığını görüyoruz. Algılanan değerler istatistiksel olarak anlamlı bir şekilde 3 olmaktadır (bkz. Tablo A.23).

Tablo 11.26 : Risk yönetimi aktivitelerinin durumu

Kriterler	Cevap*	Yüzde**	Ortalama	St. sapma
Problemler önceden saptanabilmektedir	katılıyorum	45,7	3,26	0,95
	ne/ne	31,4		
	katılmıyorum	22,9		
Riskten korunma, riski izleme ve yönetme planı oluşturulmaktadır	katılıyorum	40	3	1,163
	ne/ne	20		
	katılmıyorum	40		

* Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

11.2.7.4. "Personel Eğitimi/ Ödül Ceza/ İletişim-Uyum-İşbirliği/ Hedeflerin Belirsizliği" Mekanizmalarının Genel Durumu

Tablo 11.27'yi incelersek;

"Yazılım geliştirme ekibinin deneyimli olması ve değilse eğitimden geçirilmesi" ne ilişkin durumun aldığı en sık değerin "katılıyorum/ tamamen katılıyorum" kategorisinde olması her ne kadar sevindirici de olsa, sıklık düzeyinin istenilen düzeyde olmaması üzücüdür. Yöneticiler, çalışanların yeteneklerini geliştirmek için gerekli eğitimi onlara sağlamalıdırlar.

Diğer yandan çalışanları mutlu edecek ödül ve takdir mekanizmaları ile çalışmayanları çalışmaya sevk edecek ceza mekanizmalarının tam anlamıyla gerçekleşmemesi motivasyon düşüklüğüne sebep olabilecektir. Çalışanların motivasyonu ne kadar yüksek ise performansları da o kadar yüksek olacaktır. Yöneticiler önce çalışanlarını motive etmeli, daha sonra bu motivasyondan yüksek performans beklemeli, aldığı performansa karşı da çalışanını tekrar motive edecek şekilde ödüllendirmelidir.

Organizasyonlarda uyum-iletişim-işbirliği-hedeflerin belirlenmesi durumuna bakıldığında en sık değerlerin olumlu yönde olduğu gözlenmektedir. Takım içindeki insanların iletişimi güçlü olmadan, gerçek bir uyum sağlanamamaktadır. Takım üyeleri arasında ortak bir dilin olması, ortak hedeflerin olması, başarı ve başarısızlıkların paylaşılması, bilgi paylaşımının olması sağlam bir takım için gerekli noktalardır. Günümüzde gelişmiş yazılım mühendisliği araçlarına rağmen, takım içi koordinasyon proje başarısını etkileyen en önemli faktörler arasında gelmektedir.

Tablo 11.27 : Personel eğitimi/ ödül ceza/ iletişim-uyum-işbirliği/ hedeflerin belirsizliği mekanizmalarının genel durumu

Kriterler	Cevap*	Yüzde**	Ortalama	St. Sapma
Yazılım geliştirme ekibi deneyimlidir, değilse eğitimden geçirilmektedir	Katılıyorum	62,8	3,77	1,003
	ne/ne	28,6		
	katılmıyorum	8,6		
İşini iyi yapanla yapmayanı ayırt edecek ceza/takdir mekanizmaları vardır	Katılıyorum	31,4	2,97	1,224
	ne/ne	40		
	katılmıyorum	28,6		
Takımlar arası/takım içi iletişimde problem yoktur	Katılıyorum	68,6	3,77	1,239
	ne/ne	11,4		
	Katılmıyorum	20		
Hedeflere ulaşmada uyum ve işbirliği sorunu yaşanmamaktadır	Katılıyorum	62,8	3,66	1,162
	ne/ne	20		
	Katılmıyorum	17,1		
Hedefler, yetki ve sorumluluklar tam olarak bellidir	Katılıyorum	60	3,54	1,172
	ne/ne	22,9		
	Katılmıyorum	17,2		

* Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

11.2.7.5. "Ölçüm Yönetimi" Aktivitelerinin Durumu

Tablo 11.28'e göre, yazılım geliştirme safhalarında bulunan hata sayıları ve tiplerinin kaydedilmesine ilişkin kriterin aldığı en sık değerler olumlu ve olumsuz yönde eşittir. Proje sonrası görüşmelerin yapılarak deneyimlerden kazanç sağlanmasında ise en sık değer olumlu yöndedir. Fakat, katılmayanların ve yarı

katılanların toplam sıklığı katılanların sıklığını geçmektedir. Algılanan 3 ortalama değerinin istatistiksel olarak bir anlam ifade ettiği tablo A.23'de gösterilmiştir (bkz. Tablo A.23).

Ölçüm yönetimi aktiviteleri, risk yönetimi aktiviteleri gibi organizasyonların en eksik kaldıkları aktiviteler arasına girmektedir.

Tablo 11.28 : Ölçüm Yönetimi aktivitelerinin durumu

Kriterler	Cevap*	Yüzde**	Ortalama	St. sapma
Yaz. Geliştirme safhalarında bulunan hata sayıları, tipleri kaydedilmektedir	Katılıyorum	45,7	3,26	1,54
	ne/ne	8,6		
	katılmıyorum	45,7		
Proje sonrası görüşmeler yapılarak deneyimlerden kazançlar sağlanmaktadır	katılıyorum	48,6	3,31	1,345
	ne/ne	25,7		
	katılmıyorum	25,7		

* Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

11.2.8. Yazılım Süreç İyileştirme Çalışmalarına İlişkin Durum

Yapılan araştırmada ankete katılan organizasyonlardan %62,9'unun süreç iyileştirme çalışması yaptığı, %37,1'inin ise herhangi bir iyileştirme çalışması yapmadığı gözlenmiştir (bkz. Tablo A.29).

Süreç iyileştirme çalışması yapan organizasyonların %36,4'ü ISO9001 ve diğer kalite güvence programları kullanırken, CMM'i %27,3'ü kullanmaktadır. Ayrıca, iyileştirme standartlarının arasında bulunan SPICE'ı kullanan organizasyona rastlanmamıştır (bkz. Tablo A.30).

İyileştirme programı kullananların %59,1'inin iyileştirme çalışmalarının henüz başlangıç aşamasında olduğu gözlenmiştir. Bu nedenle iyileştirme çalışmalarının başarılı ya da başarısızlığına dair bir değerlendirme yapamamışlardır. İyileştirme çalışmalarını değerlendirenlerin %4,6'sının iyileştirme programını başarısızlıkla sonuçlarken, %22,7'si başarılı olduğunu savunmaktadır. %13,6'sı da ne başarısız/ne başarılı kategorisinden yana cevap vermiştir. Süreç iyileştirme standartlarını kullananların büyük bir çoğunluğunun henüz başlangıç aşamasında olması organizasyonların süreç iyileştirme çalışmalarını uygulama konusunda biraz yavaş davrandıklarını göstermektedir (bkz. Tablo A.31).

11.2.9. Süreç İyileştirme Çalışmalarında Karşılaşılan Problemler

Anketi cevaplayanlara yazılım süreç iyileştirme çalışmalarında karşılaştıkları 3 temel problem sorulmuş ve şu şekilde cevaplar alınmıştır:

Tablo 11.32 : Değinilen problemler ve sıklıkları

Karşılaşılan problemler	Sıklık
Yönetimin desteklemesi	6
Zaman kısıdı	8
Yeterli mali kaynak ayrılamaması	5
İnsan kaynağı eksikliği	4
Yetkin ve deneyimli çalışanın bulunmaması	2
Mevcut işlerin yoğunluğu	2
Kültür	1
Gerekli uygun alt yapı ve bilgi eksikliği	2
Disiplin eksikliği	1
Mühendislik yaklaşımı eksikliği	1
Avantajların net ölçülememesi	1
Ölçümlerin yapılamaması	1
İyileştirme programı hakkında eğitim,anlayışın yaygınlaştırılması	2
Operasyonel süreçlerle entegrasyon	1
Tüm süreçlerin gözden geçirilmesi	1
Tüm dokümantasyonların güncellenmesi	1
Ekip çalışması	1
Personel desteği ve isteği	1

Bu maddelerden de anlaşılacağı üzere süreç iyileştirme çalışmalarındaki temel problem “süreç iyileştirmenin bir proje olarak kabul edilmemesi” olduğudur. İyileştirme çalışmalarında başarılı olunabilmesi için bu çalışmalar için yeterli kişi, zaman ve bütçe ayrılmalıdır. Süreç iyileştirme çalışmaları, yürüyen işlere göre ikinci plana bırakılırsa, bu çalışmalar hep sonraya ertelenecektir. Bu nedenle iyileştirme çalışmaları da bir proje olarak görülmeli, gerekli planlar yapılmalıdır.

