T.C. TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DOKTORA TEZ ÇALIŞMASI

Yazılım Mühendisliğinde Uygulama Geliştirmede Bir Çözüm:Yazılım Konfigürasyon Yönetimi

Hazırlayan : R.Reha ÇETİN
Danışman : Yrd. Doç. Dr. Cavit TEZCAN

T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM MÜHENDİSLİĞİNDE UYGULAMA GELİŞTİRMEDE BİR ÇÖZÜM : YAZILIM KONFİGÜRASYON YÖNETİMİ

R.REHA ÇETİN

DOKTORA TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN :

Yrd. Doç.Dr. Cavit TEZCAN

EDİRNE–2007

Doktora Tezi
Trakya Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Bölümü

ÖZET

Değişiklik yaşamımızın aslında değişmeyen tek gerçeğidir. Yazılımlarda yaşayan birer varlıktır. Nasıl bir canlının yaşamından ve yaşam döngüsünden bahsediyorsak , yazılımların da bir yaşam döngüsü vardır.

Yazılım Mühendisliği disiplini , yazılımların yaşam döngüsünü oluşturan tüm süreçleri ve bu süreçlerdeki tüm faaliyetleri bilimsel bir yaklaşımla , modern anlamda bir mühendislik yaklaşımı ile ele almayı amaçlar. Yazılımlarda yaşamlarını devam ettirebilmek için sürekli olarak çevrelerine , değişikliklere ve isteklere uyum sağlamak zorundadırlar.

Bu disiplin oldukça karmaşık tasarlanan yazılımlar ve bunların hayata geçirilmesi süreçleri sonunda özellikle gelen yeni isteklere , değişikliklere yazılımların belirli bir kalite güvencesi ile uyarlanması için zamana ve maliyetlere karşı yarışmaktadır. Bu yarış içinde iş kalitesi , bunun güvencesi , hızı , organizasyonu , yönetilmesi , projelendirilmesi , mevcut yapıya uyumluluğu , gelişmeye açık olması ve optimum maliyetle işlemin yapılması da global dünyada kaçılmaz bir iş zorunluluğu olmaktadır.

İşte bu aşamada yazılım çalışma ekipleri için Yazılım Konfigürasyon Yönetimi, değişiklerin yönetilebilmesi için bir çözüm olarak karşımıza çıkmaktadır.

Disiplin olarak yazılım geliştirme sürecinde değişikliklerin yazılımın yaşam platformu içinde bilimsel olarak bir tanım , yapı içinde ele alınması tanımlanması

işlemini çözüm olarak yapabiliriz. Bizler bu araştırma ve çalışmamızda Yazılım Mühendisliği'nde bir çözüm olarak tanımlanan yazılımlarımızın kalite güvencesi için Yazılım Konfigürasyon Yönetimi planlamasını ele almaktayız. Doğal olarak yazılımların sistemsel yapılarını , tanımlarını ve hangi amaçlar için temelde geliştirildikleri ile ilgilendik. Yazılımların da bir proje olması sebebi ile Proje Yönetim mantığı ile bilimsel bir yaklaşımla yönetilmesi ve ele alınması gereğini vurguladık. Yazılım yaşam döngüsü ve döngü içindeki adımları ele aldık. Kalite ve kalite güvencesi metodlarının bize sağladıklarına baktık. Böylece önümüze yazılım değişikliklerin uyarlamasından bir çözüm yaklaşımı , önerisi olarak da Yazılım Konfigürasyon Yönetimi kavramı , planlaması çalışma konusu olarak çıkmıştır.

Doctorate Thesis
Trakya University Graduate School of
Natural and Applied Sciences
Department of Computer Engineering

ABSTRACT

Changes are only permanent item in our life. The software is also vital entity in The World. We can mention about a life cycle for software like a living creatures. Indeed, this is called " software life cycle" for all living systems.

The software engineering will aimed to handle all of activities for life cycles such as defining requirements , identifiying processes , planning , designing , testing , assuring , implementing and maintaining the systems. We will always adapt the software in case of changes and change requests by the help of our Software Configuration Management plans. This is cyclic job process for all living software as a part of software life cycle.

The goal of Software Configuration Management plans are to ensure that procedures and policies for all changing , change requests , appliying changes as well as to control and implement for a software systems. We will reach as a solution this is Software Configuration Management because of working , producing effectively for all software systems.

Software Configuration Management is an umbrella activity that is applied through the software process. All information produced as part of software engineering becomes a natural part of software configuration. The configuration management is also managed and organized all changes the context of software and designed systems .

TEŞEKKÜR

Doktora çalışmam sırasında beni sürekli bıkmadan destekleyen, her türlü yardımı sağlayan Prof.Dr. Mesut RAZBONYALI ve Yrd. Doç. Cavit TEZCAN'a çok teşekkür ederim.

Çalışmalarım sırasında bana uzun zamandır sabırla, sevgiyle değerli katkılarını sağlayan sevgili Eşim Elif Oya, canım Kızım Burcu, Annem Ayla ve Babam Yılmaz ÇETİN'e çok teşekkür ederim.

Ayrıca yardımlarını esirgemeyen tüm Trakya Üniversite'si Bilgisayar Mühendisliği ve Fen Bilimleri Enstitüsü bölüm çalışanlarına teşekkür ederim.

İÇİNDEKİLER

ÖZET 3
ABSTRACT 5
TEŞEKKÜR 6
1. BİLGİ SİSTEMLERİ KAVRAMLARI 10
1.1. Bilgi Sistemleri Tipleri 10
1.2. Hareket 'Transaction' İşleme Sistemleri 11
1.3. Ofis Otomasyonu ve Bilgi İşleme Tabanlı Sistemler 11
1.4. Yönetim Bilişim Sistemleri 12
1.5. Karar Destek Sistemleri 12
1.6. Yapay Zeka'ya dayanan Uzman Sistemler 12
1.7. Bilgisayar Destekli Ortak Çalışma ve Paylaşım Ortamlı Sistemler 13
1.8. Üst Yönetime Destek olan İzleme, Anlık Raporlama Sistemleri 13
2. SİSTEM GELİŞTİRMENİN YAŞAM DÖNGÜSÜ 14
2.1. Amaç,Durum ve Problemi Belirleme 15
2.2. Bilgi İsteklerini Belirleme 16
2.3. Sistem Gereksinimlerinin Analizi 16
2.4. Önerilen Sistemi Tasarlama 17
2.5. Yazılım Geliştirme ve Belgeleme 17
2.6. Sistemin Testleri ve Bakımları 18
2.7. Sistemin Yürütülmesi ve Değerlendirilmesi 19
2.8. Bakımın Sistem Üzerindeki Etkileri 19
2.9. CASE Tool/Araçlarının Kullanımı ve Sebepleri 20
2.9.1. Analistlerimizin Üretkenliğini Arttırmak 20
2.9.2. Analistlerle Kullanıcı İletişimini Geliştirmek 21
2.9.3. Sistem Yaşam Döngüsü Aktivitelerinin Sistem Entegrasyonu 21
2.9.4. Doğru , Hassas Değişiklik ve Bakım Faaliyetlerini Başarmak 21
2.10. Tersine ve Yeniden Yapılanma olarak Yazılım Mühendisliği 22
2.11. Nesneye Dayalı Sistem Analizi ve Tasarım 23
3. YAZILIM MÜHENDİSLİĞİ UYGULAMASI 25
3.1. Yazılım Mühendisliği ile Kalite Güvencesi 25
3.1.1. Toplam Kalite Yönetimi Yaklaşımı 26
3.1.2. 6 SİGMA 26
3.1.3. Toplam Kalite Yönetimi Sorumluluğu 27
3.1.4. Yapısal Yürütme Yaklaşımı 27
3.1.5. Sistem Geliştirme ve Tasarımı 28
3.1.6. Bottom-Up Tasarım 29
3.1.7. Top-Down Tasarım 29
3.1.8. Modüler Geliştirme 30
3.2. Yazılım Mühendisliği ve Belgeleme 31
3.2.1. Ön Kodlama /gerçek olmayan (Pseudocoding) 31
3.2.2. Prosedür El Kitapları 32
3.2.3. Sözel Yöntemler 33
3.2.4. Tasarım ve Belgeleme Tekniği Seçimi 33
4. TEST, BAKIM ve DENETLEME YAPISI 35

4.1. Test Prosesi 35
4.1.1. Test Verisi ile Yazılım Testi (Birim Test 1) 36
4.1.2. Test Verisi ile Bağlantılı İlk Ürün Testi (Birim Test 2) 36
4.1.3. Test Verisi ile Entegre Test (Sistem Test 1) 37
4.1.4. Ait olunan Modülle Gerçek Veri ile Test(Sistem Test 2) 38
4.1.5. Tüm Ürün Testi (Final Sistem Test) 38
4.2. Kabul / Acceptance Test Yapısı 39
4.3. Bakım Yapısı 40
4.4. Denetleme Yapısı 40
5. PROJE YÖNETİMİ 42
5.1. Proje Yönetim Metodolojisi Nedir ? 44
5.2. Proje Yönetimi ile Geliştirme Yaşam Döngüsü İlişkisi 45
5.3. Metodolojinin Dinamiği Nasıldır ? 46
5.4. Proje Tanımı Nedir ? 47
5.5. Proje Tanımlamada Kullanılabilecek Bileşenler 47
5.6. Proje Tanımı Kimlerle ve Nasıl Geliştirilmeli 48
5.7. Proje Aktivitelerinin Geliştirilmesi 49
5.8. Proje Aktivitelerinin Tanımlanması 51
6. KONFİGÜRASYON YÖNETİMİ 52
6.1. Genel Konfigürasyon Yönetimi Kavramları 52
6.1.1. Proje Yönetim Destek Fonksiyonları 53
6.1.2. Konfigürasyon Yönetimi Organizasyonu 53
6.2. Konfigürasyon Yönetim Planı (KYP) 55
6.3. Yazılım Konfigürasyon Yönetimi 57
6.3.1. Bir Yazılım Konfigürasyon Yönetimi Senaryosu 60
6.3.2. Konfigürasyon Yönetim Sistemlerinin Bileşenleri 61
6.3.3. Ana Hatlar / Tanımları 'Baseline' 62
6.3.4. Yazılım Konfigürasyon Öğeleri 65
6.4. Yazılım Konfigürasyon Yönetimi Veri Tabanı 67
6.4.1. Değişiklik Veri Tabanı Rolü 68
6.4.2. Tanımlar ve İçerik 70
6.4.3. Yazılım Konfigürasyon Yönetim Özellikleri 71
6.5. Yazılım Konfigürasyon Yönetim Prosesi 73
6.5.1. Yazılım Sistemlerindeki Nesnelerin Belirlenmesi 76
6.5.2. Versiyon Denetimi 76
6.5.3. Değişiklik Yönetim ve Kontrolü 77
6.5.4. Konfigürasyon Denetimi 80
6.5.5. Raporlama 82
SONUÇ 83
KAYNAKLAR: 87
ÖZGEÇMİŞ 89
EKLER 90
EK 1 : Ekran Örnekleri 90
EK 2 : Uygulama Kaynak Kod Örnekleri 105

GİRİŞ

Çalışmamızdaki temel amacımız yazılım geliştirme süreçlerimizde sağlıklı ve bilimsel bir Yazılım Mühendisliği yaklaşımı geliştirmektir. Yazılım Geliştirme süreçlerimizde bizlere sıkıntı veren konuları Konfigürasyon Yönetimi planlamalarımızla yazılım geliştirme açısından bir çözüm yaklaşımı ile sonuçlandırabilmektir.

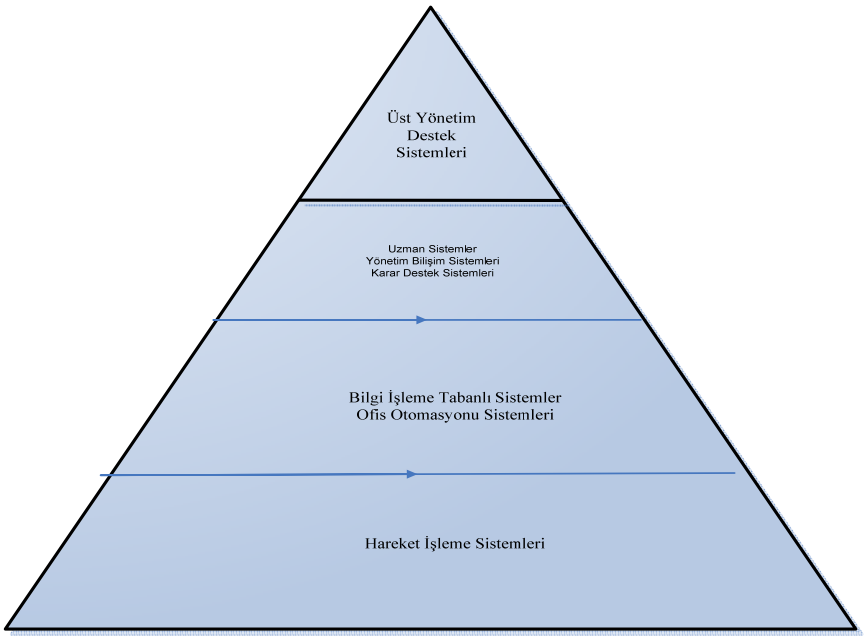
Bu amaca uygun olarak da tüm yazılım konfigürasyon öğeleri ve kavramlarına bir bütün olarak açıklık getirebilmektir.

Yazılım Konfigürasyon Yönetiminin de bir yazılım geliştirme çözüm yaklaşımı olduğunu elde ettiğimiz sonuçlar ve deneyim olarak ortaya tanımlı olarak koyabilmek istemekteyiz.

1. BİLGİ SİSTEMLERİ KAVRAMLARI

1.1. Bilgi Sistemleri Tipleri

Bilindiği gibi Bilgi Sistemlerimizi İŞ'imizin farklı amaç ve gereksinimlerine göre geliştirmekteyiz. Gereksinimlerimiz ve amaçlarımıza göre tasarlananan Bilgi Sistemlerini, geliştirme hedeflerine göre sınıflandırabiliriz. Bu sınıflandırmayı yaparken asıl amacımız yapısal bir yaklaşım göstermektir. Böylece yoğun Bilgi Teknolojileri kullanımı içeren bu model yapılanmayı ana amacımıza uygun Sistem Geliştirme faaliyetlerimizde kullanabiliriz. Ayrıca bu yapısal yaklaşımı da anlaşılabilir yapmak ve mevcut hiyerarşik yapıya uyarlayabilmek için aşağıdaki gibi bir piramit şekil gösterimle özetlemekteyiz.(Şekil 1.1)



ŞEKİL 1.1 Yapısal Bilgi Sistemleri Piramidi

Şekil 1.1 Yapısal Olarak Bilgi Sistemleri Piramidi .

1.2. Hareket 'Transaction' İşleme Sistemleri

Sayısal olarak manuel yöntemlerle işlenemeyecek , değerlendirelemeyecek kadar çok yüksek hacimde olan rutin müşteri işlemleri , fatura verileri , bordro değerleri gibi verilerin bilgisayarlarla işlenmesi için tasarlanan ve geliştirilen sistemler bütünüdür. Veriler yapısı gereği sistemlere insanlar tarafından girdi olarak verilirler. Verilerin çok miktarda ve rutin yapıda olmasından dolayı bilgisayarlarda işlenmesi ile ciddi zaman tasarrufu hatasız işlem yapılarak sağlanır. Bu yapıdaki sistemler sayesinde bankalar, resmi kurumlar , uçuş sistemleri , hizmet ve üretim sektöründeki kuruluşlar kendi hayati fonksiyonları için dış ortamlardan gerekli olan girdi niteliği taşıyan amaç verilerini sistemlerine temin ederler. Bunları günlük rutin faaliyetlerini devam ettirebilmek için işleme tabi tutarlar. Mal, ürün ve hizmetlerinin üretimde kullanacakları temel veri girdisini oluştururlar. Böylece uygun bir yapıda kendileri için gerekli olan operasyonel verileri kesintiye uğramaksızın sağlarlar.

1.3. Ofis Otomasyonu ve Bilgi İşleme Tabanlı Sistemler

İki seviyede bu tip sistemler incelenebilirler. Ofis Otomasyonu Sistemleri veri ile uğraşanlara verileri analiz etmekten çok oluşan verileri bir paylaşım ortamında kullanımı sağlarlar. Bu şekilde verinin ilgili gruba yayılmasını ve paylaşımını sağlarlar. Genel yapıda kelime işlemciler, çalışma tabloları , masaüstü ürünler, anlık haberleşme platformları , elektronik mesajlaşma , elektronik takvim ve zaman düzenleyiciler , sesli mesajlar ve video konferans sistemleri bu kapsamdadır. Bunun dışında özellikle profesyonel çalışanlar, mühendisler, doktorlar ve araştırmacılar ise Bilgi İşleme Tabanlı Sistemlerde

yeni bilgi oluşturması ve bunları büyük organizasyonlarda farklı coğrafyalarda karşılıklı paylaşım düzenine dayanan sistemlerle çalışırlar.

1.4. Yönetim Bilişim Sistemleri

Yönetim Bilişim Sistemleri (YBS), Hareket İşleme Sistemlerinden daha çok bilgisayar sistemlerinden yararlanarak hareketleri de işleyerek kullanıcılar için bir amaca uygun paylaşım sağlayacak yorumlanabilir , sınıflandırılmış bilgi yapısı oluşturan , ilişkilendirilerek değerli bir anlam kazanmış sistemler topluluğudur. Bu yapıda Yazılım, Donanım , İnsan , Karar Vermeye yardımcı olan analizleri yapılmış çok geniş aralıkta organizasyona uygun bir Sistem yapısı söz konusu olacaktır. Bu yapıyı sağlamak için sistemler ortak bir Veri Tabanından verileri paylaşarak tanımlı bir bilgi kümesi oluşturmaktadır. Kesin ve tam olarak bir üründen daha çok bilgiye erişim yapısını tanımlayan genel bir kavram olarak Yönetim Bilişim Sistemleri tanımlanmaktadır.

1.5. Karar Destek Sistemleri

Daha üst düzey olarak bilgisayar yardımı ile karar vermemize yardımcı olan sistemler bütünüdür. Alışılagelmiş YBS'lerden temel farkı ise yapılarında kararlara destek olacak olan İş Zekasına uygun olan mekanizmalarla programlanmış bir yapının bulunmasıdır. Karar vericilerin, yöneticilerin tercihlerine ve önceliklerine göre İş Zekasındaki yapıya uygun olarak tasarlanmışlardır.

1.6. Yapay Zeka'ya dayanan Uzman Sistemler

Yapay Zeka sistemleri, genel olarak kendiliğinden hareket edebilen özel geliştirilmiş sistemlerdir. Bu yapıda kendiliğinden hareketi bu özel sistemlere

tanıtan programlanarak tanıtılmış işlem zincirleri ile bir önceden öğrenilmiş kazanım söz konusudur. Bu yapıda 2 temel konu esas olmaktadır. Doğal olarak sistemlerin konuşma dillerini anlayabilmeleri ve problem karşısından mantıksal olarak bir reaksiyon gösterebilme yeteneğine sahip olmaları beklenmektedir. Uzman Sistemler ise ek olarak bir Bilgi Edinme Tabanına göre özel donanım ve algıları ile kazandıları deneyimlerle bir sorunu çözebilme yeteneğine sahip sistemler bütünüdür. Bu sistemler aslında gelecekte tüm Sistem Geliştirmeciler ve Analistler tarafından hedef sistemler olarak üzerinde yoğun çalışılan ilgi alanlarıdır.

1.7. Bilgisayar Destekli Ortak Çalışma ve Paylaşım Ortamlı Sistemler

Henüz bir son ürün olarak sunulmamış üzerinde ortak çalışma yapılan geliştirme faaliyetlerinde paylaşılarak kullanılan tüm sistem yapıları bu kapsamdadır. Farklı coğrafya ve zaman dilimlerinde olup ortak çalışma, paylaşım ve sonuç isteyen entegre çalışma platformları olan sistemlerdir. Genel yapısında global olarak faaliyet gösteren maliyet optimizasyonu , zaman , bilgi ve yetenek paylaşımı esasına dayanan sistemlerdir. Birbirine Internet ve/veya Özel ağlarla bağlanmış bilgisayarlar yardımı ile özel çalışma platformları sağlayan yazılımlar , yazılım araçları ile sistem yapılanması sağlanmıştır. Yüksek teknoloji ve kullanım yetenekleri ile desteklenmektedirler.

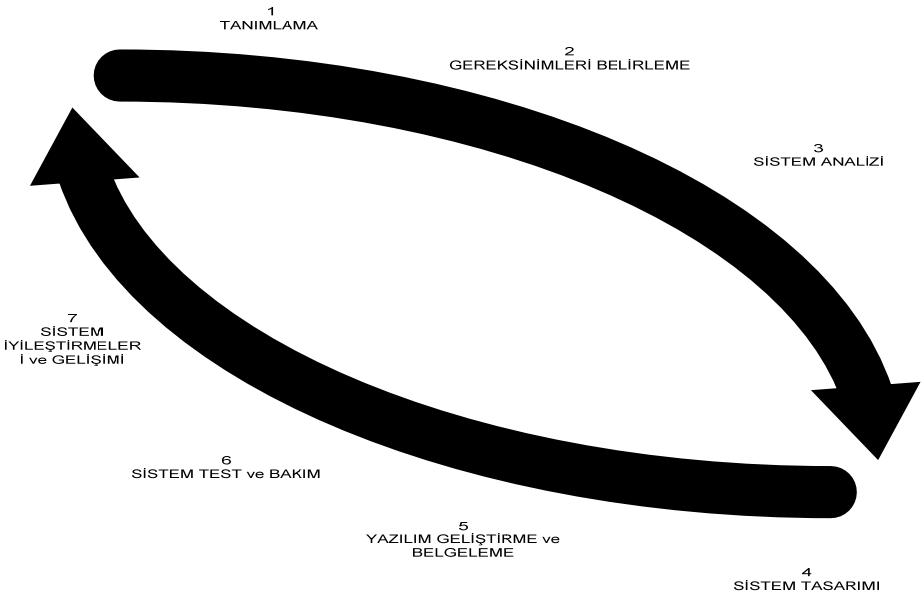
1.8. Üst Yönetime Destek olan İzleme, Anlık Raporlama Sistemleri

Stratejik düzeyde karar destek sağlayabilmek amacıya çok özel tasarlanmış olan sistemlerdir. Kendine özgü raporlama, sunum ve gösterim kabiliyetlerine

sahip doğrudan sonuç odaklı olarak Üst Yönetimin kararlarında esneklik , hız , hasassiyet sağlamak üzere işe özel yapılandırılmış sistemlerdir.

2. SİSTEM GELİŞTİRMENİN YAŞAM DÖNGÜSÜ

Sistem Geliştirmenin Yaşam Döngüsü (Şekil 2.1) ile kullanıcı ve iş analizi yapanları özel olarak bu döngü içinde yer alan unsurlar olarak irdeleyeceğiz.Temel amacımız ise Yazılım Mühendisliğinde Uygulama Geliştirmede yapısal olarak gelişimizi sağlayacak olan 'Değişiklik İsteklerimizi ve Değişiklikleri' yönetmemizi bir disiplin içine alacak olan 'Yazılım Konfigürasyon Yönetimi' prosedürlerimizi Yazılım Yaşam Döngümüz ve unsuruları içinde gerekliliği açısından net bir platforma yani çözüm ortamına yaklaşımımız olarak kavramsal olarak yerleştirmektir.



Şekil 2.1 Aşamalı Sistem Geliştirme Yaşam Döngüsü

2.1. Amaç,Durum ve Problemi Belirleme

Bu fazda sistem geliştirme yaşam döngüsü içinde amacımıza uygun olarak durumu ve problemi belirmek için analizler yapılmaktadır. Bu analiz aslında projenin kalan kısmının sağlıklı bir yapıda başarı ile yürümesi için oldukça kritik öneme sahiptir. Kimse zaman , kaynak ve para harcayarak çalıştığı bir projede problemi tam olarak tanımlayamadığı için başarısız olmak istemez.

Dolayısı ile ilk aşamada analistlerin proje resmini tam olarak net çizebilmesi herşeyden daha önemli bir olaydır. Bu çalışma ilk aşamada zaman kaybı gibi görünmektedir. Fakat başarılı projelerle sorunlu olup başarısız olan projeler arasındaki ilk ve en temel fark olan başlangıç noktası olması açısından görev kritik bir unsurdur. İş ve proje deneyimi yeterli olan ekiplerle çalışmaya başlayan analistler bu başlama unsuruna her türlü ürün geliştirme ve zaman baskısına rağmen direnç gösterirler. Veri akışı ve tümleşik Veri Otomasyon sistemi açısından da benzer mekanizmanın özellikle projenin ilerleyen geliştirme fazlarında 'Konfigürasyon Yönetimi' mekanizması açısından ele alınması da benzer değerde bir kritik unsur olarak projelerde karşımıza çıkmaktadır.

Tasarımcılar ve analistler projelerde rekabetçi olarak endüstriyel standartlara uygun olarak İş'lerin yaşam döngüsünde gelişmesi için fırsatları değerlendirirler.

En önemli bileşen, İş Geliştirmede amacı net olarak tanımlamaktır. İlk aşamda bu çalışmalara analistler, kullanıcılar, proje koordinasyonunda görevli proje yöneticileri katılarak analiz çalışmalarına başlarlar. Bu

aşamadaki aktiviteler kullanıcılarla işi konuşmak, elde edilen verileri özetlemek, proje hedefini özetlemek ve sonuçları belgelemek şeklindedir. Bu aşamada oluşan ürün ise özetlenen amaçları ve iş tanımını içeren fizibilite raporlarının hazırlanmasıdır. Yönetimde bu raporlara göre projenin devam edip etmemesi kararını verecektir.

2.2. Bilgi İsteklerini Belirleme

Bu aşamada ise tespit edilen kullanıcıların katılımı ile analistler gereksinim duyulan bilgileri toparlarlar. Kullanılan araçlar yardımı sayesinde konu ile ilgili olarak etkileşimli biçimde iş bilgileri alınır, örneklemeler yapılır, veri yapıları incelenir, sorgulamalar tamamlanır, prototip model için ilk örneklemeler ve ön-tasarımlar yapılır.

Bu amaca uygun olarak Hızlı Uygulama Geliştirme ‘Rapid Application Development, RAD’ ortamlarından ve araçlarından faydalanmak metodu en çok tercih edilen yaklaşımdır. Bu aşamada İş analisti istenen işin ne olduğunu , çevresini ve kapsamını tam olarak anlamaya , kavramaya ve tanımlamaya çalışmaktadır.

2.3. Sistem Gereksinimlerinin Analizi

Özel teknik ve araçlarla Analistler gereksinimleri belirlemeye çalışırlar , bunları belgelerler. Veri akış diagramı yardımı ile Girdileri, Prosesleri, Çıktıları iş fonksiyonlarına göre yapısal bir grafik gösterimle belgelerler. Veri Akış şemaları ile sistemde kullanılan veri öğlerinin Veri Sözlüğünü ‘Data Element Dictionary’ oluştururlar.Yapısal karar analizleri ile Karar Tablolarını ve Karar Ağaç yapılarını oluşturular.

İş analistleri döngünün bu fazında Önerilen Sistem özetini Fayda/Maliyet Analizi, alternatifler ve öneriler listesini içerecek şekilde hazırlarlar.

2.4. Önerilen Sistemi Tasarlama

Analistler topladıkları bilgilerle Bilgi Sisteminin ilk mantıksal tasarımını tamamlarlar. Tasarlanan sistemin veri ve tasarım öğelerinin doğrulama testlerini yaparlar. Kullandıkları tekniklerle etkin veri giriş tasarımları ile temel ekran tasarımlarına sistemin dönüşmesini sağlarlar. Bir anlamda ilk Sistemin Dışsal Tasarımını tamamlarken bunun doğru çalışan bir forma gelmesini, veri ilişkileri ile iş akışının düzgün olmasını test ederler. Kullanıcı ve ekran arayüzlerinin tasarımları tamamlanır. Veri Yapısının ve veri öğeleri arasındaki ilişkilerin tasarımları, iyilemeleri yapılırken sistemin ekran ve diğer medyalara oluşturacağı çıktıların tasarımları yapılır.

Son olarak da tasarım kontrolleri, yedekleme prosedürleri ile sistem ve veri korumaları ile ilgili düzenlemelerle ele alınır.Tüm temel olacak girdi-çıktı taslakları, veri yapısı özellikleri, tanımları, işlem ve proses detayları, karar yapıları ve ağaçları, iş akış diagramları, sistem şemaları, standartlar el kitapları ve tanımlanan tüm rutinlere ait belgelemeler tamamlanarak ürün paketi şekline getirilir.

2.5. Yazılım Geliştirme ve Belgeleme

Analistler Yazılım Geliştirmesi yapacak takımlarla çalışmalara başlarlar. Yazılımın gereksinim duyacağı İçsel Tasarımlar yapılır. Çeşitli yapısal yaklaşımlarla ön-kodlama teknikleri kullanılarak tasarımın form değiştirerek kodlanan bir yazılım ürünü şekline gelmesine başlanır. Oluşan teknik

kodlama kısıtları, uyumsuzlukları bulunan ve tanımlanan yeni uygulama standartları ile giderilir. Uyum içinde kendi içerisinde tutarlı bir kod bütünü şekline yazılım ürününün gelmesi için gerekli teknik prosedürler hazırlanır. Uygulama yardım menüleri ve içerikleri hazırlanmaya başlar. Yazılım için tanımlanan standartlara uygun yazılım geliştirme standart belgelemeleri yapılarak kodlama işlemleri yapılır. Ürünlerin uyum içinde bütünlükle çalışması, tasarımda hedeflenen kriterlere uygunluğu sağlanır. Özellikle Planların (Konfigürasyon Yönetimi, Veri ve Veri Tabanı Yönetimi , Prosedürler, Migration ve Yürütme Planlamaları vb.) teknik yapıya uyumlu olarak oluşması sağlanır. Bunların çalışabilmesi için yapılanma kurulur.

2.6. Sistemin Testleri ve Bakımları

Oluşabilecek tüm sorunların giderilebilmesi , problemlerin yakalanabilmesi için çeşitli seviyelerde testlerin yapılmasına başlanır. Bu test sonuçlarında oluşan tespitler belgelenerek , düzeltilmek ve tekrar test edilmek üzere Yazılım Üretim noktasına ürünler iletilir. Birim testleri ve çeşitli seviyedeki sistem testleri icra edilerek tüm yazılım sisteminin sorunsuz , uyumlu ve entegre olarak girdi , çıktı , prosesler açısından çalışabilmesi sağlanır.

Bakım gereksinimlerine karşın iş akışı ve bunların oluşturacağı Değişiklik ve Konfigürasyon Yönetim prosedürleri Planlar çerçevesinde uygulanmaya ve tanımlanmaya başlanır.

Böylece sistemin belirli bir güven ve güvenirlikle çalışabilmesi için yazılımın yaşamına uygun bir platformda başlamasının garantisi kalite güvencesi ile sağlanmaya çalışılır.

2.7. Sistemin Yürütülmesi ve Değerlendirilmesi

Analistler bu aşamada hayata geçen sistemin kullanıcılar tarafından kullanılması için gerekli olan çalışmaları yaparlar. Hazırlanan Eğitim Planları çerçevesinde eğitimler tamamlanır , kullanıcılar, teknik personel ve görevliler eğitilirler. Sistemlerde ürünler kabul ortamlarından gerçek ortamlara taşınır . Gereken hazırlık verileri ve varsa kodlama , veri düzenlemeleri , taşımaları , veri uyarlamaları yapılır. Sistemler gerçek ortamlarında platformları, veri tabanları , katalogları , yedekleme prosedürleri vb. tamam edilerek çalışır şekle getirilir. Oluşan uygulama eksikleri düzeltilerek gerekiyorsa iyileştirmeleri , ayarlamaları yapılır. Bu işlemler genellikle döngüsel olarak sorunlar varsa çözülene dek tekrar edilir. Sistemler kademeli olarak gerçek kullanıcılara Yürütme Planlarına uygun olarak devir edilerek nihai yazılım ürünü olarak yaşama alınırlar. Eşlik edilerek ve gerekli makul zamanda sistemler izlenerek ürünler öncelikle pilot kullanıma ve daha sonra tüm fonksiyonları ile kullanıma alınırlar. Gerekli olan varsa ayarlama , entegrasyon ve tutarlılık uygulamaları ürünler üzerinde yapılır.