Diğer önemli bir problem ise “yönetimin desteği” olarak karşımıza çıkmaktadır. Süreç iyileştirme çalışmalarının başarılı olabilmesi için yönetim, çalışmalara destek ve önem verdiğini göstermelidir. Yönetim tüm çalışma boyunca, gerekli durumlarda çalışmalarda bulunmalı, problemlere çözüm getirmelidir. Yönetimin desteği olmadan çalışmalar ilerlemeyecektir. Proje yönetiminde motivasyon ve ilham veren birisi yoksa ve proje, proje yönetimi tarafından beslenmez ise projenin başarılı olması çok zordur (Olson, 2001).

Organizasyonlarda gözlemlenen problemlerin arasında personelin desteği, gerekli uygun alt yapı ve bilgi eksikliği de gelmektedir. Organizasyonun tümünde, iyileştirme çalışmalarının tam olarak anlaşılması için farklı düzeylerde eğitimler verilmesi, kişilere amacın ne olduğu tam olarak anlatılması gerekmektedir. Amaçlar, yönetim

tarafından ölçülebilir şekilde ortaya konmalıdır. Çalışanlar, nelerin düzeleceğini, ne tür sorunların ortadan kaldırılacağını gördüklerinde çalışmalara daha istekle yaklaşacaklardır. Organizasyon kültürünün süreç iyileştirmeye yatkınlığı problemi de bu şekilde azaltıla bilinecektir.

11.2.10. Geliştirme Sürecine İlişkin Genel Durum

Anketin bu bölümünde cevap verenlerden yazılım geliştirme safhalarını, harcanan zaman miktarlarına göre numaralandırmaları istenmiştir. Sıralamanın sonucu şu şekildedir: Kodlamaya harcanan süre > Teste harcanan süre > Tasarıma harcanan süre > Gereksinim belirlenmesi ve analizine harcanan süre

Avrupa'da yapılan yazılım projeleri üzerine bir araştırmada Avrupa'daki yazılım kuruluşlarının proje zamanının %12'sini müşteri gereksinimine, %18'ini planlama ve spesifikasyonların belirlemesine, %20'sini tasarıma, %25'ini koda ve %25'ini teste aktardıkları gözlemlenmiştir. Durum Japonya ve Amerika için de benzerdir. Blackburn'ün araştırmasına göre yazılım projelerinde analiz ve planlama ile tasarım safhasına aktarılan zaman en az %50 arttıkça, kod yazma ve teste ayrılan zaman, yazılımda meydan çıkan hataların sayısı ile birlikte düşmektedir. Doğru tasarım ve analizle kod sadece mekanik bir işleme dönüşmekte ve unutulmamalıdır ki kod yazma doğru yönetilen bir proje söz konusu olduğunda yazılım sürecinin en kolay kısmı olmaktadır (Blackburn,1996).

11.2.11. Yazılım Geliştirme Sürecinde Karşılaşılan Problemler

Anketi cevaplayanlara yazılım geliştirme sürecinde karşılaştıkları 3 temel problem sorulmuş ve Tablo 11.33'deki gibi cevaplar alınmıştır:

Tablo 11.33 : Yazılım geliştirme sürecinde karşılaşılan problemler ve sıklıkları

	sıklık	**yüzde ***
kısıtlı kaynak		
Kısıtlı zaman, yeterli sürenin verilmemesi	11	42,31
Kısıtlı insan kaynağı	4	15,38
Maliyet-para kısıtı	4	15,38
	19	
analiz sürecinin kalitesi		
Analiz eksikliği, iyi yapılamaması, gereksinimlerin tam belirlenememesi	7	26,92
Gereksinimlerin değişken olması, sonradan değişmesi, alt yapı değişiklik istekleri	7	26,92
Gereksinim belirlenmesine yeterli zaman ayrılamaması	2	7,69
Talep edenin talebini net ifade edememesi	3	11,54
Müşterinin teknik yetersizliği	1	3,85
Talep edenin ifade edilmeyen iş kurallarını yazılımdan beklemesi	1	3,85
Analiz çalışması için müşteri tarafından yeterli zaman tanınmaması	1	3,85
Analiz yapabilecek deneyimli çalışan eksikliği	1	3,85
	23	

test sürecinin kalitesi	**sıklık**	**yüzde ***
Teste önem verilmemesi	1	3,85
Test yapacak çalışanın temin etme zorluğu	1	3,85
Test safhasının verimli geçirilememesi, yeterli verinin üretilememesi	3	11,54
	5	
tasarım sürecinin kalitesi		
Yanlış tasarım	2	7,69
	2	**7,69**
kodlama sürecinin kalitesi		
Yanlış kodlama	1	3,85
	1	**3,85**
Konfigürasyon yönetimi		
Dokümantasyon eksikliği, tam yapılamaması	3	11,54
Entegrasyonların sağlanma güçüğü	2	7,69
	5	**19,23**
planlama/ ara-hedeflerin belirlenmesi		
Eksik, yanlış planlama, değişikliklerin proje planına etkisinin tam olarak saptanması	3	11,54
Belirsizlik	1	3,85
	4	**15,38**
iletişim/çalışanların sorumsuzluğu/alışkanlıklar		
Ekipler arası koordinasyon	1	3,85
Yazılım uzmanının az sorumluluk alması veya almaması	1	3,85
Yazılımcıları eski alışkanlıklarından vazgeçirmek	1	3,85
Mühendislik yaklaşımı eksikliği	1	3,85
	4	**15,38**
Diğer		
Yönetim desteğinin eksikliği	1	3,85
Kalite kontrol	1	3,85
Objektif hata oranları ölçümlenmesi	1	3,85
Öngörülemeyen yazılımsal problemler	1	3,85
Teknik konu- teknoloji eksikliği	1	3,85
	5	**19,23**

cevapsız anket sayısı: 9
* 26 ankette problemler belirtilmiştir = 100%

Problem sıklığının en fazla analiz safhasında olduğu gözlenmiştir. Bunu kaynakların kısıtlı olması takip etmektedir ve diğer problemler sıklık yüzdeleriyle Tablo 11.33'de görülmektedir.

11.2.12. Yazılım Geliştirme Projelerinin Başarı Kriterleri

Bu bölüm, yazılım proje başarısının ölçümü ile ilgilidir. Planlanan zaman ve bütçe, müşteri memnuniyetini sağlama, bakım kolaylığı faktörlerine göre projelerin başarıları değerlendirilmeye çalışılmıştır.

Yazılım geliştirme projelerinin başarı kriterlerinin oluşma sıklığı Likert ölçeği ile ölçülmeye çalışılmıştır. Tablo 11.34'e göre "Geliştirilen yazılımların kullanıcıyı tatmin etmesi " kriteri en fazla ortalama değeri almıştır. Bu sonuç, organizasyonların bu amacı gerçekleştirmede çoğunlukla problemleri olmadığını göstermektedir. Bu kriteri sırayla "geliştirilen yazılımın bakımının kolay olması" , "projelerin zamanında teslim edilmesi" ve "hesaplanan maliyette tamamlanması" takip etmektedir.