2.8. Bakımın Sistem Üzerindeki Etkileri

Sistemlerin kuruluşları tamamlanıp yaşamaya başladıkları andan itibaren onları güncel tutabilmek içinde bir çaba kendiliğinden başlar. Yönetim Bilişim Sistemleri için tipik kuruluşlarda yaşam döngüsünün yaklaşık %60 gibi bir kısmı mevcut sistemlerin bakım faaliyetleri ile geriye kalan yaklaşık

% 40 gibi oranı da yeni sistemleri geliştirmek ve diğer faaliyetler için harcanmaktadır. Çok az sistemde bu bakım gereksinimi olmadan çalışma söz konusudur. Bu özellikle büyük ve karmaşık yapıdaki sistemlerde çok daha fazla oranda bakım gerektiren yapıdadır. Bunun en önemli 2 temel sebebi vardır. Birincisi , sistemlerin yaşayan bir canlı gibi olmasıdır. Diğeride tüm yaşayan sistemler gibi günün değişen koşullarına ve gereksinimlerine sistem yaşamak istiyorsa , canlı kalmak istiyorsa kendini uyarlamak zorunda olmasıdır. İşte burada bu uyarlama, bakım , ek geliştirme isteklerinin bütünü aslında sistem için birer 'değişiklik isteği' demektir. Bu sistem yapısı içinde de bir 'Konfigürasyon Yönetim Planı' ile bu değişikliklerin ele alınması , yapılması , uyarlanması ve izlenmesi şarttır. Özellikle gelişen Bilgi Teknolojileri olanakları ile ivmelenerek hız kazanan oldukça karmaşık yapıdaki sistemlerimiz için bu yapısal yaklaşımı tüm işlerimizin içine gömülü bir şekilde sistem yaşam döngüsüne eklemeliyiz.

2.9. CASE Tool/Araçlarının Kullanımı ve Sebepleri

Bilgisayar Destekli Yazılım Mühendisliği 'Computer Aided Software Engineering' araçlarını ve imkanlarını büyük hacimli Bilgi Sistemleri projelerimizde aşağıdaki ana amaç ve sebeplerden dolayı işlerimize entegre olarak kullanma zorunluğumuz vardır.

2.9.1. Analistlerimizin Üretkenliğini Arttırmak

Sistem Analistleri grafik gösterimlerle , analiz, tasarım ve karmaşık veri yapılarına sahip veri tabanı tanımlarını, uygulama tasarımlarını modellemelerini temin eder. Bu görünen analizlerle , modelleme ve

gösterim notasyonları ile projelerin anlaşılırlığı ve paylaşılabilirliği artar. Yapılan çalışmalarda kolaylıkla değişimler uygulanabilir , paylaşılabilir ve takip edilebilir şekle gelir. Bu yapılanma ve uygun teknik araçlarla doğrudan Analistlerin verimliliği artmaktadır.

2.9.2. Analistlerle Kullanıcı İletişimini Geliştirmek

Sistem yaşam döngüsünün kullanıcı ve iş analistleri arasında mükemmel bir şekilde oluşması için her türlü CASE aracı kullanımı iletişim için gereklidir. Bu araçlarla oluşan destekle gelişen bir iletişim ortamı sağlanacaktır.

2.9.3. Sistem Yaşam Döngüsü Aktivitelerinin Sistem Entegrasyonu

Bir fazdan başka bir faza iş akışı geçerken yaşam döngüsü entegre olarak devam etmelidir. CASE araçları kullanışlı birçok geri besleme ve iterasyon adımı ile istekleri sisteme entegre olarak görmemizi , takip etmemizi sağlar. Buda döngünün entegre olarak oluşmasına yardımcı olmaktadır.

2.9.4. Doğru , Hassas Değişiklik ve Bakım Faaliyetlerini Başarmak

Bakım ve değişiklik isteklerini CASE araçları en doğru ve anında sisteme adapte etmemizi sağlayan önemli bir yardımcıdır. Bu sayede kullanıcı ve analistlerin sistem üzerindeki değişiklik ve değişiklik isteklerinin etkilerini takip etmesi , tespit etmesi sağlanır. Bu değişikliklerin cross-referans raporlama olanakları ile de CASE araçları yardımı ile elde edilmesi gelişmiş düzeyde bir Konfigürasyon Yönetim

Planına sahip olmamızı temin eder. Böylece başarı ile yönetebileceğimiz bir plana sahip oluruz.

2.10. Tersine ve Yeniden Yapılanma olarak Yazılım Mühendisliği

Özellikle uzun süre önce geliştirilmiş, görev yapmakta olan sistemlerin ve sistem bileşenlerinin tersine ve yeniden yapılandırılarak incelemesi için yapılan yazılım mühendisliği çalışmalarıdır. Buda alışılagelmiş olan sistemlerin çalışma düzenlerini ve modüllerinin çalışma özelliklerinin detaylı incelenmesi işlemidir. Bu sistemlerin tüm detayları bilgisayar destekli yazılım araçları ile tekrar analiz edilir, yapılandırılır ve incelenir.

Bu Mühendislik çalışmaları genelde iş tanımı için yeniden yapılandırma için yapıldığından dolayı İş İşleme Yeniden Yapılanma 'Business Processing Reenginering,BPR' olarak adlandırılır.

Bu çalışma aslında bir yazılımı kodlama çalışmalarının tam tersi olarak uygulanmasıdır. Varolan ve çalışan yazılım ürünlerinin bu çalışma sonuçlarının Analistler tarafından geriye doğru analiz edilmesi çalışmasıdır. Burada analistler tespitlerini doğru ve yeni sistemler için yapıcı yapabilmeleri için oldukça önemli bir anahtar görev alırlar.

Sağlanan yazılım araçları ve ortamları ile ürüne ait

- Veri Yapıları , tanımları , elemanları , kayıt desenleri, öğeleri
- Ekran tasarımları , ekran girdi-çıktıları
- Raporları ve bunların formatları

- Program ve modüllerinin ilişkisel bağları , hiyerarşileri
- Tüm Veri tabanı tasarımı ve ilişkileri

elde edilir.

Bu araçlar aslında çalışma prensibi açısından ürününün üzerinde bir arabağlantılı kayıt sistemi gibi çalışır. Tüm teknik detay ve özelliklerini dinleyerek , analiz ederek , kontrollü olarak elde etmeye yararlar. Sonra da elde edilen bilgiler sınıflandırılarak sistem ve sisteme ait yaşam döngüsü hakkında geri kazanılmış bir yapı oluşur.

2.11. Nesneye Dayalı Sistem Analizi ve Tasarım

Hızlı gelişen yazılım geliştirme ortamlarında yazılım geliştirme faaliyetlerini yapısal bir yaklaşımla çok daha iyi bir şekilde gerçekleştirmek için yapılan analiz , tasarım ve çalışma platformlarıdır.

Karmaşık yapıda çalışan Bilgi Sistemleri için süregelen bakım , uyarlama ve yeniden tasarım fazlarında nesneye dayanan sistem analizi ve tasarımı yaklaşımı kullanılmaktadır. Bu yaklaşım modelleme olarak evrensel modelleme dili (Universal Modelling Language) ve case modelinde de sistemi altparçalara ayırma yöntemini kullanır.

Klasik prosedürel programlamadan farklı bir yaklaşım gösterir. Her bir nesneyi sistemin bir öğesi/parçası olarak tanımlar , ele alır. Tanımlanan her nesne ayrıca sistemde gerçek bir olay veya öğe olarak rol almaktadır. Nesneler müşteriler, veri elemanları , siparişler,

vb. olabilirler. Nesneler sınıflarına göre gruplanarak en uygun çoklu kullanım ve bakım için tasarlanırlar. Bir sınıf kümesi ise nesnenin bulunduğu sınıfta paylaşılarak kullanılacak biçimde özelliklerle donatılırlar.

3. YAZILIM MÜHENDİSLİĞİ UYGULAMASI

3.1. Yazılım Mühendisliği ile Kalite Güvencesi

Kalite , Bilgi Sistemleri tasarımcıları ve analistleri için en kritik konulardan biri olarak her zaman gündemde olmuştur. Kalite güvencesi kavramını dikkate almadan yapılan her türlü Bilgi Sistemi faaliyetleri iş riski açısından çok ciddi bir potansiyel taşımaktadırlar. Temelde Yazılım Mühendisliğinde 3 temel yaklaşımla kalite güvencesi garanti altına alınabilmektedir:

- Top-down yaklaşımla sistemlerin tasarımını toplam kalite güvencesine almak,
- Uygun yazılım araçları ile yazılımları belgelemek
- Yazılımları test etmek, bakımını yapmak ve kontrollü denetlemek.

Kalite güvencesinde ise 2 temel yaklaşım modellemesi uygulanmaktadır. İlkinde Bilgi Sistemleri kullanıcıları için yazılımın kalitesini ölçmek ve değerlendirmek tek başına önemli olmaktadır. Diğer yaklaşımda ise daha az maliyetli bir yol olarak kullanıcı şikayet ve sorunları gelmeden ilk aşamada sorunları tespit ederek çözümlemek ve o şekilde ürünleri kullanıma sunmak şeklindedir.

Ürünlerin kullanıma alınıncaya kadar ki yaşam evrelerinde birçok geliştirme adımı vardır. Bunlar için çok ciddi maliyet ve zaman harcanması yapılmaktadır. Ürünlerin kullanım öncesinde belirli bir güven aralığında kalite güvencesi sağlanarak kullanıma alınması hem sonraki hemde ürün

yapılırken ki maliyeti açısından hayati önem taşımaktadır. Bu nedenle sistem analistleri 3 temel yaklaşımla Kalite konusunu güvence altına almaya çalışırlar.

3.1.1. Toplam Kalite Yönetimi Yaklaşımı

Toplam Kalite Yönetimi (TKY) temelde sistemlerin geliştirme aşamalarında gerekli olan bir yaklaşımdır. TKY nin ana elemanı ise organizasyonel yapıda genel olarak çok yönlü ve kapsamlı bir kalite çabasının sağlanmasıdır. Bu yapıda müşteri hedefine , ürün amacına , liderliğe, planlamaya, sürekli gelişime velhasıl her unsurun kalite çabasına katkısını ister. Analiz ve tasarım aşamalarında sadece bu kavramın peşinde olmayıp dinamik olarak her aşamada kalite kavramı yaşam döngüsünde etkin ve etken olarak hedef amacın içindedir.

3.1.2. 6 SİGMA

6 Sigma'nın gelişi kalite yönetimini de değiştirmiştir. Her analistin 6 Sigma'nın farkında olması, bunun prensiplerini uygulamaya dikkat etmesi esasına dayanmaktadır. Orjin olarak Motorola firmasının yapmış olduğu bir uygulama ile bu kavramla dünyayı tanışmıştır. 6 Sigma bir metodolojiden çok kalite kavramını oluşturma kültürü niteliğindedir. 6 Sigma'nın temel amacı da ürün sorunlarını / hatalarını gidermektir. Sadece ürün, servis ve proseslere uygulanmaktadır. 6 Sigma top-down bir yaklaşımdır. Organizasyonun CEO rolünün proje başarısına bu felsefeyi uyarlamasını hedefler. 6 Sigma proje liderine 'Black Belt' adı

verilir. Proje takımı elemanlarına ‘Green Belt’ denir. Bu metodik yaklaşımından çok bir felsefe ve kültürdür.

3.1.3. Toplam Kalite Yönetimi Sorumluluğu

Basit olarak bakınca aslında bilgi sistemlerinin kalitesinden sistem yönetimi ve kullanıcıları direk olarak sorumludurlar. TKY ‘de 2 unsur kalitede belirleyicidir. İlki mevcut yönetimin tam organizasyonel desteği , diğeri de iş için proje yöneticileri ve analistler tarafından verilen kalite taahhütleridir. Organizasyonel destek ile kalite yönetimi Bilgi Sistemleri kalite çemberlerini organizasyonel seviyelerle bilgi sistemlerinin nasıl ve nelerle ilerletileceğini tanımlar.

Böylece Kalite Planları, Yönetimi ve bunlara ait standartlar tanımlanır, şekle gelir. Ardından da bunlar yaşam döngüsü içinde uygulamaya alınır.

Analistler bölümler arasındaki kalite kavramlarını ve standartlarını geliştirip, uygulanmasını temin ederler. Oluşan etkileşimle ürünlerin kalite sorumluluğu bölümler tarafından paylaşılır duruma gelir. Bu şekilde toplamda oluşacak olan ürün, mal ve hizmet sunumunda bir Toplam Kalite kavramı, paylaşımı ve ortak hedefi olarak yaşam döngüsü sistem açısıdan üretim prosesi içinde yer alır.

3.1.4. Yapısal Yürütme Yaklaşımı

En güçlü yaklaşımlardan birisi sistem analist takımlarının rutin olarak yapısal yaklaşımla kalite yönetimi faaliyetlerini yaşam döngüsünde

uygulamasıdır. Yapısal yürütmede eş pozisyonlar sistemleri izleyerek tüm geliştirme, sorun noktaları, problemler ve uygun değişimleri takip ederler.

Bu yaklaşımda en az 4 adet role gereksinim vardır. Bir analist yazılımcı, Yürütüm koordinatörü, yazılımcı veya eş analist ve önerileri toplayacak bir eş uygulamacı.

Her pozisyon yürütüm işleminde özel bir rol alırlar. Koordinatör faaliyetleri kontrol ederek, iş atamalarını yapar, üretimin planlanan zamana uygun devam etmesini temin eder. Yazılımcı ve analistler ürünü sorunu çözecek şekilde gerçekleşetirecek çalışmaları yaparlar. Raportörde kayıtları ve oluşan iz bilgilerini çalışmalara katılarak kayıtlı şekle getirir.

Yapısal yürütüm sistemin ve/veya ürünün tamamlanmasına kadar devam eder. Bu yaklaşımda oluşan kayıt mekanizması kayıtlı iş yapma kavramı açısıdan sistemin yaşam döngüsüne devam ettiği sürece geri besleme mekanizmasında bu kayıtların iyileştirme, değişiklik yapma ve konfigürasyonu başarı ile yönetmek için kullanılır. Böylece toplam kalite temininde de önemli ve değerli bir yaklaşım özelliği taşır.

3.1.5. Sistem Geliştirme ve Tasarımı

Modüler programlama yaklaşımı 2 yöntemle uygulanır. Her birisinin kendi yapısına göre avantaj ve dezavantajları bulunmaktadır. Bu yaklaşımlarda yapıları gereği gerek Konfigürasyon Yönetimine gerekse

Kalite Güvencesi Yönetimine sağladıkları imkanlarla bize pozitif faydalar sağlarlar.

3.1.6. Bottom-Up Tasarım

Bu yaklaşımla organizasyon içindeki bilgisayarlaşma sürecinin en alt seviyesinden ürün seviyesine kadar tüm süreçler ele alınmalıdır. Sistemler arasındaki entegrasyon çalışmalarında sıkıntılar yaşanabilir. Her sistem kendi açısından değerlemeden geçecektir. Bu nedenle tekrar içeren ve dolayısı ile maliyet unsuru olacak çalışmalar yapılacaktır. Bunun dışında sisteme tekrarlardan dolayı bazı veri setlerinin tekrar gereksiz girilmesi söz konusu olabilir. Ayrıca genel olarak organizasyonun toplam amacı ve bütünlüğü görünemeyebilecektir.

3.1.7. Top-Down Tasarım

Daha kolay bir gösterimle yapısal yaklaşımla tüm resmin görünür şekle gelmesi sağlanır. Böylece bütünlük içinde tüm sistemlerin etkileşimle, entegre bir biçimde ele alınması sağlanır. Bu tüm sistem ve alt bileşenlerine toplam bir kalite yönetimi ve güvencesi getirecektir. Temelde sistemi modüler olarak alt sistemlere bölerek ele alan ve analiz eden bir yaklaşımdır. Alt sistemlerle ortak fonksiyonların tek bir kez hazırlanmasını ve paylaşılarak kullanmasını sağlayan yapıdadır. Böylece gereksiz tekrarları önler, maliyet azalmasına ve etkinlik artışına neden olur. Modüler olması sebebi ile kolay entegre edilir ve fonksiyonel parçalarla kolaylıkla izlenebilir bir yapıdadır.