Algılanan ortalama değerlerin istatistiksel olarak anlam ifade ettiği tablolarda gösterilmektedir (bkz. Tablo A.35 ve Tablo A.36).

Tablo 11.34 : Yazılım geliştirme projelerinin başarı kriterlerinin oluşma sıklığı

Başarı kriterleri	Cevap*	Yüzde**	Ortalama	St. sapma
Yazılım uygulamaları, kullanıcıyı	hiçbir zaman/ nadiren	2,9	3,89	0,832
yüksek derecede tatmin etmektedir	bazen	31,4		
	çoğunlukla	40		
	her zaman	25,7		
Geliştirilen yazılımın bakımı kolaydır	hiçbir zaman/ nadiren	11,4	3,71	0,957
	bazen	20		
	çoğunlukla	51,4		
	her zaman	17,1		
Yazılım projeleri	hiçbir zaman/ nadiren	22,9	3,34	1,136
zamanında teslim edilmektedir	bazen	31,4		
	çoğunlukla	28,6		
	her zaman	17,1		
Yazılım projeleri	hiçbir zaman/ nadiren	22,9	3,31	1,105
hesaplanan maliyette tamamlanmaktadır	bazen	31,4		
	çoğunlukla	31,4		
	her zaman	14,3		

* Cevaplar şu şekilde gruplanmıştır: Hiçbir zaman/ Nadiren:1,2; Bazen: 3; Çoğunlukla:4, Her zaman:5

** 35 anket=100%

Formal metodoloji kullanan organizasyonlarla kullanmayanların proje başarı düzeylerinin karşılaştırılması için yapılan t testi sonrasında formal metodoloji kullananlarla kullanmayan organizasyonların proje başarılarının birbirinden farklı olmadığı bulunmuştur (bkz. Tablo A.37). Bu sonuç, Fletcher ve Hunt (1993) un gözlemlerini doğrular biçimdedir. Organizasyonlar için uygun ölçme teknikleri ve araçların kullanıldığı uzun dönemli, devam eden işler üstlenilmedikçe yazılım mühendisliği uygulamalarının etkilerini izole etmek mümkün olmayacaktır.

Süreç iyileştirme çalışmaları yapan organizasyonlarla yapmayanların proje başarı düzeylerinin karşılaştırılması için yapılan t testi sonrasında da iyileştirme çalışması yapan ve yapmayanlar arasında bir farkın olmadığı gözlenmiştir (bkz. Tablo A.38). Burada iyileştirme çalışmaları yapan organizasyonların büyük bir çoğunluğunun başlangıç aşamasında olması farkın oluşmamasında en büyük etken olduğu düşünülmektedir.

12. SONUÇ

Yazılım işi belirsizliklerin çok olduğu, yapılandırılması güç bir iştir. Yazılım problemleri yapısız ve düzensiz problemlerdir. Yazılım işi aynı zamanda, kişisel tecrübe ve uzmanlık, kalite ve yazılım mühendisliği anlayışı, yüksek teknoloji gerektirmektedir. Tüm bu zorluklar, yine de yazılım sürecini iyileştirmeye engel teşkil etmemelidir.

Bu çalışmada Türkiye'deki yazılım geliştiren organizasyonlardaki yazılım geliştirme sürecini, geliştirme süresince karşılaşılan problemleri, kullanılan teknik ve araçları, yönetim süreçlerini, yapılan iyileştirme çalışmalarını ve bu esnada karşılaşılan zorlukları, son olarak da yazılım projelerinin başarı kriterlerine göre durumlarını anlamaya ilişkin bir anket çalışması yapılmıştır.

Ankete katılan 35 şirket üzerinden yapılan değerlendirmelere baktığımızda ortaya çıkan tablo çok iyi değildir. Özellikle proje yönetim sürecinde noksanlıkların olduğu görülmektedir.

Bulunan sonuçlar kısaca değerlendirilirse;

Ankete katılan şirketlerin büyük çoğunluğunun süreç boyunca en azından bir model izliyor olması iyidir. Yazılım şirketlerinde yazılım geliştirme süresince kullanılan yazılım geliştirme modeli, metodolojisi her organizasyona göre değişebilen bir karardır. Bu karar, organizasyonun yapısına, geliştirilecek olan projenin özelliklerine göre değişebilmektedir. Ankete katılan şirketlerdeki, yazılı belgelere dayalı, disiplinli süreç geliştirme metodolojileri kulanım oranı %62,9'dur. Belgelere dayalı metodolojiler, süreçleri olgunlaştırarak, kurumsallaştırarak kişilerden bağımsız hale getirmektedirler. Fakat bunu yaparken değişikliklere ayak uyduramama riski taşımaktadırlar. Yine de tutarlı bir yolda süreçler üzerine odaklanarak ilerlemek proje başarısı için iyidir.

Ankete katılan şirketlerin yazılım mühendisliği teknikleri kullanımımda nesne yönelimli tekniklere yapısal tekniklerden daha fazla ağırlık verdiği görülmektedir. Her safhada, birkaç teknik organizasyonlarca kullanılmaktadır. Fakat bakım safhası için gerekli görülen konfigürasyon yönetimi araçlarının düşük bir oranda kullanıyor olması projeler için risk teşkil etmektedir.

Yönetim desteğinin, deneyimli çalışanların, sistemli eğitimin eksikliği yazılım mühendisliği metotlarının, araç ve tekniklerinin kullanımını zorlaştıran etmenler olarak karşımıza çıkmaktadır. Firmalar, eğitime daha fazla yatırım yapmalı, çalışanların deneyimlerini paylaşacakları, iletişimin kolay kurulabileceği ortamlar sağlamalıdırlar.

Yazılım mühendisliği uygulamalarına karşı genel tutumlar ve başarılı adaptasyonu engelleyen etmenler hakkındaki genel düşünceler literatürü destekler şekildedir. Bu da yazılım mühendisliği uygulamaları konusunda genel bir bilginin olduğunu göstermektedir.

Yönetim süreçlerine baktığımızda ise bu sürecin fazla güçlü olmadığını görüyoruz. Özellikle risk yönetimi ve ölçüm yönetim aktivitelerinin çoğu organizasyon tarafından yerine getirilmediği görülmektedir. Oysa ki, projelerdeki başarısızlığın en temel nedenlerinden biri yazılım süreçlerinin etkin yönetilmemesidir. Yazılım süreçlerinin etkin yönetimi için geliştirilen yazılım süreç iyileştirme yaklaşımlarında da biraz yavaş davranıldığı gözlenmiştir. Bu konuda yönetim desteği, iyileştirme çalışmalarına bir proje gözüyle bakılması kriterleri önem teşkil etmektedir. Ankete katılan şirketlerde bu hususlarda problemlerin olduğu görülmüştür.

Geliştirme süresince en fazla sürenin kodlamaya harcanması, geliştirme sürecinde de karşılaşılan en sık problemlerin analiz sürecinin kalitesiyle ilgili olması yanlış bir yolun izlendiğini göstermektedir. Unutulmamalıdır ki analiz safhasına ayrılan sürenin uzun ve verimli geçmesi ileriki safhalarda karşılaşılan hataları azaltacak, dolayısı ile bu fayda maliyete ve zamana yansıyacaktır.

Projeler, başarı kriterlerine göre değerlendirildiklerinde zaman ve maliyet konusunda problemlerin olduğu sonucuna varılmıştır. Bu problemler, izlenilen yollara bağlı olabileceği gibi yazılımın kendi doğasından da kaynaklanabilmektedir. Ancak, yazılım geliştiriciler, Dr. Paul Dorsey'in de dediği gibi kendilerini yazılım oluşturmanın "farklı" olduğunu söyleyerek avutmamalıdırlar. Dolayısı ile yazılım geliştiren organizasyonlar, hedeflerine ulaşabilmek için yazılım yönetim ve geliştirme süreçlerini daha etkin ve verimli hale getirmelidirler.