Bu yaklaşımla Toplam Kalite Yönetimi yakın bir işbirliği içinde olumlu bir sinerji yaratmaktadır.Hızlı, etkin, daha uygun maliyetli ve Konfigürasyonu, bakımı yönetilebilir bir sistem oluşumuna imkan vermektedir.

3.1.8. Modüler Geliştirme

Top-down yaklaşımla Modüler Geliştirme birlikte kullanım için uyumlu bir programlama yöntemdir. Bu yaklaşım programlamayı mantıksal, yönetilebilir ve fonksiyonel parçalarla yapmayı önermektedir. Modüllerin yapıları netleşince entegrasyonu kolaylıkla birbirlerinden bağımsız olarak gerçekleştirebilir. Arabirimlerin modüler arasında yapılması birbirinden bağımsız olarak ele alınabilmektedir. İdeal yapıda bir fonksiyon için bir atanmış modül şeklinde yapılanmanın tasarlanması şeklindedir.

Üç ana avantajı yaklaşım bize sağlar.

- Kolayca tasarlanır, kodlanır, takip edilir
- Bakımı ve konfigürasyonu kolayca uygun maliyetle yönetilir. Değişikliklerin tüm sistemler yerine sadece ilgili modüllerde yapılması ile ciddi zaman, para ve emek tasarrufu sağlanır.
- Sistem yapısı kolaylıkla modüller izlenerek kavranabilinir. Fonksiyonlar sayesinde kodları kolaylıkla takip edilir, izlenebilir.

3.2. Yazılım Mühendisliği ve Belgeleme

Planlama ve kontrol sistemler için gerekli olan temel bir unsurdur. Yazılım Mühendisliğinde de planlama, kodlama çalışması başlamadan ele alınıp yapılması zorunlu olan bir işlem adımdır. Bu nedenle amacımıza uygun araç ve bunu destekleyen platformlarla bu planlama basamağını düzenlememiz gereklidir. Elimizdeki yazılım araçları ile de sistemi oluşturan yapıları alt modüller şeklinde oluşturmamız yönetilebilirlik açısından gereklidir.

Bunun yanısıra yapılanları belgelemek de en kritik iş öğelerinden birisidir. Belgeleme çalışmaları özel amaçlı belgelemenin dışında işlem ve işler yapılırken iş içinden kendi yapısı ile oluşan üretilen belgelerin de olması önemlidir. Bu belgelerle sistemin yaşam döngüsü içinde sonraki evrelerinde kaynak değişimlerini de ve gereksinim değişimlerini de sisteme kolayca adapte edilmesini ve takip edilmesini temin edeceğiz. Özellikle oluşan Değişiklik İstekleri ve bunların Konfigürasyon Yönetimi gereksinimlerinde belgeleme çalışmaları yapılmış, bunların tamamlanmış olduğu sistemler uzun süreli bir ömürle daha etkin, az maliyetli ve güncel yaşama adapte olabilen sistemler olacaklardır. Yazılım Mühendisliği CASE araçları ve bunların kullanımı da sistemlerin yaşamları açısından bu yüzden kritik bir önem taşır.

3.2.1. Ön Kodlama /gerçek olmayan (Pseudocoding)

Kullanıcılarla ve istekde bulunanlarla teknik olarak işlemleri yapacaklar arasında bir iletişim arabirimi oluşumuna, yani bir iletişim diline gereksinim vardır. Bu iki yolla yapılabilir. İlkinde tarafların hepsi aynı dili öğrenir ve aynı kavramları kullanır. Bu pratik olarak kolayca

uygulanabilecek bir yöntem değildir. Diğerinde ise arada kavramlar ve ara ürünler konusunda asgari doğru anlaşmayı sağlayacak ön-kod yani gerçek olmayan bir kodlama / dil kullanımıdır. Benzer durum yazılımın kodlanacağı dil içinde geçerlidir. Hızı ve etkinliği arttırmak için arabir kodla algoritmik olarak kodlama yapılırsa, prototip uygulama geliştirmek her açıdan avantaj sağlamaktadır. Bu mantıksal bir bütünle içinde bazı ana kelime ve doğal dil benzeri mantıksal karar verme operatörlerinden oluşan, farzedilen bir gösterime dayanan Bacus Naur Form (BNF) yapısında basit tanımlı bir karşılıklı iletişim yoludur, dilidir.

3.2.2. Prosedür El Kitapları

Büyük organizasyonlarda kullanılan ortak standartları tanımlamayı sağlayan yazılı tanım dökümanlarıdır. Detay bazda standartlaşan kurum içi faaliyetleri yapısal olarak tanımlayan belgelerden, formlardan, görev tanımı, sorumluluk ve her türlü plandan oluşurlar. Sabit ve değişmez yapıda değillerdir. Tam tersine dinamik ve gelişen bir yapıdadır. Gelişimlerini sistem yaşam döngüsü içinde gelen değişikliklerin standartların da geliştirilmesine gereksinimi olursa bu Prosedür El Kitaplarına yansıtılması ile sağlanması şarttır. Böylece Prosedürler de kendi içinde dinamik olarak gelişeceklerdir.

3.2.3. Sözel Yöntemler

Bu yöntemde ise karşılıklı olarak serbest ama kayıtlı bir platformda Kullanıcılar, Analistler, Uygulama Yazılımcıları, Yöneticiler ve tedarikçiler isteklerini tartışarak ele alıp şekillendirirler. Akış şemaları, diagramlar, tablolar, belgeler, prototip uygulamalar, sistem tasarımcıları tarafından tüm katılımcılara sunulur. Alınan sonuçlar ve varılan kararlar bu belgeler üzerinde gerek duyulursa güncel şekle getirilirler. Elektronik ortamlarla ve medya ile katılımcılar arasında paylaşımı temin edilir.

3.2.4. Tasarım ve Belgeleme Tekniği Seçimi

Burada seçilecek olan araçlar , hatırlama teknikleri , üretim araçları seçilen yazılım geliştirme platformları ile uyum içinde koordineli olarak kullanılmalıdır. Teknik seçimi ile ilgili olarak şu kriterlerle hareket edebiliriz.

- Varolan dokümantasyon yapısına uyumlu olmalı
- Tüm organizasyonlar tarafından anlaşılabilir olmalı
- Başvuru yapıldığında hep kullanılabilecek güncellikte olmalı
- Çalışılan sistem boyutu ile uygun olmalı
- Yapısal tasarım kriterlerine uygun bir yapıda çalışabilecek ortam olması çok önemlidir
- Kolaylıkla gereksinim duyulacak uyarlamalar belgeler üzerinde uygulanabilmelidir.

- Güncel yapıda herkesin tanımlı olan erşim güvenlik tanımlarına göre kullanımına uygun olan bir elektronik platformda olmalı
- Yapılan değişiklikler ve düzenlemelerle ilgili Kongfigürasyon Yönetim Planı kapsamında kim , ne zaman ve neleri bu belgede değiştirmiş takip edilebilir olmalı. Ayrıca hangi değişiklik isteği ile ilintili bu uyarlamanın yapıldığı da takip edilebilmeli.

4. TEST, BAKIM ve DENETLEME YAPISI

4.1. Test Prosesi

Tasarımı tamamlanarak yazılım geliştirme yaşam döngüsü süreçleri ile hayata geçen her ürün ve düzenleme, uyarlama, değişiklik yapılan her uygulama parçası mutlaka testlerden geçirilir. Bu testler basit anlama deneme-yanılma ve düzeltme gibi bir metodla sistemlere uygulanamazlar.

Test süreci sistemler sadece geliştirilirken yapılan bir işlem değildir. Yaşayan sistemlerde sonu olmayan bir süreç olarak işin kendi doğasında yer alırlar. Bu nedenle bir kez yapılan ve sonra unutulup kapatılan bir işlem niteliği taşımaz. Test işlemleri genelde sıkıcı, usandırıcı ve sevimsiz bir süreç bütünüdür. Bu şekilde bir yapıda olmasına rağmen titizlikle yapılması kesinlikle çok gerekli olan bir süreçtir. Test işlemleri çeşitli zaman aralıklarında ve tüm sistem modülleri üzerinde yapılırlar. Sistemleri gerçek yaşam ortamlarına taşımadan önce mutlaka kayıtlı takip yapısına dayanan önceden prosedürel olarak tanımlanmış bir Test Planı çerçevesinde azami titizlik, özen ve dikkatle yapılması gereken bir işlemler bütünüdür. Ürünü yapmak kadar önemlidir. Ürünlerin testlerde başarılı olması için mutlaka hem mevcut yerel standartlara hemde global standartlara uyumlu üretilmesi gerekir.

Test çalışmalarında yazılımcıların, analistlerin, kullanıcıların, operatörlerin ve değişik rollerin tamamının önemli görevleri vardır. Test sürecinde gerekli olan kontrollü test ortamları, araçları, uygulamaları ve yetenekleri kalite

güvencesi sağlayabilmek için kullanmaktayız. Tipik olarak test mekanizmasının yapısını aşağıdaki gibi ele alabiliriz.

4.1.1. Test Verisi ile Yazılım Testi (Birim Test 1)

Programın yazılımını yapan kişinin tanımlanan isteklere uygun olarak her bir program parçasını kendi kendine test etmesi esasına dayanır. Yazılımı yapan kişi olarak oluşan Birim Ürünü o anki çevre koşulları ve kendi test verileri ile anlık test etmek demektir. Normal koşullarda test işlemlerini ürünü yapan kişi yapmaz. Fakat bu prensip burada geçerli değildir. Erken ürün yapma aşaması olan bu aşamada yapılan Birim Ürün ilk temel testlerden ürünü kodlayan yazılımcı tarafından teste alınır. Böylece yapılan işin geçerli ve tutarlı sonuçlar üretttiği ürünü yapan tarafından kontrol edilir. Test verileri de bu aşamada test yapabilmek için hem analistler hemde yazılımcılar tarafından hazırlanmaya başlanır. Birim Testlerde azami kontrolü sağlayabilmek ve istenen fonksiyonları gerçekleştirebilmek için kontrollü , tekrar kullanılabilir birim test veri seti de ürün olarak oluşur. Bütün bu test aşamaları hazırlanan Test Planı kapsamında mutlaka yazılı olarak belgelenir.

4.1.2. Test Verisi ile Bağlantılı İlk Ürün Testi (Birim Test 2)

Test verileri ile ilk yapılan kontrol ve testlerden başarı ile geçen ürünler ikinci aşama testlerin yapılması için test grubuna teslim edilirler. Bağımsız olarak birim testlerden geçen ürünler bağlantılı testlere alınırlar. Birbiri ile bağlı olarak test edilebilmeleri için gerekli olan test

planlaması ve test veri setleri düzenlenir. Bağlantılı olan ürün ve alt modüllerin bu testler sonucunda tespit edilen sorunları giderilir. Tespit edilen durumlar belgelenir. Bu şekilde bağımsız yapılan ama bağlantı çalışacak olan yazılım ürünlerinin bir sistem kavramı altında test edilerek birleştirme işlemi süreci başlatılmış olur. Bu aşamayı başarı ile geçen bağlantılı birimler sistem testi aşamasına aktarılırlar.

4.1.3. Test Verisi ile Entegre Test (Sistem Test 1)

Komple olarak test verileri ile sistemin bir bütün olarak testi işlemi aşamasıdır. Daha önceki aşamalarda hazırlanan test veri setleri bu entegre test sürecinde test grubu tarafından kullanılır. Test sistemi, test verileri ile test edilirken:

- Prosedürlere uygun olarak test edilen modüllerin uygun belgelemesinin yapılıp yapılmadığı kontrol edilir, tespit edilen eksiklikler tamamlanır.
- Veri setlerinde yeterli olmayan açıklamalar varsa veya belgeler arasında tutarsız olanlar varsa uygun yapıya getirilir.
- Sistem akışına uygun olarak sistemin yapısının doğru akışla çalışması organize edilir.
- Final belgeler olarak girdi , çıktı ve fonksiyon düzeyinde uygun test edilmiş çalışma düzeni sağlanmalıdır.

Projenin teslimi yaklaştığı için genelikle çok ciddi zaman baskısı altında ve stres altında bu sistem testleri yapılır. O nedenle özellikle test ekibinin çalışması ve yapılan çalışmaları düzenli belgelemesi çok

önemli bir unsurdur. Bu aşamada oluşan hatalar , sorunlar, düzeltmeler, kabul kriterleri, belge içerik ve düzenleri, sistem performans, sistem stres testleri de kapsamlı testin içindedir.

4.1.4. Ait olunan Modülle Gerçek Veri ile Test(Sistem Test 2)

Tüm test aşamalarını tamamlayan sistemleri gerçek verilerle test etme süreci başlar. Bu evrede gerçek veriler test ortamında sisteme girilir, taşınır, oluşturulur. Test sürecinde oluşan test belgeleri, test varyasyonları ve test senaryoları da kullanılarak sistemler bu verilerle test edilirler. Sonuçlara göre oluşan problem ve hatalar giderilir. Sistemlerin gerçek verilerle tutarlı ve uyumlu çalışması sağlanır. Test Senaryoları tüm aşamaları ile eksiksiz uygulanır. Böylece sistem iş akışının sorunsuz ve diğer tüm alt modüllerle uygun çalışması gözlemlenir. Sonuç ürünlerin oluşan girdi ve işlemler sonucunda üretilen çıktılar açısından irdelenmesi yapılır. Bu aşamada test sürecinin bir parçası olup diğer aşamalar gibi açık, net ve Test Planına uygun olarak belgelenirler.

4.1.5. Tüm Ürün Testi (Final Sistem Test)

Test ortamında gerçek verilerle test edilen sistemler hazırlanan Gerçek ortama Konfigürasyon Yönetim Planı çerçevesinde taşınırlar. Artık sistemler gerçek çalışacakları donanım ve yazılım ortamına kavuşmuş olurlar. Bu ortamda ürünlerin pilot çalışması sağlanır. Varsa uyarlama ve detay ayarları yapılır. Sistemlerin entegre olarak uyumlu çalışması gözlenir. Sonrasında gerçek kullanım ve ürünü kullanımını kabul

edecek kullanıcılar için hazır şekle getirilir. Tespit edilen son sorunlar da giderilerek ürün kullanıcı/müşteri teslimine hazır şekle getirilir.

4.2. Kabul / Acceptance Test Yapısı

Tüm testlerden başarı ile geçen sistem ve ürünler kullanıcı/müşteri tarafından daha önceden hazırlanan Kabul Planı kapsamında işleme alınırlar. Bu aşamada baz olarak ele alınması gereken Kabul Kriterleri olarak adlandırabileceğimiz bazı konular vardır.

Bunlar :

- Mevcut İş Tanım belgeleri ele alınarak fonksiyonların tam olarak yerine getirildiğinin karşılıklı olarak teyid edilmesidir. Oluşan uyuşmazlıklar tespit belgeleri ile değişiklik isteği olarak ve/veya problem isteği olarak belgelenir. Böylece tanımlanan değişiklik planları da ilk kez uygulamaya başlanmış olur. Fonksiyon kabulü kullanıcı tarafından yapılır.
- Test sonuçları, belgeleri ve tüm fazlarının üstünden birlikte değerlendirme yapılır.
- Teknik ekip tarafından da (eğer müşteri organizasyonunda varsa) ürünlerin iç ve dış tasarım standartları kontrol edilerek, teslim alınır.
- Test Planlarında tanımlanan performans, stres kriterlerini ürünlerin temin edip etmediğine bakılır. Varsa sorunlar tespit edilir, giderilir. Sorun yoksa mevcut yapıda kabuller belgelerle yapılır.