Yazılım alanında da daha iyileştirilmesi gereken bir çok alan vardır. Yine de, yazılım mühendisleri yaptıklarıyla gurur duymalıdırlar. Kompleks yazılımlar olmadan, uzay keşfedilemeyecek, Internet ve modern telekomünikasyon olmayacak, tüm seyahatler daha tehlikeli ve pahalı olacaktı. Kısa sürede çok şey başarılmıştır. Türkiye'nin de hızla büyüyen bu sektörde daha fazla rekabet gücünün olması için konuya daha

fazla önem vermesi, yazılım projelerini dünya standartlarında geliştirmesi, tanımlı metot ve standartları eğitim kapsamına alması gerekmektedir.

KAYNAKLAR

Ashrafi, N., 2003. The impact of software process improvement on quality: in theory and practice, *Information & Management*, **40**, 677-690.

Arifoğlu, A. ve Gür, M., 2005. e-CMM : e-Kurum Olgunluk Modeli, Bilişim Sistemleri Enformatik Enstitüsü ODTÜ, Ankara.

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

Blackburn, J. D., Scudder, G. D. and Wassenhove L. N., 1996. Improving speed and productivity of software development., *IEEE Transactions on SW Engineering*, **22**. No: 12, 875-885.

Blum, B.I., 1992. Software engineering: a holistic view, Oxford University Press, New York.

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: Essays on Software Engineering, 20th Anniversary Edition Reading, MA: Addison-Wesley.

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

Chep, A., Tricarico, L., Bourdet, P. and Galuntucce, L., 1998. Design of object oriented database for the definition of machining operation sequences of workpieces, *Computers Industrial Engineering*, **34**, No.2, U.K., 257-279.

Dorsey, P., 2005. Top 10 reasons why systems projects fail, Dulcian, Inc.

Field, T., 1997. When bad things happen to good projects, *CIO*, October**,** 55-62.

Fletcher, T. and Hunt, J., 1993. Software engineering and CASE: bridging the culture gap, McGraw-Hill, New York.

Gibbs, W., 1994. Software’s chronic crisis, *Scientific American*, **271**(3), 86-95.

Gill, S. N., 2005. Factors affecting effective software quality management revisited, *ACM SIGSOFT Software Engineering Notes,* **30**, No: 2, March 2005.

İnce, F., 1998. Yazılımda süreç olgunluk yetenek modelleri, Boğaziçi Üniversitesi. İstanbul,Türkiye.

Jiang, J.J.,Klein, G., Hwang, H., Huang, J. and Hung, S., 2004. An exploration of the relationship between software development process maturity and project performance, *Information & Management,* **41**, 279-288.

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

Kettunen, P. And Laanti, M., 2005. How to steer an embedded software project: tactics for selecting the software process model, *Information and Software Technology,* **47**, 587-608.

Kolarik J. W., 1995. Creating quality, McGraw-Hill-International Editions.

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

Kurnaz, S., Çetin, Ö. ve İnce, F., 2003. Yazılım mühendisliğinde kalite ve UML, *Havacılık ve Uzay Teknolojileri Dergisi,* **1**, Sayı:2, 1-12.

MacCormack, A., 2001. Product-Development Practices that work, *MIT Sloan Management Review 42(2)*, **42**, No. 2, 75–84.

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

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.

Niazi, M., Wilson, D. and Zowghi D., 2005. A maturity model for the implementation of software process improvement: an empirical study, *The Journal of Systems and Software*, **74**, 155-172.

Nidumolu, S., 1996. Standardization, requirements uncertainty and software project performance, *Information & Management,* **31**, 135-150.

Özkan, M., 2001. Yazılımda kalite, *Computer Life Dergisi*, **53**, Haziran.

Paulk, M., 1995. How ISO 9000 compares with the CMM, *IEEE Software,* **12**(1), 74-84.

Poo, D.C.C. and Chung, M.K., 1998. Software engineering practices in Singapore, *The Journal of Systems and Software*, **41**, 3-15.

Pooley, R., 2004. Applying UML: advanced application, Butterworth-Heinemann, Oxford.

Proya, 2003. Yazılım geliştirme faaliyetlerinde değişiklik ve konfigürasyon yönetimi sistemlerinin artan önemi.

Reel, J. S., 1999. Critical success factors in software projects, *IEEE Software*, May/June, 18-23.

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

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.

Na, K., Li, X., Simpson, J.T. and Kim, K., 2004. Uncertainty profile and software project performance; A cross-national comparison, *The Journal of Systems and Software,* **70**, 155-163.

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.

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

The Standish Group, 2004. CHAOS Demographics and Project Resolution, 2004 Third Quarter Report.

The Standish Group, 2001. Extreme CHAOS - The 2001 update to the CHAOS report.

Tokgöz, G. M., 2004. UML ile Yazılım Modellenmesi , *EMO Bilgisayar Mühendisliği Dergisi,* Aralık.

EKLER

Tablo A.1 : Firma yapısı

Firma yapısı	Sıklık	Pay (%) *
tamamen yerli	28	80
yabancı ortaklı	5	14,3
tamamen yabancı	2	5,7

* 35 anket = 100%

Tablo A.2 : Çalışan sayısı

Çalışan sayısı	Sıklık	Pay (%) *
50 'den az	17	48,6
50-100 arası	12	34,3
100-200 arası	4	11,4
400'den fazla	2	5,7

* 35 anket = 100%

Tablo A.3 : Son 5 yılda geliştirilen yazılım sayısı

Geliş. yazılım sayısı	Sıklık	Pay (%) *
1-5 arası	11	31,4
6-10 arası	9	25,7
11-15 arası	3	8,6
16-20 arası	3	8,6
20'den fazla	9	25,7

* 35 anket = 100%

Tablo A.4 : Formal metodoloji kullanımı

	Cevaplar	Sıklık	Pay (%) *
Org. formal sistem geliştirme metod. kullanmaktadırlar	evet	22	62,9
	hayır	13	37,1

* 35 anket = 100%

Tablo A.5 : Formal metodoloji kullanma süresi

	Cevaplar	Sıklık	Pay (%) *
formal metodoloji kullanım süresi	1 yıldan az	0	0
	1-2 yıl	9	40,9
	3-4 yıl	6	27,3
	4 yıldan fazladır	7	31,8

* 35 anket = 100%

Tablo A.6 : Şirketin büyüklüğü ve formal metodoloji kullanımı arasındaki ilişki

Sirketin buyuklugu * formal sistem gelistirme metod. Crosstabulation

Count

		formal sistem gelistirme metod.		
		evet	hayýr	Total
Sirketin buyuklugu	50 den az	9	9	18
	50 den fazla	13	4	17
Total		22	13	35

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)	Exact Sig. (2-sided)	Exact Sig. (1-sided)
Pearson Chi-Square	2,624[b]	1	,105		
Continuity Correction[a]	1,613	1	,204		
Likelihood Ratio	2,676	1	,102		
Fisher's Exact Test				,164	,102
Linear-by-Linear Association	2,549	1	,110		
N of Valid Cases	35				

a. Computed only for a 2x2 table

b. 0 cells (,0%) have expected count less than 5. The minimum expected count is 6,31.

Tablo A.13 : Yazılım müh. metotlarına ilişkin eğitim durumu

	Cevaplar	Sıklık	Pay (%) *
Yazılım müh. metotlarına ilişkin düzenli eğitim programları sunulmaktadır.	evet	14	40
	hayır	21	60

* 35 anket = 100%

Tablo A.14 : Alınan eğitim biçimleri

Alınan eğitim biçimi	Sıralama	Cevaplar	Sıklık	Pay (%) *
Formal eğitimler				
	3	Şirket içi eğitimciler tarafından verilen eğitimler	20	57,1
	4	Kendi şirketiniz dışındaki eğitimciler tarafından verilen eğ.	14	40,0
Formal olmayan eğitimler				
	1	İş üstünde eğitim	27	77,1
	2	Kişilerin kendi çabasıyla eğitim (video, bilgisayar, …kanalıyla)	24	68,6
	5	Diğer...	0	0,0

* 35 anket = 100%

Tablo A.15: Yazılım mühendisliği uygulamalarına karşı gösterilen tutumlar

Sıralama *	Kriterler	Cevap**	Yüzde***	Ortalama	St. sapma
1	Nesne tabanlı yazılım mühendisliği	katılıyorum	77,2	4,14	1,192
	metotları faydalıdır	ne/ne	11,4		
		katılmıyorum	11,4		
2	SEP, yazılım geliştirme ve bakım	katılıyorum	77,2	4,06	1,11
	kalitesini iyileştirmektedir	ne/ne	14,3		
		katılmıyorum	8,6		
3	SEP, yazılım geliştirme ve bakım	katılıyorum	68,5	3,86	1,216
	sürelerini azaltmaktadır	ne/ne	20		
		katılmıyorum	11,4		
4	SEP, yazılım geliştirme ve bakımı	katılıyorum	74,3	3,77	1,33
	üzerindeki kontrolü güçlendirmektedir	ne/ne	5,7		
		katılmıyorum	20		
5	Organizasyon tarafından adapte edilen	katılıyorum	57,2	3,6	1,355
	standart metodolojilerin kullanılması kolaydır	ne/ne	22,9		
		katılmıyorum	20		
6	Maliyetleri azaltmaya yardım etmektedir	katılıyorum	57,2	3,4	1,193
		ne/ne	22,9		
		katılmıyorum	20		
7	SEP, organizasyonun rekabet avantajı	katılıyorum	51,4	3,29	1,178
	kazanmasına yardım etmektedir	ne/ne	25,5		
		katılmıyorum	22,9		

* Sıralama ortalamaya göre yapılmıştır.

** Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

Tablo A.17 : Yazılım mühendisliği uygulamalarına karşı tutumların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 4 olup olmadığının testi.

One-Sample Test

	Test Value = 4					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SEP, yazilim gelistirme ve bakimi uzerindeki kontrolu guclendirmektedir	-1,016	34	,317	-,229	-,69	,23
Maliyetleri azaltmaya yardim etmektedir	-2,975	34	,005	-,600	-1,01	-,19
SEP, organizasyonun rekabet avantaji kazanmasina yardim etmektedir	-3,589	34	,001	-,714	-1,12	-,31
SEP, yazilim gelistirme ve bakim surelerini azaltmaktadir	-,695	34	,492	-,143	-,56	,27
SEP, yazilim gelistirme ve bakim kalitesini iyilestirmektedir	,305	34	,763	,057	-,32	,44
Organizasyon tarafindan adapte edilen standart metodolojilerin kullanilmasi kolaydir	-1,747	34	,090	-,400	-,87	,07
Nesne tabanli yazilim muhendisligi metotlari faydalidir	,709	34	,483	,143	-,27	,55

Tablo A.18 : Yazılım mühendisliği uygulamalarına karşı tutumların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 3 olup olmadığının testi.

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
SEP, yazilim gelistirme ve bakimi uzerindeki kontrolu guclendirmektedir	3,431	34	,002	,771	,31	1,23
Maliyetleri azaltmaya yardim etmektedir	1,983	34	,055	,400	-,01	,81
SEP, organizasyonun rekabet avantaji kazanmasina yardim etmektedir	1,435	34	,160	,286	-,12	,69
SEP, yazilim gelistirme ve bakim surelerini azaltmaktadir	4,170	34	,000	,857	,44	1,27
SEP, yazilim gelistirme ve bakim kalitesini iyilestirmektedir	5,635	34	,000	1,057	,68	1,44
Organizasyon tarafindan adapte edilen standart metodolojilerin kullanilmasi kolaydir	2,620	34	,013	,600	,13	1,07
Nesne tabanli yazilim muhendisligi metotlari faydalidir	5,674	34	,000	1,143	,73	1,55

Tablo A.19 : Yazılım mühendisliği uygulamalarının başarılı adaptasyonunu engelleyen etmenlere karşı tutumlar

Sıralama *	Kriterler	Cevap**	Yüzde***	Ortalama	St. sapma
1	Yazılım mühendisliği metotları, araçları konusunda disiplinli eğitim eksikliği	katılıyorum	77,1	3,94	1,056
		ne/ne	14,3		
		katılmıyorum	8,6		
2	Yönetim desteğinin eksikliği	katılıyorum	65,7	3,77	1,215
		ne/ne	17,1		
		katılmıyorum	17,1		
3	Yaz.müh. metotlarını, araçlarını efektif kullanacak deneyimli çalışanların azlığı	katılıyorum	74,3	3,66	1,454
		ne/ne	5,7		
		katılmıyorum	20		
4	Büyük finanssal yatırımlar	katılıyorum	48,6	3,4	1,311
		ne/ne	28,6		
		katılmıyorum	22,9		
5	Adapte etmek için amaçların net, açık olmaması	katılıyorum	42,9	3,31	1,367
		ne/ne	28,6		
		katılmıyorum	28,6		
6	Yaz. Müh. metotları ve araçlarının karmaşıklığı	katılıyorum	47,6	3,17	1,043
		ne/ne	25,7		
		katılmıyorum	25,7		
7	Yaz. müh. araçlarının seçiminde, araçları kullanacak mühendislerin, teknik danışmanların etkin olmaması	katılıyorum	42,8	3,09	1,245
		ne/ne	25,7		
		katılmıyorum	31,4		
8	Yazılım mühendisliği metotlarıyla geliştirilen yeni sistemlerin eski sistemlerle uyuşmaması	katılıyorum	40	3,06	1,235
		ne/ne	28,6		
		katılmıyorum	31,4		
9	Yaz. müh. metotlarını destekleyecek uygun ortam ve araçların eksikliği	katılıyorum	40	3	1,237
		ne/ne	28,6		
		katılmıyorum	31,4		
10	Günümüzdeki projeler için standart metodolojilerin uygun olmaması	katılıyorum	34,3	2,71	1,447
		ne/ne	17,1		
		katılmıyorum	48,6		

* Sıralama ortalamaya göre yapılmıştır.

** Cevaplar şu şekilde gruplanmıştır: Katılmıyorum:1,2; Ne/Ne: 3; Katılıyorum: 4,5

** 35 anket=100%

Tablo A.20 : Yazılım mühendisliği uygulamalarının başarılı adaptasyonunu engelleyen etmenlere karşı tutumların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 3 olup olmadığının testi.

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Yazilim muhendisligi metotlarini, araclarini efektif kullanacak deneyimli calisanlarin azligi	2,674	34	,011	,657	,16	1,16
Yazilim muhendisligi metotlari ve araclarinin karmasikligi	,973	34	,338	,171	-,19	,53
Yazilim muh. metotlariyla gelistirilen yeni sistemlerin eski sistemlerle uyusmamasi	,274	34	,786	,057	-,37	,48
Yazilim muh. metotlari, araclari konusunda disiplinli eðitim eksikligi	5,284	34	,000	,943	,58	1,31
Yönetim desteðinin eksikligi	3,757	34	,001	,771	,35	1,19
Buyuk finanssal yatirimlar	1,806	34	,080	,400	-,05	,85
Gunumuzdeki projeler icin standart metodolojilerin uygun olmamasi	-1,169	34	,251	-,286	-,78	,21
Yazilim muh. araclarinin seciminde, araclari kullanacak muhendislerin,teknik danismanlarin etkin olmamasi	,407	34	,686	,086	-,34	,51
Adapte etmek icin amaclarin net, acik olmamasi	1,360	34	,183	,314	-,16	,78
Yazilim muh. metotlarini destekleyecek uygun ortam ve araclarin eksikligi	,000	34	1,000	,000	-,42	,42

Tablo A.21 : Yazılım mühendisliği uygulamalarının başarılı adaptasyonunu engelleyen etmenlere karşı tutumların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 4 olup olmadığının testi.