4.3. Bakım Yapısı

Sistemlerin yazılım ve donanım bileşenlerine uygun şekilde bir bakım kavramı tanımlanır. Burada geliştirilen ürüne göre de planlama yapılır. En temel bakım gereksinimi oluşturacak olan bir Değişiklik İstek Formu yapısı ile bakım sistemi başlayacaktır. Bu esas alınarak diğer sistem yaşam döngüsü içinde sistemin ürettiği tüm belgelerin uyum içinde hareket edebileceği tutarlı bir yapısal veri otomasyonu yapılır. Bakım da en önemli öğe ise KAYIT kavramının olmasıdır. Yani Kayıt altına alınmış , yönetilebilir , konfigürasyonu kontrol edilebilir bir kavram şemasının oluşması temeldir. Bu yaklaşımda esas olan da belge düzeni ile değişiklik isteklerinin kayıt altına alınarak sisteme ait yaşamın her evresinin, tarihçesinin neden , nasıl , ne zaman , kim tarafından , ne kadar zaman ve maliyetle izlenebilmesidir. Özellikle zaman ilerledikçe insan kaynaklarında ve organizasyonlarda oluşan değişimlerde de bu yolla kurumsal hafızanın kişilerden bağımsız olarak sağlanması hedeflenir.

Kayıtlı Bakım Kavramının uygulanan bu Planlar sayesinde kurumların zaman içinde bir çeşit sistem üzerindeki log kayıtları olarak kişilerden ve organizasyonlardan bağımsız bir yapıda entellektüel kurum sermayesi olarak kurum değerleri arasında yerini almasını sağlayacaktır.

4.4. Denetleme Yapısı

Kalite Güvencesini belirli bir güven aralığı ile temin edebilmek için yapılan kontrol fonksiyonlarının tamamıdır.

Güvenirlik tanımına uygun olarak ürün ve gerçekleştirilen fonksiyonların kontrol edilmesi temeldir. Periyotlara bağlı olarak içsel veya dışarıdan gelen bağımsız otoriteler tarafından sistemler ve ürünleri kontrolden geçerler. Tespit edilen sonuçlar genellikle doğrudan Üst Yönetim'e raporlanır. Bu raporların değerlendirmesinde ayrıca pro-aktif olarak alınması gereken önlemler ve önlem zincirleri de öneri olarak yapılmalıdır. Bu öneri demetleri normal sistem yaşam döngüsü kapsamında sistemlerde düzeltici ve düzenleyici faaliyet olarak geri besleme mekanizmasına aktarılır. Böylece yapısal olarak sistemlerin denetleme sonuçlarına göre kendilerini dinamik olarak düzeltebilecek bir mekanizma ile çalışması sağlanır. İyileştirme ve geliştirme çalışmaları içinde objektif ve yönetimden destek almış unsurlar olarak sistem yaşamı ve kalitesi geliştirilmesi için değeri yüksek olan kritik unsurlardır.

5. PROJE YÖNETİMİ

Proje Nedir ? şeklinde bir tanım ararsak karşımıza birçok farklı tanım çıktığını göreceğiz. Aslında her tanım aynı amaçla benzer bir kavramı bize anlatmaya çalışmakla beraber farklı anlatım öğelerini içermektedir. Öz bir tanım yapacak olursak şöyle diyebiliriz.

PROJE , başlangıç ve bitişleri net olarak tanımlanmış ortak bir hedefe ve amaca ulaşmak için süre kısıtı , bütçesi ve kaynakları olan faaliyetlerden oluşan bir eylemler bütünüdür.

Bunları ele almak için bazı kriterler koyarak küçük , orta ve büyük projeler olarak nitelendirebiliriz. Kiriterlerimizi bir tablo ile gösterirsek : (Şekil 5.1)

	KÜÇÜK Proje	**ORTA Proje**	**BÜYÜK Proje**	**DEV Proje**
Süre	1 Aydan kısa	1-4 Ay	4-12 Ay	12 Aydan çok
Maliyet	150 iş günü	150-1500 iş günü	1500-6000 işgünü	6000 iş günü +
Organizasyon	Dar ekip	2-3 alt ekip	10 alt ekip	Yüzlerce ekip
Kompleksite	Az	Basit görünür	Büyük çaplı	Çok grift ve çaplı

Şekil 5.1 Proje Kriter Tablosu

Projenin amacı , hedefe ulaşmak olması sebebi ile işin en başında tespit edilmesi gereken konu neyin proje neyin proje olmadığının net olarak belirlenmesidir. Genelde profesyonel hayatta ticari nitelik taşıyan işler için bu proje kavramı kullanılmakla birlikte , bilimsel çalışma hayatında da bu kavram işlemler ticari olmamakla birlikte çok daha yaygın olarak kullanılmaktadır. Özellikle bilimsel çalışmalara da sponsor yardımı ile alınan maddi araştırma bütçelerinin olması, yani projelere maliyet unsurunun girmesi sebebi ile artık bilimsel araştırmalarda oldukça detaylı ve profesyonelce projelendirilmektedir. Modern proje yönetim teknikleri ile yönetilmektedirler. Projelerin sadece bir aktivite zinciri olmayıp bir devam edegelen süreç olması sebebi ile proje yöneticisi tarafından mutlaka aktivite bitiş-başlangıç kriterleri ile takip edilmesini gerekli kılmaktadır. Proje Yönetiminde de bilimsel bir metod ve modern uygun araçlar kullanmak günümüzde bir iş standardı şekline gelmiştir. Bunun global rekabetin doğal bir sonucu olarak maliyet kontrol değerini karşımıza proje yönetimi süreçleri şeklinde getirdiğini görmekteyiz. Projeler geçici süreçlerden oluşan son hedef olarak varılmak istenen bir sonuçtur. Bu nedenle projelerde mutlaka varılmak istenen hedefler proje bitiş noktası olarak projenin ilk safhalarında somut olarak tanımlanmalıdır. Birçok projede ilerleyen zamanla bu tanım eksikliği projenin bitirilmesine engel olabilmektedir.

Benzer şekilde projenin başarıya ulaşabilmesi için Yönetim , Proje ekipleri ortak olarak ‘Biz bu projeyi niye yapıyoruz ? ‘ sorusunu ve yanıtını net olarak

bilmelidirler. Böylece projede ölçülebilir bir başarı kriteri, hedefli ve ortak bir amaçlı yapılacak tanımlı iş varlığı söz konusu olacaktır.

5.1. Proje Yönetim Metodolojisi Nedir ?

Metod uygulamasının amacını :

- Standartlaştırma
- Çoklu bir disiplin sağlama
- İyi yönetebilme
- Ürün kalite ve teslimlerini güvence altına alabilme
- Belirli bir bütçe kısıtına uyabilme
- Zaman kısıtını sağlayabilme
- Kontrol edebilme
- Rekabetçi olabilme
- Takip edilebilir ve belgelenebilir olabilme
- Tüm proje resmini ortak platformda paylaşabilme

olarak tüm Bilgi Teknolojileri Yazılım projelerinde görebiliriz. Tabiki bu evrensel değerler sadece yazılım projeleri için değil benzer şekilde tüm projeler için metodik olarak uygulayabilmekteyiz. Burada asıl sorunda Proje Yönetimi içerdiği değişken fazlalılığı ve iş kolu çokluğu nedeni ile tek ve standart bir çözüm yolu sunamamaktadır. Her organizasyonun kendi yapısında, kendi işine göre tasarladığı metodolojiyi proje bazında yeniden ele alıp , değerlendirip işine en

uygun çözümü bulma gerçeği olduğudur. Daha sonrada bu doğrultuda proje başlamalıdır.

Proje Yönetimi süreklilik arz eden devamlı bir süreç ve bağlantılı süreçler bütünüdür. Aktivitelerin alt alta sabit sıralandığı bir lineer yapı söz konusu değildir. Tüm süreçler belirli ilişkilerle birbirine bağlantılı olarak içiçe girmiştir. Bunları tamamen bağımsız düşünmek ve değerlendirmek olası değildir.

Benzer şekilde denetleme , test etme ve tespit edilenleri raporlama tüm sistemde sürekli devam eden bir iç süreç olarak projelerde vardır.

Metodoloji bu noktada proje yönetim iş akışını göstermekle birlikte , projeye özel geliştirme faaliyetlerinin detaylarını göstermez. Bunlar uygulama detayı olarak projelerin içinde sistem yaşam döngüsü ile hayata geçmektedirler.

5.2. Proje Yönetimi ile Geliştirme Yaşam Döngüsü İlişkisi

Yaşam döngüsü ile projelerin uygulama aşaması aynı anda gerçekleştirilebilir. Proje Yönetimi ile proje geliştirmede kullanılacak olan metodlar birbirinden bağımsızdır.

- Metodolojinin Uygulanabilirliği

Proje Yönetim metodolojisi genel tanımlarla tüm projelere uygulanabilecek şekilde tanımlanmıştır. Şirket politikaları ile hangi projelere uygulanabileceğine yönetim karar verir. Önemli olan bu karar/lar sonrasında projelerde PY metodolojisi kullanarak projelendirme yapılmasıdır.

- Metodolojinin Yapılanması

Büyük ve karmaşık yapıdaki projelerde yönetim açısından etkin, kontrollü ve yönetebilir proje aktivitelerin olması sağlıklı proje yapısı için çok kritik önemdedir. Bu tip projelerde yönetimsel kontrol ve izleme çok daha önemlidir. Küçük projelerde ise daha çok uygulama temelli iş gerçekleşmelerine dikkat etmek etkinliği artıracaktır.

5.3. Metodolojinin Dinamiği Nasıldır ?

Proje Yönetimi her sorunu çözen ve yoluna koyan sihirli bir yaklaşım içeren gizemli bir çözüm yolu değildir. Başarı ile projelerin yapılabilmesi için gerekli olan iş adımlarını önermektedir. Başarılar Proje Yönetim Metodolojisinin uygulanabilir olması ve uygulanması ile orantılı olmaktadır. Projenin başarısında iş onayı alınmadan önce yapılan araştırmalar rol oynar. Bu araştırma form ve raporlaması sonuçları da detayların oluşması için projeye başarı sağlayacak bir belge niteliği kazandırmaktadır.

Metodoloji sürekli olarak gelişen yapıdadır. Projelerde kazanılan deneyimlerle proje süreçleri bilgi geri beslemesi yolu ile geliştirilmelidir. Metodolojide canlı bir varlık gibi organizasyonla beraber büyümeli, gelişmelidir. Süreçlerin en iyilenmesi sayesinde metod ve yaklaşımlarda gelişecektir.

5.4. Proje Tanımı Nedir ?

Projeler hepsi kendi çerçevesinde bağımsız niteliklerde ve kendine özel yapıdadır. Bu nedenle projelerin tanımının yazılı olarak mutlaka yapılması gereklidir.

Proje Tanımlaması ve işlerin net olarak analizi proje başarısının anahtar olmazsa olmaz parçasıdır.

Projelerin başlarında projeler net değilse hemen kolayca projeleri tanımlamak zordur. Bu proje tanım dökümanı işin niteliğine göre birkaç sayfadan kitaplar dolusu belgelerden olabilecek şekildedir. Bu aşama proje başarısı için yaşamsal hassasiyet taşır. Tanımlama aşaması ilk aşamadır ama proje planlaması ile de iç içe özellik taşır. Tanımlama çalışmalarında plana aktarılan bilgiler sayesinde proje süresi ve tahmini maliyeti konusunda varsayımlar yapmak olanağımız olur.

5.5. Proje Tanımlamada Kullanılabilecek Bileşenler

İlk işimiz Proje Tanım belgesini hazırlamaktır. Bu proje ile ilgili bize genel bilgiler verecektir. Proje Tanım belgesi hazırlanması ile proje organizasyonumuzu geliştirmeye başlarız.

Tanım aşamasında proje ekibi

- Proje Hedeflerini
- Proje Başarı Faktörlerini
- Proje Organizasyonunun
- Projenin Yönetsel ve Stratejik planları ile uyumlu olmasını

tanımlamalıdır.

Ayrıca Proje organizasyonu yapısında

- Sponsor
- Proje Yöneticisi
- Proje Ekibi
- Proje Liderleri Takımı
- Proje Danışmanları
- Kullanıcı ve müşteriler

tanımlamaları vardır.

Proje Yöneticisi bilgileri derler , toplar , gözden geçirir , formal metodlarla düzenler,

Standartları tanımlar , kaynak gereksinimlerini belirler, teknoloji uygulaması ve yayılması ile ilgili metodları oluşturur.

5.6. Proje Tanımı Kimlerle ve Nasıl Geliştirilmeli

Proje ekibinin kullanacağı bilgileri geliştirmek ve elde etmek için

- Beyin fırtınası
- Ürünle ilgili kişilerin demoları ve sunumları
- İlgili kullanıcılarla çeşitli seviyelerde toplantılar
- İdari toplantılar
- Teknoloji ve araştırma metodları
- Görüşme sonuçları ve izlenimleri raporlamak

yaklaşımları kullanılır.

Bu çalışmalarda KİMLER sorusu da görev ve fonksiyonları ile aşağıdaki gibidir.

- Sponsor : Yani projenin sahibi , hedefleri , amacı ortaya koyup onaylayan kuruluş yada kişilerdir.
- Üst Yönetim ve Yönetim Takımı : Projeyi irdeleyen karar mekanizması ile projeyi izleyen , onaylayan ekip.
- Proje Yöneticisi : Projenin sorumlusu olan yönetici/ler
- Proje Ekibi : Proje faaliyetlerini yerine getirmek için tüm bağımsız ve/veya bağlantılı aktiviteleri yapan proje çalışma grubunun tamamıdır.
- Danışmanlar : İstendiğinde veya gereksinim duyunca yardım alınan gözetmenlik , yol gösterme ve öneri getirme mekanizmasında çalışanlardır.

5.7. Proje Aktivitelerinin Geliştirilmesi

Proje Planlama sürecinin en önemli bölümü aktivitelerin tanımlanmasıdır.

Bu aktivite listesinden projede yer alan İşlerin kolayca yönetilebilmesi daha küçük parçalara ayrıştırılması yapılır. Ayrıştırılmış bu İş listesinde projede yer alan tüm aktiviteler yer alır. Projenin bu modüler parçalara ayrılması sayesinde projeye ait zaman ve maliyet tahminlemesi çok daha net olarak yapılabilecektir.

PY aktivitelerin tanımlanmasınından da sorumludur. Sponsor ve PY birlikte projenin ana tanımlarını yaparlar. Projede teknik konular varsa bu kısımına da Bilgi Teknolojileri Yöneticisi katılımıyla tanımlamalar yapılır.

Aktivite listesi hiyerarşik yapıda yazı ile veya grafik olarak yada her iki yönetimi hibrid olarak kullanarak yapılır. Aşağıdaki gibi örneklersek:

1.0. PROJE YÖNETİMİ

1.1. Proje Planlaması

1.1.1. Proje Planının Geliştirilmesi

1.1.2. Proje Planının Entegrasyonu

1.1.3. Proje Planının Revizyonu .....

1.2. .Projenin İzlenmesi

1.2.1. Projenin Raporlanması

1.2.2. Projenin Gerçek Zamanlı Verilerinin Toplanması

1.2.3. Proje Verilerinin İstatistiksel Modelle Ölçümlenmesi

1.3. Kalite Güvence Aktivitelerinin Gerçekleştirilmesi

1.3.1. Kalite Yönetim Planınn Hazırlanması

1.3.2. Denetim Planının Hazırlanması

1.4. Konfigürasyon Yönetiminin Gerçekleştirilmesi

1.4.1. KY Planın Hazırlanması

1.4.2. Yazılım ve Donanım Envanteri

1.4.3. Değişiklik Komitelerinin Tanımlanması

1.4.4. Tüm Envanterin Bakım Tanımlanması

2.0. TASARIM

2.1. Tasarım Kavramı

Şekil 5.2 Yazılı Gösterim

5.8. Proje Aktivitelerinin Tanımlanması

Temel olarak yanıt aradığımız soru şu olacaktır. ‘Projemizin başarı ile hedefine ulaşması için yapılması gerekenler nelerdir ?’

Bu kapsamda projeyi modüler parçalara bölmek için kontrol edilebilir birimlerden oluşan alt ana parçalara işler ayrıştırılır.

- Ana Aşamalar belirlenir
- Projede rol alacak olan birimler belirlenir

Oluşan bu modüllerin her birisini de bir ‘iş paketi’ olarak ele alırız. İş paketleri bağımsız olarak kendi içlerinde ele alınabilirler. Kendi içinde kontrollü olarak yönetilebilir, planlanmış, bütçelenmiş ve fonksiyonel tanımı programlanmış ünitelerdir.

Anahtar husus bu iş ayrıştırmasında ünitenin kontrol edilebilecek düzeyde olmasıdır.