One-Sample Test

	Test Value = 4					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Yazilim muhendisligi metotlarini, araclarini efektif kullanacak deneyimli calisanlarin azligi	-1,395	34	,172	-,343	-,84	,16
Yazilim muhendisligi metotlari ve araclarinin karmasikligi	-4,701	34	,000	-,829	-1,19	-,47
Yazilim muh. metotlariyla gelistirilen yeni sistemlerin eski sistemlerle uyusmamasi	-4,515	34	,000	-,943	-1,37	-,52
Yazilim muh. metotlari, araclari konusunda disiplinli eðitim eksikligi	-,320	34	,751	-,057	-,42	,31
Yönetim desteðinin eksikligi	-1,113	34	,273	-,229	-,65	,19
Buyuk finanssal yatirimlar	-2,708	34	,011	-,600	-1,05	-,15
Gunumuzdeki projeler icin standart metodolojilerin uygun olmamasi	-5,258	34	,000	-1,286	-1,78	-,79
Yazilim muh. araclarinin seciminde, araclari kullanacak muhendislerin,teknik danismanlarin etkin olmamasi	-4,343	34	,000	-,914	-1,34	-,49
Adapte etmek icin amaclarin net, acik olmamasi	-2,967	34	,005	-,686	-1,16	-,22
Yazilim muh. metotlarini destekleyecek uygun ortam ve araclarin eksikligi	-4,784	34	,000	-1,000	-1,42	-,58

Tablo A.23 : Proje yönetim süreçleri ile ilgili yaklaşımların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 3 olup olmadığının testi.

One-Sample Test

	Test Value = 3					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Yapilan proje planlari (proje kapsaminin belirlenmesi, gerekli insan gucu ve kaynaklarin belirlenmesi)yeterli ve gercekcidir	1,298	34	,203	,257	-,15	,66
Yeterli sayida ve siklikta ara hedefler belirlenmektedir	2,678	34	,011	,457	,11	,80
Gözden gecirme görusmeleri etkin ve sistemli bir sekilde yapilmaktadir	3,174	34	,003	,571	,21	,94
Yapilan isler takip edilmekte ve sonuclar denetlenmektedir	4,941	34	,000	,914	,54	1,29
Kalite aktiviteleri yerine getirilmektedir	1,769	34	,086	,314	-,05	,68
Problemler önceden saptanabilmektedir	1,601	34	,119	,257	-,07	,58
Riskten korunma, riski izleme ve yönetme plani olusturulmaktadir	,000	34	1,000	,000	-,40	,40
Yazilim gelistirme ekibi deneyimlidir, degilse egitimden gecirilmektedir	4,552	34	,000	,771	,43	1,12
Isini iyi yapanla yapmayani ayirdedecek ceza/takdir mekanizmalari vardir	-,138	34	,891	-,029	-,45	,39
Takimlar arasi/takim ici iletisimde problem yoktur	3,684	34	,001	,771	,35	1,20
Hedeflere ulasmada uyum ve isbirliði sorunu yasanmamaktadir	3,347	34	,002	,657	,26	1,06
Hedefler, yetki ve sorumluluklar tam olarak bellidir	2,741	34	,010	,543	,14	,95
Yaz. Gelistime safhalarinda bulunan hata sayilari, tipleri kaydedilmektedir	,988	34	,330	,257	-,27	,79
Proje sonrasi görusmeler yapilarak deneyimlerden kazanclar saðlanmaktadir	1,382	34	,176	,314	-,15	,78

Tablo A.25 : Proje yönetim süreçleri ile ilgili yaklaşımların algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 4 olup olmadığının testi.

One-Sample Test

	Test Value = 4					
					95% Confidence Interval of the Difference	
	t	df	Sig. (2-tailed)	Mean Difference	Lower	Upper
Yapilan proje planlari (proje kapsaminin belirlenmesi, gerekli insan gucu ve kaynaklarin belirlenmesi)yeterli ve gercekcidir	-3,750	34	,001	-,743	-1,15	-,34
Yeterli sayida ve siklikta ara hedefler belirlenmektedir	-3,180	34	,003	-,543	-,89	-,20
Gözden gecirme görusmeleri etkin ve sistemli bir sekilde yapilmaktadir	-2,380	34	,023	-,429	-,79	-,06
Yapilan isler takip edilmekte ve sonuclar denetlenmektedir	-,463	34	,646	-,086	-,46	,29
Kalite aktiviteleri yerine getirilmektedir	-3,861	34	,000	-,686	-1,05	-,32
Problemler önceden saptanabilmektedir	-4,626	34	,000	-,743	-1,07	-,42
Riskten korunma, riski izleme ve yönetme plani olusturulmaktadir	-5,086	34	,000	-1,000	-1,40	-,60
Yazilim gelistirme ekibi deneyimlidir, degilse egitimden gecirilmektedir	-1,349	34	,186	-,229	-,57	,12
Isini iyi yapanla yapmayani ayirdedecek ceza/takdir mekanizmalari vardir	-4,970	34	,000	-1,029	-1,45	-,61
Takimlar arasi/takim ici iletisimde problem yoktur	-1,092	34	,283	-,229	-,65	,20
Hedeflere ulasmada uyum ve isbirliði sorunu yasanmamaktadir	-1,746	34	,090	-,343	-,74	,06
Hedefler, yetki ve sorumluluklar tam olarak bellidir	-2,308	34	,027	-,457	-,86	-,05
Yaz. Gelistime safhalarinda bulunan hata sayilari, tipleri kaydedilmektedir	-2,853	34	,007	-,743	-1,27	-,21
Proje sonrasi görusmeler yapilarak deneyimlerden kazanclar saðlanmaktadir	-3,015	34	,005	-,686	-1,15	-,22

Tablo A.29 : Yazılım süreç iyileştirme çalışmalarına ilişkin durum

	Cevaplar	Sıklık	Pay (%) *
Süreç iyileştirme çalışması yapılmaktadır	evet	22	62,9
	hayır	13	37,1

* 35 anket = 100%

Tablo A.30 : İyileştirme sürecinde kullanılan standartlar

	Cevaplar	Sıklık	Pay (%) *	Pay (%) **
Kullanılan standartlar	CMM	6	17,1	27,3
	ISO9001	8	22,9	36,4
	Diğer kalite güvence programları	8	22,9	36,4
	Toplam	22	62,9	100

* 35 anket = 100%

* 22 cevap = 100%

Tablo A.31 : Süreç iyileştirme programının başarısı

	Cevaplar	Sıklık	Pay (%) *	Pay (%) **
İyileştirme programının başarısı	Henüz başlangıç aşamasındayız	13	37,1	59,1
	Başarısız	1	2,9	4,6
	Ne/Ne	3	8,6	13,6
	Başarılı	5	14,3	22,7
	Toplam	22	62,9	100

* 35 anket = 100%

* 22 cevap = 100%

Tablo A.35 : Yazılım geliştirme projelerinin başarı kriterlerinin oluşma sıklığının algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 3 olup olmadığının testi.

One-Sample Test

	Test Value = 3					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
Yazilim uygulamalari, kullaniciyi yuksek derecede tatmin etmektedir	6,297	34	,000	,886	,60	1,17
Gelistirilen yazilimin, bakimi kolaydir	4,415	34	,000	,714	,39	1,04
Yazilim projeleri hesaplanan maliyette tamamlanmaktadir	1,682	34	,102	,314	-,07	,69
Yazilim projeleri zamaninda teslim edilmektedir	1,785	34	,083	,343	-,05	,73

Tablo A.36 : Yazılım geliştirme projelerinin başarı kriterlerinin oluşma sıklığının algılanan ortalama değerlerinin istatistiksel olarak anlamlı bir şekilde 4 olup olmadığının testi.