6. KONFİGÜRASYON YÖNETİMİ

6.1. Genel Konfigürasyon Yönetimi Kavramları

Projenin temel anlamı ne diye kendi kendimize soralım ? Nedir ? Bunun birinci yanıtı 'PLANLAMAK' olacaktır. Yani elimizdeki varolan imkan ve kabiliyetlerle hedeflediğimiz bir işi yapabilmek için yapmış olduğumuz organize işlerin planlanmış şekline biz PROJE demekteyiz. Peki planlamak ve projelendirmek demek işin başından bitimine kadar o işle ilgili rutin faaliyetleri yapmak mı demektir? Biraz düşünürsek bunun çok zor bir soru olduğunu farkederiz. Evet planlarız , projelendiririz işleri fakat bunun anlamı iş başlayınca yeni bir istek , ilave , eksiltme , düzenleme , zorunlu müdahale , kaynak ayarlamaları vb yani DEĞİŞİKLİKler gelmeyecek demek değildir. Tam tersine dinamik bir yaşam döngüsüne sahip olan sistemler için işler , projeler yaptığımız için proje başlar başlamaz değişiklikler ve değişiklik istekleri projemizin yaşamı içine girmeye başlayacaktır. Başka bir deyişle projelerimizde aslında değişmeyen tek değişken, değişiklik işteklerinin gelmesidir. İşte bu değişiklik istekleri ve değişikleri bir disiplinle yönetmek için yapmış olduğumuz prosedürel yönetim planlarımıza Konfigürasyon Yönetimi demekteyiz. Özellikle yazılım yaşam döngüsü içinde değişiklikler yaşamın bir parçası olduğu için Yazılım Konfigürasyon Yönetimi ve öğeleri bizim için yazılım bakım , uyum , güncelleme, etkinlik ve maliyetlerini yönetmek için bir çözüm metodolojisi olarak önümüzdedir.

6.1.1. Proje Yönetim Destek Fonksiyonları

Projelerin plan aşamasında , proje takımlarını kurma, planlama, liderlik etme, izleme ve yönetme amacıyla teknik mekanizmaları oluşturup işlemesini sağlamak gereklidir. Proje Yöneticileri tanım olarak projenin yönetsel olan tüm aktivitelerinden sorumludur. PY lerinin idari ve teknik olarak pratikte iki temel ayrımda yönetim sorumlulukları vardır. Bu nedenle de takımlarının iç organizasyonunu oluştururken buna dikkat etmelidirler.

6.1.2. Konfigürasyon Yönetimi Organizasyonu

Proje Yönetimi destek fonksiyonu olarak idari görevleri :

- Projeyi programlama
- Pazarlama
- Sözleşme yönetimi
- Proje sınırlarında gereken ürün , hizmet alımı
- Döküman üretimi
- Bütçeleme ve bütçeyi yönetme

Teknik destek olarak ise

- Değişiklik İsteklerini kontrollü bir işleyişle , proje kalite güvence standartlarına uygun yürütmek
- Tüm aktivite zincirlerinin kalite güvencesini sağlamak
- Proje ürün ve hizmetlerinin izlenmesini sağlamak
- Test
- Konfigürasyon Yönetimi

- Ürün ve hizmet iş isteklerine uygun olarak geliştirilmesini temin etmek

Olarak organize olmalıdır.

Bu idari destek ve teknik destek fonksiyonları birbirleri ile organize olan ilişkili aktivitelerdir.

Bu yapı altında var olan Konfigürasyon Yönetimi proseslerini özellikle yazılım geliştirme projelerimizin yaşam döngüsü içinde sağlıklı bir ömürle yaşamını devam ettirsin istiyorsak tanım , teknik , yürütme ve uygulama açısından sağlıklı bilimsel bir tabana oturtmak zorunluluğumuz vardır. Özellikle Yazılım Mühendisliği disiplini açısından baktığımızda geliştirilmekte olan yazılım faaliyetlerinde % 60 ın üzerinde bir süreyi uygulamalara bakım yapmakla geçirdiğimizi düşünürsek bu Konfigürasyon Yönetimi kavramlarının tanımlanması , yönetilmesi unsurunun değerini çok daha iyi anlayabiliriz. Yazılım projeleri de diğer alışılagelen projeler gibi belirli bir kaynakla ve mali bütçe ile hayata geçmektedir. Burada da diğer klasik projeler gibi (yani inşaat projeleri , otomobil projeleri , askeri projeler, uçuş sistemleri , vb..) üretim ve sonradan yürütüm için geçen zamanın bir maddi değeri vardır. İşte 100 birim zamanlık ömre sahip bir projede bunun 70 birim zamanın o projeye ait gelen yeni değişiklik isteklerini , uyarlamaları , hataları düzeltmeyi içerdiğini düşünürsek işin mali boyutunu çok daha iyi kavrayabiliriz. Bunun da direk olarak yaşayan bir yazılım projesinde mali değer olduğunu görebiliriz. Projede

kaynak olarak ele alabileceğimiz makine , ortam ve insan yeteneklerinin de projelere ait teknik ve idari konfigürasyon planlarına ulaşması ve bunları hızla , etkin şekilde uygulanması gereğinin çok açık bir durum olduğunda da fikren birleşiriz. Projelerde başarı içinde önemli bir kritik başarı faktörü olarak bunu kabul ederiz.

Konfigürasyon Yönetimini de bir organizasyon olarak ele alıp tanımlı şekle getirmeliyiz. Konfigürasyon Yönetiminin sorumlulukları ise :

- Konfigürasyon Yönetiminden kim sorumlu olduğunun , yetki düzeyinin tanımlanması
- Standartların tanımlanması, yerleştirilmesi, proje ekiplerinin referans kılavuz olarak kabul edeceği standartlar el kitaplarının oluşturulması ve bunların güncel olarak yaşatılması
- Konfigürasyon Yönetiminde kullanılacak olan araçları, kaynakları ve işlem proseslerinin tanımlanması

Şeklinde özetlenebilir.

6.2. Konfigürasyon Yönetim Planı (KYP)

Konfigürasyon Yönetim Planı , Proje Yönetim planına bağlı bir başlıkta veya bazı uygulamalarda olduğu gibi Kalite Güvence Planı içinde bir alt-başlık olarak tanımlanabilir. Her iki yaklaşımda kavramsal , yönetsel ve uygulama açısında planın önem yada durumunda bizce bir sakıncalı durum yaratmaz.

Biz bu konuda bağımsız yapıda Proje Planına bağlı diğer planlar gibi olması yaklaşımını konun öneminin gitgide daha çok öne çıkması nedeni ile daha uygun bulmaktayız. Zaten bütün planlar birbirleri ile entegre olarak ortak bir ürünü geliştirme amacına hizmet ettiği için bu şekilsel bir gösterim olarak tarafımızdan ele alınmaktadır. Bu planın şekli yerini tartışmak yerine kavramsal olarak önemi bizler için çok daha kıymetli bir amaç obje olarak hedefimizdir. Örnek bir Konfigürasyon Planında temel başlıklar neler olmalı diye bakarsak aşağıdaki ana konu başlıklarını görmekteyiz.

ÖRNEK : **Konfigürasyon Yönetim Planı :**

1. Organizasyon ve Kaynaklar

1.1 Organizasyonal yapı

1.2 Kişisel yetenek seviyesi

a.İnsan Kaynakları yetenek ve eğitim düzeyleri

b.Gerekli işler ve ortamlar

c.Kullanılan ekipman ve araçlar

2. Standartlar El Kitapları, Prosedürler, Politikalar, Kılavuzlar

2.1 Bilgi akış diyagramı

2.2 Otomasyon parametreleri

3. Konfigürasyon Tanımlama

3.1 Kontrol kalemi tanımlama metodu

3.2 Konfigürasyon kontrol metodu

a.Kontrol kalemleri listesi

4. Metodların tanımlanması

4.1.Doküman isimleme ve arşivleme, yazılım bileşenleri, revizyonlar ve sürümler.

5. Kontrol Kalemlerinin Onayı yada Reddi

6. Versiyon Kontrolü

6.1 Yazılım ve dokümantasyon versiyonlarının hazırlanması

6.2 Onay ve Kabul prosedürünün çıkarılması

7. Proje Dokümantasyonun Saklanması ve Teslimi

7.1 Saklama gereklilikleri (elektronik ve kağıt olarak)

8. 3.Parti Konfigürasyon Yönetimleriyle İlişkiler

9. Diğer Bilgi

6.3. Yazılım Konfigürasyon Yönetimi

Yazılım geliştirilmesi söz konusu olunca 'DEĞİŞİKLİK' kaçınılmazdır. Doğal olarak değişiklik söz konusu olunca yazılım projelerinde çalışan yazılım mühendisleri arasında bunların yönetimi problem ve sorunlarına neden olurlar.

Değişiklik isteklerinin yaratmış olduğu bu çelişkileri, değişiklikleri uygulamadan, planlamadan önce yöneterek yazılım ürünlerimizin hatalarının azalmasını ve kalite güvence normlarının yükselmesini temin ederiz. İşte yazılımlarda bilimsel olduğu kadar sanatsal yeteneklerimizle bu çelişkileri ve değişikleri projelerde yönetebilmek de Konfigürasyon Yönetimi olarak adlandırılır.

Amacımızda hatalarımızı ve sorunlarımızı en aza indirerek üretkenliğimizi maksimize etmektir. Bu aslında insanın beşeri yaşamında en çok iktisadi teori olarak hep gündeminde yer tutmuş olan üretkenlik, karlı üretim yani kar maksimizasyonu sorunu ile eşdeğerdir.

Yazılım proseslerinde Değişiklik Yönetimi tüm prosesleri kapsayan onların tamamı üstünde yer alan bir üst-aktivitedir. Çünkü değişiklik yazılım geliştirme ve yazılım yaşam döngüsü içinde her an olabilecek bir aktivitedir.

Yazılım Konfigürasyon Yönetimi aktiviteleri

- Değişiklikleri anlar ve tanımlar
- Değişiklileri denetler
- Yapılan değişikliklerin isteklere uygun olarak yapıldığını garanti eder
- Değişikliklerin çeşitli şekillerde raporlanmasını

temin eder.

Yazılım Konfigürasyon Yönetimi(YKY) ile yazılım destek faaliyetlerinin çok net bir şekilde birbirinden ayrılması gerekir.

Yazılım desteği, geliştirilen yazılımın müşteri ve/veya kullanıcıya tesliminden sonra operasyonel olarak çıkan işlemler ile ilgili yapılan yazılım mühendisliği çalışmalarıdır.

Yazılım Konfigürasyon Yönetimi ise yazılım mühendisliği projeleri başladıktan bitene kadar yani yazılımın tüm yaşamı boyunca yapılan izleme ve denetleme proseslerinin tamamıdır.

Yazılım Mühendisliğinin ana amacı değişikleri kolaylıkla , yönetilebilir şekilde isteğe uygun gerçekleştirip böylece harcanan zaman ve kaynakları en aza indirgemektir.

Yazılım prosesinin çıktıları olan bilgiler 3 ana kategoride ele alınabilir. (1) Yazılım kodları (kaynak kod seviyesinden obje koda kadar) (2) yazılımların çalışması ile elde edilen sonuç ürünler ve (3) veriler. Bu üç bileşeni de içeren yazılım prosesleri ortak olarak yazılımların konfigürasyonudur.

Herbir konfigürasyon öğesi diğer bir öğe ile ilişkilidir. Bir tanesinde en ufak bir sıkıntı olması sonucu etkiler. Ayrıca bu prosese bir başka değişkende heran dahil olur buda değişikliklerdir. Değişiklikler herhangi bir zamanda ve sebeple vuku bulabilir. Bu aynı zamanda yazılım mühendisliğinde yazılım yaşam döngüsünün doğasından gelen bir yaşamın kalıtımsal özelliğidir. Yani değişiklik ve değişim olması için bir sebep yoktur. Yaşam döngüsünün her anında karşımıza bir sebeple değişiklik çıkacaktır.

Değişikliğin çıkış kaynağı nedir dersek ? Bu sorunun yanıtı değişikliğin yapısına göre farklılık gösterir. Bununla beraber değişikliğin temel kaynağında dört sebep görebiliriz:

- Değişen Pazar ve iş koşullarına bağlı olarak iş isteklerinin değişim gereğini dikte etmesi
- Yeni müşteri istekleri sonucunda verileri üreten bilgi sistemlerinin sistem modifikasyon , uyarlama gereksinimleri.
- Yazılım mühendisliği ekiplerinde veya proje önceliklerinde yeniden işleri organize etmek ve/veya büyüme , küçülme gereksinimleri
- Yeniden tanımlanması istenen sistem ve ürünlere bağlı planlama , bütçeleme yapma gerekleridir.

Yazılım Konfigürasyon Yönetimi, yazılım yaşam döngüsü boyunca geliştirilen değişiklikleri düzenleyen aktivitilerin tamamıdır. Bunu yazılım süreçlerine uygulanan bir yazılım kalite güvencesi işlemi olarak adlandırabiliriz.

6.3.1. Bir Yazılım Konfigürasyon Yönetimi Senaryosu

Tipik bir Konfigürasyon Yönetimi senaryosunda proje yöneticisi yazılım gruplarından sorumludur. Konfigürasyon yöneticisi ise Konfigürasyon Yönetimi prosedürlerinden, politikalarından sorumludur. Yazılım mühendisi iç ve dış müşterilerin kullandığı yazılımları geliştirmekten sorumludur. Senaryomuza bakarsak , orta boy bir yazılım projesinde 5 kişilik bir yazılım geliştirme ekibi 18.000 satırlık yazılım kodu geliştirmiş olduğunu varsayalım.

Operasyonel düzeyde, senaryoda birçok rol ve görevler bulunmaktadır. Proje yöneticisi için amaç yazılımın verilen süre içinde

tamamlanmasıdır. Bununla beraber yöneticiler yazılımların geliştirilme sürecini izlerler, sorun ve problemleri tespit edip, çözümler bulurlar. Bu işlemleri yazılım sistemlerinin ürettiği raporları inceleyip, analiz ederek takip ederler.

Konfigürasyon Yöneticisinin amacı projedeki kod geliştirme, test etme, değişikleri yerine getirme, envaneteri yönetme çalışmalarına ait prosedür ve politikaların uygulanmasını temin etmektedir. Kod üzerinde yapılan değişiklerin yapılması için uygulama tekniklerini geliştirir. Değişiklik Denetim Kurulu'nda değerlendirilerek gelen formal değişiklik isteklerini kontrollü olarak yetkili olarak uygulayan mekanizmayı kurar, uygular. Proje planını tüm çalışma grubunu kapsayacak şekilde iş, görevler ve sorumluluklar bazında hazırlar, yayımlar .

6.3.2. Konfigürasyon Yönetim Sistemlerinin Bileşenleri

Yazılım Konfigürasyon Yönetimi 4 önemli bileşeni ile geliştirilen yazılım sisteminde ele alınması önerilebilir.

- Bileşenlerin elemanları

  Veri tabanı yönetim sistemleri içinde tüm konfigürasyon yönetim bileşenlerini yönetebilmek ve erişebilmek için araçlar , gereçler ve olanakların tamamıdır.

- Proses elemanları

  Değişiklik yönetiminde kullanılan yazılımlar, yazılım mühendisliği öğelerini en etkin biçimde kullanacak tüm prosedür

ve yöntemlerin tanımlanması, uygulanması , tasarlanması ve takip edilmesini içeren tanımlar bütünüdür.

- Üretim elemanları

  Yazılımın doğru konfigürasyon altında çalışmasını garanti altına almaya yarayan geçerli bileşenleri (versiyon denetimi , değişiklik izleri takibi vb.) bir disiplin içinde çalıştıracak otomatik araçları oluşturmaktır.

- İnsan Kaynakları

  Etkin Yazılım Konfigürasyon Yönetim uygulamasını tanımlı prosesler ve araçlar ile yapacak yazılım ekibinin tamamıdır.

Bütün bu bileşenler tek başına varlıkları ile birşey ifade etmezler. Fakat işleyiş bütünlüğünde hepsinin birarada bir bütün içinde olması , çalışması Yazılım Konfigürasyon Yönetimi açısından bir yapısal yaklaşım değeridir. 4 Bileşende fonksiyonel olarak bir yapısal yaklaşımın entegre olarak çalışması ile oluşur.

6.3.3. Ana Hatlar / Tanımları 'Baseline'

Değişiklik yazılım geliştirmede hayatın bir gerçeğidir. Müşteriler isteklerinde her zaman uyarlama , değişiklikler yapmak isterler. Yazılım geliştirenlerde teknik yaklaşımlarında benzer geliştirme, uyarlama ve değişiklikler yapmak isterler. Yöneticiler de proje yaklaşım ve stratejilerini geliştirmek isterler. Bütün bu uyarlamalar ve nedenleri nelerdir ? Yanıt ise çok basittir. Zaman geçtikçe tüm bu gruplar sistemlerle ilgili çok daha fazla bilgi sahibi olurlar. (yani neye

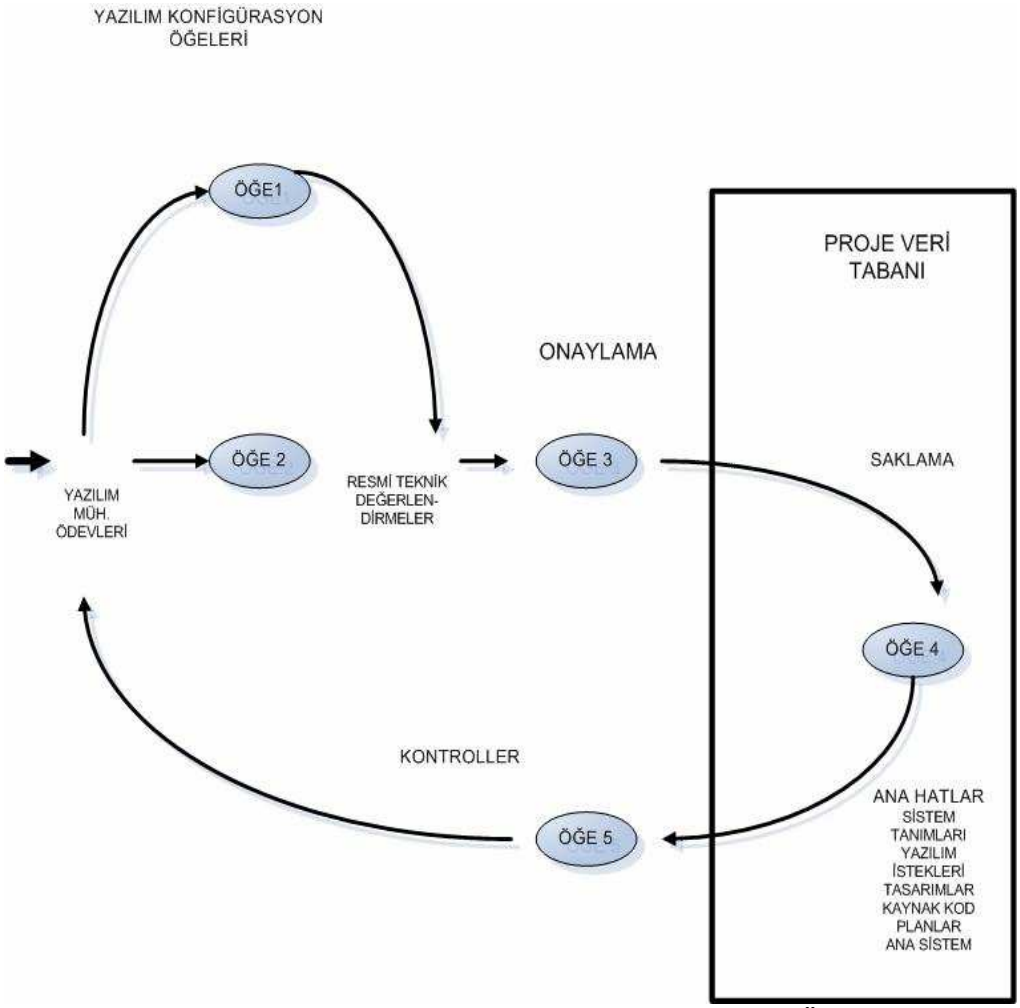
gereksinimleri olduklarını , en doğru yaklaşımın ne olduğu , en iyi nasıl bu işi yapabilecekleri vb.) Bu ek bilgilerle değişiklik isteklerini daha iyi geliştirebilirler. Bununla beraber bunların yazılım uygulamacıları tarafından da daha iyi karşılanması, kabul edilmesi doğal olarak zorlaşmaya başlar. Ayrıca birçok değişiklik daha geçerli bir isteğe, soruna dayanmaktadır.

Ana tanımların yazılım konfigürasyon yönetimi kavramında olması ile değişikliklerin kontrollü olarak yapılmasına ve böylece zarar verecek düzeyde ciddi sorunları oluşmasına engel olan bir yapısı vardır. Burada Uluslararası Elektronik Mühendisleri Enstitüsü otoritesi Ana tanım denince : ancak resmi bir değişiklik kontrol prosesinden geçerek gelen onaylanmış, üzerinde herkes tarafından hemfikir olunmuş geliştirilecek ürün ve hizmetlere ait tanımlanan özelliklerin hepsini anlamaktadır. Bizlerde üzerinde tüm süreçlerde fikir birliği sağlanmış olan geliştirilmesi başlanan ürün, hizmetlere ait özelliklerin, sınırların, yapıların anlatıldığı onaylanmış resmi belgeler olarak Ana Hatları algılamaktayız. Bir çeşitl yazılım sistemlerimizin sınırlarını belirleyen kırmızı yaşamsal tanım çizgileri olarak bu ana tanım kavramını düşünebiliriz.

Bir Konfigürasyon öğesi ana hat olmadan önce , normal bir değişiklik prosedürü olmaksızın, hızlıca yapılabilir. Bununla beraber buna ait bir ana hat tanımlanırsa bunun değişimi resmi bir onaylı proses zinciri ile ele alınır. Değişiklik yine yapılır fakat buna ait yönetim planında

tanımlanan prosesler , yollar , mekanizmalar devreye alınarak resmi olarak işlemler yapılır.

Yazılım Mühendisliği disiplininde ana hatlar yazılım geliştirmenin kilometre taşlarıdır. Teknik olarak yazılım geliştirme adımların bir yada birkaç konfigürasyon öğesinin tamamlanması ile bir ana hat 'baseline' belirlenir.Bu tasarım elemanlarının ele alınması ile belgelemeler yapılır, değerlendirilir. Hatalar ve sorunlar tespit edilir, düzeltilir. Bir tasarım modelinin tümü ele alınıp, değerlendirilip , hatalardan arındırıldıktan sonra onaylanınca tasarım modeli bir ana hat olur. Program tasarımındaki daha sonraki değişikliklerin tamamı bu noktadan sonra onaylanarak ve değerlendirilerek yapılır. Tüm ana hatları da detaylı olarak bir grafiksel olarak aşağıdaki gibi gösterebiliriz.(Şekil 6.1) Bu gösterimde yapılan yazılım analiz çalışmaların ilerleyişi ile ana hatların tanımları oluşur. Yazılım mühendisliği işleri bir veya birden fazla yazılım konfigürasyon öğesinden oluşur. Bu öğeler ele alınıp , tek tek irdelenir, onaylananları proje veri tabanına kayıt edilir. Eğer yazılımcı yazılım projesinde bulunan bir yazılım konfigürasyon öğesinde çalışmak isterse bunu proje veritabanından kendi birim çalışmalarını yaptığı özel çalışma platformuna taşır, kopyalar. Daha sonra Yazılım Konfigürasyon Yönetimi tarafından kontrol edilen bu yapıda uyarlamalarını yapar. Şekildeki okların yönü öğelerle yapılan çalışmaların izlemesi gereken yolu tanımlar.



Şekil 6.1 : Proje Veri Tabanındaki Ana Yazılım Konfigürasyon Öğeleri

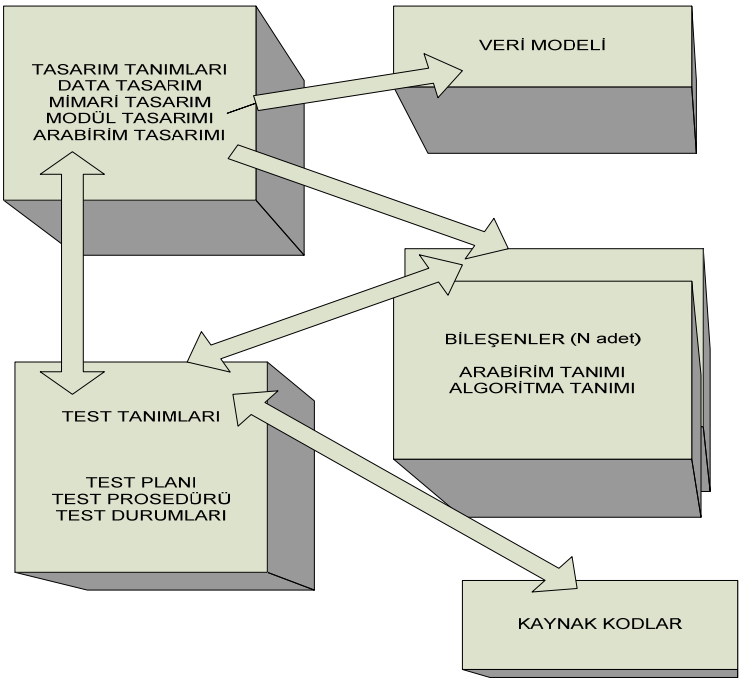
6.3.4. Yazılım Konfigürasyon Öğeleri

Yazılım Konfigürasyon öğeleri yazılım mühendisliği iş adımlarının bir parçası olarak ortaya çıkarlar. Çok ekstrem bir durum olarak bazen çok büyük ve özel bir durumda tek başına bir bölüm yada çok kapsamlı bir test işleminin bir parçası da olabilir. Yazılım Konfigürasyon öğesi bir belge, tüm testlerin bir parçası veya bir C++ fonksiyonun kendisi veya Java appletle yazılmış bir kod olabilmektedir.

Öğeler, yazılım çalışmalarından bir türev ürün olarak elde edilmektedirler. Yazılım mühendisliği organizasyonlarında konfügürasyon kontrolü altında bir yazılım aracı olabilmektedir. Bu yazılım konfigürasyon öğesi bazı editorlerde, derleyicilerde, tarayıcılarda ve yazılım otomasyon araçlarında yazılım konfigürasyonun hazır bir parçası şeklindedir. Çünkü bu ortamlarda yazılım kodu üretilirken yazılım konfigürasyon kontrol prosesi otomatik olarak devreye girerek çalışmaktadır. Her ne kadar versiyon sorunları nadiren de olsa , yeni versiyon ürünlerde orjinal versiyondan başka kod oluşturma gibi sorunların olması da olasıdır. Bu nedenle yazılım geliştirme araçları yazlım konfigürasyon yönetimi prosesine uygun olarak ana hatlara dikkat eden ürünler üretmeye yardımcı olacak şekilde tasarlanmaktadır.

Tekil bir isimle proje veri tabanında konfigürasyon yönetim öğeleri kataloglanmalı ve organize edilmelidirler. Bir şekille konfigürasyon nesnelerini açıklayabiliriz. (Şekil 6.2)

Konfigürasyon nesneleri, başka nesnelerle isim, özellik, tanım gibi belirleyicilerle ilişkilendirmelidir. Bu nesneler ise : Tasarım özellikleri, Veri modelleri , Bileşenler, Kaynak Kodlar ve Test tanımları olarak ayrı ayrı tanımlanırlar. Bununla beraber bu nesneler birbiri ile oklar aracılığı ile gösterimsel olarak ilişkilendirirler. Dairesel oklarla birleştirme ilişkisini tanımlar.



Şekil 6.2 Konfigürasyon Nesneleri

6.4. Yazılım Konfigürasyon Yönetimi Veri Tabanı

Yazılım mühendisliğinin ilk zamanlarında yazılım yada kodlama işlemleri delikli kart ortamına yapılmaktaydı, konfigürasyon öğesi olarak da bu yazılı ortamlarda bakım , değişiklikler yapılmaktaydı. Bu işlemin birçok nedenden dolayı ciddi sorunları vardı. Bu ortamın kağıt bazlı olması nedeni ile aramak, bulmak, saklamak , sıralamak, değiştirmek oldukça zor zaman alıcı ve hatalara çok açık bir

yapıdaydı. Detaylı ve karmaşık konfigürasyon öğeleri ile ilişkili çalışmalar yapmakta imkansız derecede zordu.

'Repository' diye genel olarak adlandırabileceğimiz bu konfigürasyon öğelerini saklama platformlarımız/veri tabanımız yazılım mühendisliğinin ilk zamanlarında yazılımı yapan, geliştiren insanlardı yani programcılardı. Fakat artık şimdi bunları manyetik platformlardan oluşan veri tabanlarında veya saklama ortamlarında bulunduruyoruz. Buda işin uygun olarak yürütülmesine yetmediği için öğelerin entegre olarak çalışmasını sağlayacak birçok entegrasyon aracından faydalanmaktayız.

6.4.1. Değişiklik Veri Tabanı Rolü

Yazılım Konfigürasyon Yönetimi veri tabanı yazılım ekibine değişiklikleri etkin olarak yönetebilmesi için yardım eden bir veri yapısı mekanizmasıdır. Aslında bir veri tabanı yönetim sistemi fonksiyonları sağlar. Ek olarak da aşağıdaki fonksiyonları temin ederler.

- Veri Bütünlüğü

  İlgili nesneler arasında iletişimi ,geçerliliği , tutarlılığı, uyumluluğu ve otomatik olarak birbirleri ile bağlantılı olmayı içermektedir. Böylece veriler ve veri elemanları arasında tekil olmayı da temin etmektedir.

- Bilgi Paylaşımı

  Bir yazılım geliştirme ekibi içinde bilgilerin , ortamların , ortamlara çoklu erişimin , yapılacak değişimlerin eşanlı olarak

birçok kullanıcı ile kontrol edilerek paylaşımını, güncellenmesini ve kontrollü kullanımını temin eder.

- Araçların entegrasyonu

  Konfigürasyon Yönetim fonksiyonlarının veri modeli yardımı ile yazılımcılar tarafından kullanılan tüm yazılım geliştirme araçları ile tam uyumlu yerine getirilmesine yardım eder.

- Veri entegrasyonu

  Birden çok konfigürasyon öğesinin bir yazılım konfigürasyon yönetim görevi ile uyumlu veri seti ile tutarlı çalışmasını sağlar.

- Metodoloji zorlaması

  Tanımlanan ve veri tabanından saklanan kurallar zinciri ile yazılım geliştirmesi yapan yazılımcılara bir disipliner çalışma özelliği sağlayan belgeli mühendislik yaklaşımı tanımlar. Bunun çerçevesinde işlemlerin , proseslerin yürümesini tutarlılıkla temin eder.

- Belgeleme standardizasyonu

  Yazılım mühendisliği kapsamındaki tüm faaliyetlerin tanımlanmış bir belgeleme standardı ile kayıt altına alınmasını temin eder. Yapılan kodlama ve diğer yazılım çalışmaları içinde kayıtlı , belgeli çalışma yapısı önerir.