One-Sample Test

	Test Value = 4					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
Yazilim uygulamalari, kullaniciyi yuksek derecede tatmin etmektedir	-,813	34	,422	-,114	-,40	,17
Gelistirilen yazilimin, bakimi kolaydir	-1,766	34	,086	-,286	-,61	,04
Yazilim projeleri hesaplanan maliyette tamamlanmaktadir	-3,670	34	,001	-,686	-1,07	-,31
Yazilim projeleri zamaninda teslim edilmektedir	-3,422	34	,002	-,657	-1,05	-,27

Tablo A.37 : Formal metodoloji kullanan organizasyonlarla kullanmayanların proje başarı düzeylerinin karşılaştırılması

Group Statistics

	formal sistem gelistirme metod.	N	Mean	Std. Deviation	Std. Error Mean
toplamkalite	evet	22	14,0455	3,53859	,75443
	hayýr	13	14,6154	2,56705	,71197

Independent Samples Test

		Levene's Test for quality of Variance		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	ig. (2-tailed	Mean Difference	Std. Error Difference	Lower	Upper
toplamkalit	Equal varianc assumed	1,578	,218	-,506	33	,616	-,56993	1,12623	2,86126	1,72140
	Equal varianc not assumed			-,549	31,432	,587	-,56993	1,03734	2,68442	1,54456

Tablo A.38 : Süreç iyileştirme çalışması yapan organizasyonlarla yapmayanların proje başarı düzeylerinin karşılaştırılması

Group Statistics

	Surec iyilestirme calismasi	N	Mean	Std. Deviation	Std. Error Mean
toplamkalite	evet	22	13,5455	3,20308	,68290
	hayir	13	15,4615	2,87563	,79756

Independent Samples Test

		Levene's Test for quality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	ig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
toplamkalit	Equal varianc assumed	,277	,602	-1,774	33	,085	-1,91608	1,08027	-4,11391	,28174
	Equal varianc not assumed			-1,825	27,576	,079	-1,91608	1,04997	-4,06835	,23618

EK A

Sayın İlgili,

İTÜ İşletme Fakültesi'nde yürütülmekte olan yüksek lisans tezi kapsamında; "Yazılım Geliştirme Sürecinin İyileştirilmesi ve Türkiye Uygulamaları" başlıklı bir çalışma yapılmaktadır. Bu bağlamda ilişikte sunulan anketimize vereceğiniz cevaplar ve ayıracağınız süre ile bu çalışmaya katkıda bulunmanızı diliyoruz. Lütfen yanıtınızı 8 Nisan Cumartesi akşamına kadar gönderiniz. İşletmenize ait bilgiler kesinlikle gizli tutulacak olup, elde edilen bilgiler toplu halde ve genel sonuçlar çerçevesinde analiz edilecek ve arzu ettiğiniz takdirde tarafınıza ulaştırılacaktır.

Halefşan SÜMEN Zuhal GÜL

Adres

İstanbul Teknik Üniversitesi

İşletme Fakültesi

İşletme Mühendisliği Bölümü

Maçka 34357 / İSTANBUL

E-posta: sumenh@itu.edu.tr, gulz@itu.edu.tr

Anketi dolduran kişinin:

Adı Soyadı:

Görevi:

1.BÖLÜM: İşletmenin demografik özellikleri ile ilgili soruları içermektedir.

İşletmenin Adı:

Telefon No:

Web Adres:

1- Firmanız aşağıdaki yazılım uygulama alanlarından hangisine uymaktadır?

☐ Bilimsel ve mühendislik yazılımları (istatistik analiz paketleri,...)

☐ Mesleki yazılımlar (stok kontrol p., müşteri takip p., muhasebe p.,...)

☐ Yapay zeka yazılımları (robot yaz., satranç ya da briç oynatan p. …)

☐ Görüntüsel yazılımlar(oyun ve animasyon yazılımları,..)

☐ Sistem yazılımları (derleyiciler, haberleşme programları, işletim sistemi,...)

2- Firmanız aşağıdaki yapılardan hangisine uymaktadır?

☐ Devlet ☐ Tamamen Yerli Organizasyon

☐ Tamamen Yabancı ☐ Yabancı Ortaklı

3- Firmanızda çalışan sayısı ne kadardır?

☐ 50'den daha az ☐ 50 -100 arası

☐ 100-200 arası ☐ 200-400 arası ☐ 400'den daha fazla

4- Firmanız son 5 yılda kaç tane yazılım geliştirmiştir?

☐ 1-5 arası ☐ 6-10 arası

☐ 11-15 arası ☐ 16-20 arası ☐ 20'den fazla

5- Firmanızın yazılım aktivitelerini sıralamak gerekirse nasıl sıralardınız?

[1: en fazla…….3: en az]

Yeni uygulamalar geliştirmek: _____

Yapılan yazılımlara bakım yapmak: _____

Son kullanıcılara operasyonel, teknik destek sağlamak: _____

2. BÖLÜM: Yazılım geliştirme sürecinde kullanılan yazılım mühendisliği metodolojilerinin, modellerin, teknik ve araçların genel durumunu incelemeye ilişkin soruları içermektedir.

A. Yazılım geliştirme metodolojileri uygulamalarına ilişkin soruları içermektedir.

1- Disiplinli,dokümante edilmiş (formal) sistem geliştirme metodolojileri kullanıyor musunuz?

☐ Evet ☐ hayır

2- Kullanıyorsanız, ne kadar süredir kullanmaktasınız?

☐ 1 yıldan az ☐ 1-2 yıl

☐ 3-4 yıl ☐ 4 yıldan fazladır

3- Formal veya formal olmayan (informal) metodolojiyle hangi yazılım süreci modellerini izlemektesiniz?

☐ Şelale modeli

☐ RUP (Rasyonel Bütünleştirme Süreci M.)

☐ Artışlı modeller (aşama aşama teslim)

☐ Çevik modeller (XP,FDD)

☐ Spiral model (şelale+risk analizi)

☐ 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 standartlaştırılması

☐ Veri sözlülüğü

☐ UML

☐ Nesne tabanlı analiz

☐ Varlık-Bağıntı diyagramları

☐ Hızlı prototip

☐ Durum değişikliği diyagramı ve analizi

2-Tasarım safhasında;

☐ Rapor tasarlayıcılar ☐ Prototip tasarım

☐ Ekran tasarlayıcılar ☐ İşlem hacim analizi

☐ Yapı çizelgeleri ☐ Diyalog akış diyagramları

☐ Karar ağaçları ☐ HIPO (hiyerarşik girdi-süreç-çıktı) çizelgeleri

☐ Nesne tabanlı tasarım ☐ Diğer…

3- Kodlama safhasında;

☐ 4.kuşak diller (RPG, Visual Basic, Oracle Forms, Access. Paradox,Foxpro...)

☐ 3.kuşak diller (Cobol, Fortran, Basic, Pascal, C,...)

☐ Diğer…

4-Test safhasında;

☐ Kara kutu testi (işlevselliğinin sınandığı test) ☐ Yük, zorlanım, performans testi

☐ Beyaz kutu testi (yazılım kodunun sınanması) ☐ Güvenlik testi

☐ Değişiklerin entegresinden sonra yapılan bağlanım t. ☐ Birim testi

☐ Walkthrough (gözlem) ☐ Diğer

5- Bakım safhasında;

☐ Yazılım değişim yönetimi prosedürleri

☐ Konfigürasyon yönetim prosedürleri

☐ Diğer

C- Yazılım mühendisliği metotlarında eğitim durumuna ilişkin soruları içermektedir.

1- Şirketiniz yazılım mühendisliği metotlarına ilişkin düzenli eğitim programları sunmakta mıdır?

☐ evet ☐ hayır

2- Alınan eğitim biçimlerini işaretleyiniz.

Formal eğitimler

☐ Kendi şirketiniz dışındaki eğitimciler tarafından verilen eğitimler

☐ Şirket içi eğitimciler tarafından verilen eğitimler

☐ Diğer

Formal olmayan eğitimler

☐ İş üstünde eğitim

☐ Kişilerin kendi çabasıyla eğitim (video, bilgisayar, …kanalıyla)

☐ Diğer

D- Yazılım müh. uygulamalarının (SEP-Software Engineering Practices) (teknik ve araçlar, metodolojiler) organizasyon üzerindeki etkilerini içermekte olup, bu kriterlere ne ölçüde katılmakta olduğunuzu belirtiniz.

1: Hiç katılmıyorum ……..5:Tamamen katılıyorum

	1	**2**	**3**	**4**	**5**
1- SEP, yazılım geliştirme ve bakımı üzerindeki kontrolü güçlendirmektedir.	☐	☐	☐	☐	☐
2- Maliyetleri azaltmaya yardım etmektedir.	☐	☐	☐	☐	☐
3- SEP, organizasyonun rekabet avantajı kazanmasına yardım etmektedir	☐	☐	☐	☐	☐
4- SEP, yazılım geliştirme ve bakım sürelerini azaltmaktadır.	☐	☐	☐	☐	☐
5- SEP, yazılım geliştirme ve bakım kalitesini İyileştirmektedir.	☐	☐	☐	☐	☐
6- Organizasyon tarafından adapte edilen standart metodolojilerin kullanılması kolaydır.	☐	☐	☐	☐	☐
7- Nesne tabanlı yazılım mühendisliği metotları faydalıdır.	☐	☐	☐	☐	☐

E- Yazılım müh. uygulamalarının (SEP - Software Engineering Practices) (teknik ve araçlar, metodolojiler) başarılı adaptasyonunu engelleyen etmenleri içermektedir. Bu etmenlere ne ölçüde katılmakta olduğunuzu belirtiniz.

1: Hiç katılmıyorum ……..5:Tamamen katılıyorum

	1	2	3	4	5
1- Yazılım mühendisliği metotlarını, araçlarını efektif kullanacak deneyimli çalışanların azlığı	☐	☐	☐	☐	☐
2- Yazılım mühendisliği metotları ve araçlarının karmaşıklığı	☐	☐	☐	☐	☐
3- Yazılım müh. metotlarını destekleyecek uygun ortam ve araçların eksikliği	☐	☐	☐	☐	☐
4- Yazılım mühendisliği metotlarıyla geliştirilen yeni sistemlerin eski sistemlerle uyuşmaması	☐	☐	☐	☐	☐
5- Yazılım mühendisliği metotları, araçları konusunda disiplinli eğitim eksikliği	☐	☐	☐	☐	☐
6-Yönetim desteğinin eksikliği	☐	☐	☐	☐	☐
7- Büyük finanssal yatırımlar	☐	☐	☐	☐	☐
8- Adapte etmek için amaçların net, açık olmaması	☐	☐	☐	☐	☐
9- Günümüzdeki projeler için standart metodolojilerin uygun olmaması	☐	☐	☐	☐	☐
10-Yazılım müh. araçlarının seçiminde, araçları kullanacak mühendislerin, teknik danışmanların etkin olmaması	☐	☐	☐	☐	☐

3. BÖLÜM: Proje yönetim süreçleri ile ilgili genel durumu incelemeye ilişkin soruları içermektedir.

A-Şirketinizin mevcut durumunu aşağıdaki kriterlere göre değerlendiriniz.

1: Hiç katılmıyorum ……..5:Tamamen katılıyorum

	1	2	3	4	5
1- Yapılan proje planları (proje kapsamının belirlenmesi, gerekli insan gücü ve kaynakların belirlenmesi) yeterli ve gerçekçidir.	☐	☐	☐	☐	☐
2- Yeterli sayıda ve sıklıkta ara hedefler belirlenmektedir.	☐	☐	☐	☐	☐
3- Gözden geçirme görüşmeleri etkin ve sistemli bir şekilde yapılmaktadır.	☐	☐	☐	☐	☐
4- Yapılan işler takip edilmekte ve sonuçlar denetlenmektedir.	☐	☐	☐	☐	☐
5-Kalite aktiviteleri yerine getirilmektedir.	☐	☐	☐	☐	☐
6- Problemler önceden saptanabilmektedir.	☐	☐	☐	☐	☐
7- Riskten korunma, riski izleme ve yönetme planı oluşturulmaktadır.	☐	☐	☐	☐	☐
8- Yazılım geliştirme ekibi deneyimlidir, değilse eğitimden geçirilmektedir.	☐	☐	☐	☐	☐
	1	**2**	**3**	**4**	**5**
9- İşini iyi yapanla yapmayanı ayırt edecek ceza/takdir mekanizmaları vardır.	☐	☐	☐	☐	☐
10-Takımlar arası/takım içi iletişimde problem yoktur.	☐	☐	☐	☐	☐
11- Hedeflere ulaşmada uyum ve işbirliği sorunu yaşanmamaktadır.	☐	☐	☐	☐	☐
12-Hedefler, yetki ve sorumluluklar tam olarak bellidir.	☐	☐	☐	☐	☐
13- Yaz. geliştirme safhalarında bulunan hata sayıları, tipleri kaydedilmektedir.	☐	☐	☐	☐	☐
14- Proje sonrası görüşmeler yapılarak deneyimlerden kazançlar sağlanmaktadır.	☐	☐	☐	☐	☐

B- Yazılım Süreç İyileştirme çalışmalarına ilişkin soruları yanıtlayınız.

1-Yazılım süreci iyileşmesi için bir çalışma yapmakta mısınız?

☐ evet ☐ hayır

Eğer hayır ise 4.Bölüme geçiniz.

2- Hangi standartları kullanmaktasınız?

☐CMM ☐ISO-9001 ☐SPICE ☐Diğer kalite güvence programları

3- İyileştirme programı ne kadar başarılı oldu?

1: başarısız…….5: çok başarılı

1 2 3 4 5
☐ ☐ ☐ ☐ ☐

☐ Henüz başlangıç aşamasındayız.

4- Süreç iyileştirme çalışmalarında karşı karşıya kaldığınız 3 temel problem:

_____
_____
_____

4. BÖLÜM: Geliştirme sürecine ait genel soruları içermektedir.

1- Yazılım geliştirme safhalarını, harcanan zaman miktarlarına göre numaralandırınız?

[1: en az…….4: en fazla]

Gereksinim belirlenmesi, analizi: _____
Tasarım: _____
Kodlama: _____
Test: _____

2- Yazılım geliştirme sürecinde karşılaştığınız 3 temel problem nedir?

_____
_____
_____

5. BÖLÜM: Bu bölüm, yazılım geliştirme projelerinin başarı kriterlerini içermektedir.

1: Hiçbir zaman……..5:Her zaman

	1	2	3	4	5
1- Yazılım uygulamaları, kullanıcıyı yüksek derecede tatmin etmektedir.	☐	☐	☐	☐	☐
2- Geliştirilen yazılımın, bakımı kolaydır.	☐	☐	☐	☐	☐
3- Yazılım projeleri hesaplanan maliyette tamamlanmaktadır.	☐	☐	☐	☐	☐
4- Yazılım projeleri zamanında teslim edilmektedir.	☐	☐	☐	☐	☐

ÖZGEÇMİŞ

Zuhal Gül 1980 yılında İstanbul'da doğmuştur. 1998 yılında Haydarpaşa Lisesinden mezun olmuştur. 1998-2003 yılları arasında İstanbul Teknik Üniversitesi Matematik Mühendisliği Lisans öğrenimini tamamlamıştır. 2003 tarihinde İstanbul Teknik Üniversitesi Yüksek Lisans İşletme Mühendisliği öğrenimine başlamıştır.