Bu yapıyı bize sağlayan veri tabanı modelleri için biz ‘meta-model’ terimini kullanırız. Meta-model teorik olarak bilginin nasıl veri tabanında saklandığını , nasıl erişileceğini ve yazılım mühendisleri tarafından nasıl

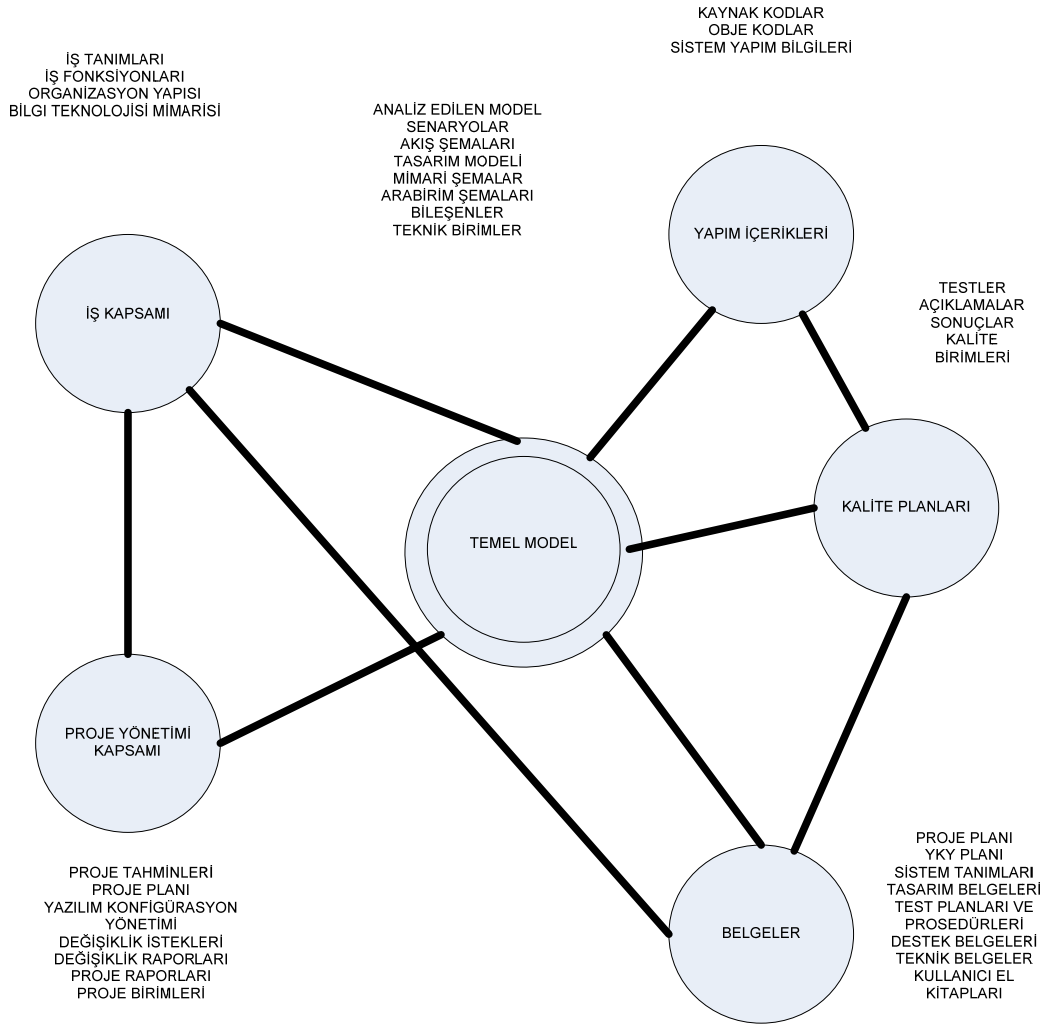
görüntüleneceğini tanımlar. Bununla beraber doğal olarak erişimin , verilerin bütünlüğünü ve güvenliğininde ele alınmasını içermektedir. Özellikle günümüzde veri bütünlüğü ve güvenliği tam otomasyonun olduğu açık sistemlerimiz için çok kritik kendi başına çok detaylı ele alınacak kadar önemli konulardır. Çok özel güvenlik ve bütünlük çalışmaları geniş araştırıcı kitleleri tarafından üzerinde inceleme yapılan çalışma alanlarından birisidir.

6.4.2. Tanımlar ve İçerik

İki temel perspektifle konuyu veri tabanının içerik ve özellikleri açısından ele alabiliriz. Veri tabanında Neler vardır ve veri tabanı tarafından hangi servisler verilmektedir .Bunu detaylı bir şekillemek faydalı olur. (Şekil 6.3)

İyi tasarlanmış bir veri tabanı aynı tip servisleri karmaşık bir veri yönetimi ile tanımlar ve bu tanıma özel yazılım mühendisliği servis tanımlarını yapar.

Ayrıca yazılım geliştirme ekibine entegre direk yönetim fonksiyonları, ürünlere özel tanımlanmış özel kurallar, diğer yazılım mühendisliği araçları ile entegre ortamla ve her türlü gelişmiş karmaşık veriyi (text, grafik , video ve ses) aynı anda saklamayı temin etmelidir.



Şekil 6.3 Veri Tabanının İçeriği

6.4.3. Yazılım Konfigürasyon Yönetim Özellikleri

- Versiyon Yönetimi

  Sorun gidermek amacı ile yazılımcının kendi özel çalışma alanına almış olduğu kodların üzerinde yapmış olduğu çalışmaların otomatik olarak yapılan değişimlere göre veri tabanında tarihsel

olarak saklanmasını sağlar. Daha sonraki adımlarda eğer yapılan değişiklikler geri alınmak istenirse ve/veya izlenmek istenirse bunları yazılımcıya , proje yöneticisine sunar. Burada sadece üzerinde çalışılan kod yerine daha olgun versiyon izleme araçları ile ekranlar, veri elemanları , veri tanımları , veri özellikleri , veri tabanı tanımları , veriler, tüm bağlantı belgeleri ve bunların otomatik bağlantılarının izlenmesi sağlanır.

- Değişiklik Yönetimi ve izleme

  Tanımlanan ilişkili konfigürasyon nesnelerinin uğradığı değişiklik istekleri ve bunların bağlantıları izlenir. Tasarım elemanları ile nesneler arasındaki etkileşim kontrollü olarak düzenlenir. Bütünlük içinde tüm bunların uyumluluğu sağlanır ve bunlar izlenir. Yazılımcının modellemede önüne işlemek için aldığı bir nesne, ilişkili olduğu tüm öğelerle entegre olarak bir aksiyomda kullanım alanına gelmesi ve izlemesi temin edilir.

- İstekleri izleme

  Geriye daha sonra da ileriye doğru çeşitli kaynaklardan tanımlı olan iş istek yapısı ile gelen tüm isteklerin kontrollü olarak önem derecesine göre işlenmesi sağlanır.

- Konfigürasyon Yönetimi

  Üretim aşamalarının tamamının tanımlı proje kilometre taşları ve bir seri konfigürasyon kontrol proses zinciri ile izlenmesini , iyileştirilmesini, geliştirilmesini ve denetimini hedefler.

- Denetleme

  Ne zaman , neden , nasıl , kimden ve nerede değişiklik çalışmalarının yapıldığını izlemektir. Bunların veri tabanından tarihçe bilgilerinin takip edililir şekilde olmasını sağlamak olarak açıklarız.

6.5. Yazılım Konfigürasyon Yönetim Prosesi

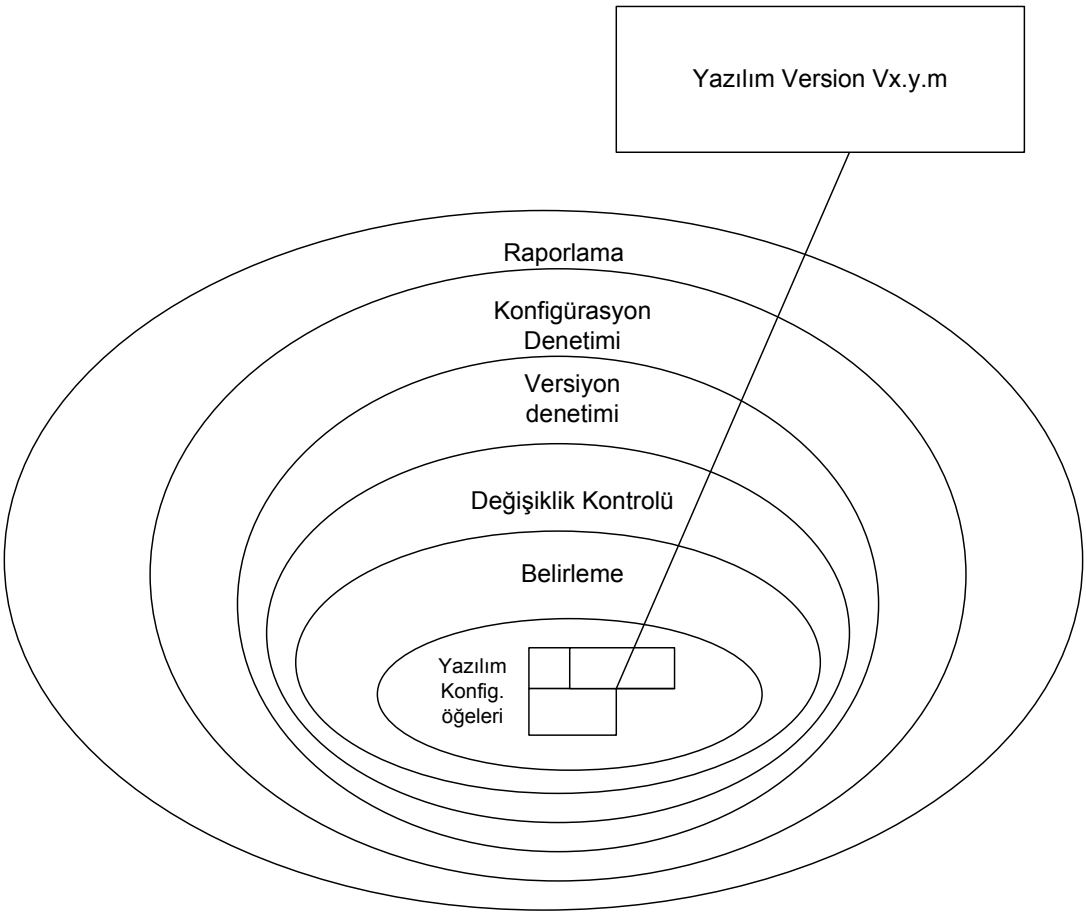
Yazılım Konfigürasyon Yönetimi prosesi dört ana amacı ödev olarak ele alır.

- Ortak olarak kullanılan tüm öğeler yazılım konfigürasyonunda tanımlanır , belirlenir.
- Bu öğelerden biri veya birçoğunun üzerinde yapılacak olan değişiklikleri yönetmeyi hedefler.
- Yazılım uygulamalarının değişik versiyonlarının oluşturulmasını kolaylaştırmayı amaçlar.
- Zamanında kalite güvencesi ile yazılımların konfigürasyonlarının bakımlarını sağlar.

Bütün bu işlemler yapılırken hantal bir şekilde ve bürokratik bir yapıda işlemlerin yapılmasını hedeflemeliyiz. Fakat özellikle yazılım geliştirme ekipleri içinde konfigürasyon yönetim ilkelerinin işlemlerini , yazılımcıların işlerini kolaylaştırması için yapısal yaklaşımla fonksiyonel olarak tasarlanması gerekir. Bunun içinde karmaşık yapıda birçok soruya , sorguya her koşulda yazılım geliştirilirken ve sonrasında yanıt verecek yapının konfigürasyon açısından kurgulanması gereklidir. Bu yapılanmada yanıt bulmamız gereken karmaşık sorularımızı şu şekilde listeleyebiliriz.

- Yazılım Konfigürasyonundaki elemanları yazılım ekipleri nasıl belirleyecekler ?
- Kurulacak olan yönetim sistemi varolan birçok versiyondaki/sürümdeki uygulamayı nasıl organize edecek ?
- Yeni yazılım versiyonları kullanıcı/müşteri kullanımına verilmeden önce nasıl konfigürasyon kontrol değişiklikleri yönetilecek ?
- Değişikliklerin onaylanmasından ve önceliklendirilmesinden kim sorumlu olacak ?
- Değişikliklerin isteğe uygun olarak gerçekleştirildiği nasıl garanti edilecek ?
- Yapılan tüm değişiklerin değerini anlamak için nasıl bir mekanizma kullanılacak ?

Bu soruları Yazılım Konfigürasyon Yönetimi görevleri açısından tanımlayabilmek için beş adet tanım kavramı (Belirleme , versiyon denetimi , değişiklik kontrolü, konfigürasyon denetleme ve raporlama / izleme) ile takip eden katmanlı bir şema ile açıklamamız faydalı olur. (Şekil 6.4)



Şekil 6.4 Yazılım Konfigürasyon Yönetim katmanları

Yazılım Konfigürasyon Yönetimi prosesleri ortak merkezlidirler. Yazılım konfigürasyon öğleri içten dışarıya doğru bir ilişki ile birbrilerine bağlıdırlar. Bu öğelerden birisi katmanlar arasında hareket edince yazılım konfigürasyon yönetimi kavramları katmanlararasında bir disiplin, uygulama prensibi olarak karşımıza çıkar. Her bir yazılım konfigürasyon öğesi yazılımın belirli bir versiyonuna belirleyici olan bir rakamsal versiyon numarası ile atanır. Böylece birçok öğe ile farklı yazılım versiyonlarını aynı yazılıma ait birer

farklı versiyon kümesi olarak görebiliriz. Her bir katmanı kendine özel olarak detaylı ele almak istiyoruz.

6.5.1. Yazılım Sistemlerindeki Nesnelerin Belirlenmesi

Yazılım konfigürasyon öğelerini yönetmek ve kontrol etmek için nesneye yönelik yaklaşım kullanılabilir. İki çeşit nesne tipi belirlemesi yapılır. Bunu temel nesneler ve küme nesneler olarak isimleyebiliriz.

Temel nesne , yazılım mühendisi tarafından analiz , tasarım , kodlama ve test sırasında oluşturulan birim bilgiden elde edilir. Örneklersek , temel nesne isteklerini tanımının bir kısmı , tasarım modelinin parçası, bir bileşenin kaynak kodu veya bir test çalışmasının bir kısmında kullanılan bir koddan oluşabilir. Küme nesne ise bu temel nesnelerin toplanması veya diğer küme nesnelerin biraraya gelmesi ile oluşur. Tasarım özellikleri bir küme nesnedir. Kavramsal olarak diğer tekil nesnelerin biraraya gelmesi veya bu toplanan kümelerin toplamı ile oluşur. Bir hiyerarşik ilişkiden daha çok çapraz olarak ilişkilendirilen bir yapı ile oluştuklarını düşünebiliriz.

Tasarım aşamasında belirlenirken sıkça üzerinde tartışılarak belirlenirler. Oluşan değişiklik isteklerine bağlı olarak kümeleşme ve bağlantılı nesneler tasarım aşamalarında şekillenebilirler.

6.5.2. Versiyon Denetimi

Yazılım geliştirme sürecinde birçok aracın ve yazılımın yönetilebilmesi amacı ile üretilen ürünlere ait kodların bir disiplin altında takip edilebilmesi çalışmasıdır. Süreç içinde yaşayan bir varlık olan yazılıma

ait gelen değişiklik istekleri değerlendirme kurullarından geçtikçe, oluşan sorunlar tespit edilip düzeltildikçe yaşayan yazılıma ekler, çıkartmalar , uyarlamalar ve düzenlemeler yapılır. Bu normal bir yaşam sürecinin sonucudur. Bu yapılanların yazılımlara uygulanmasının, uygulamanın kalite kontrolünün ve denetiminin sağlıklı yapılması gerekmektedir. Bunu düzenleyen yapıya versiyon denetimi diyebiliriz. Zaman içinde yapılanların versiyonlama açısından geriye dönük olarak zaman bazlı izlenmesine ihtiyaç duyarız. Bu denetleme içinde versiyon değişim izlerinin de takibi önemli bir konudur. Büyük ve orta boy yazılım ekiplerinde uzun süreli projelerde versiyon denetimi mekanizması ve izlenmesi kalite, düzen ve sağlıklı yazılım sonuçları elde etmemizi destekler.

6.5.3. Değişiklik Yönetim ve Kontrolü

Değişim kaçınılmazdır, dolayısı ile değişimin kontrolü de zorunlu bir işlemdir. Değişim hepimizi rahatsız eden bir süreçtir. Bizler değişim söz konusu olunca endişeleniriz. Özellikle yazılımlarda yapacağımız küçük yada büyük çaplı değişimler sonuç ürünlerde istemediğimiz çapta sorunlarla karşılaşmamıza neden olabilmektedirler. Bununla birlikte yapacağımız değişiklikler sonucunda da ürünlerimizde çok ciddi faydalar da elde edebilmekteyiz. Bu değişimler sayesinde çok önemli başarılı işler de yapabilmekteyiz. Bu nedenle bir ikilem gibi

olan bu değişim ve değişimin sonucu işlemi bizler için daha büyük başarılara gidiş için de bir yol olmaktadır. Aşırı olarak yaptığımız değişiklik kontrolleri de sorunlara neden olabilir. Tersi olarak kontrolsüz yaptıklarımız da benzer sorunlara neden olabilmektedir. İşte anahtar konuda değişikliğin bir disiplin ve düzenle uygulanması işlemidir.

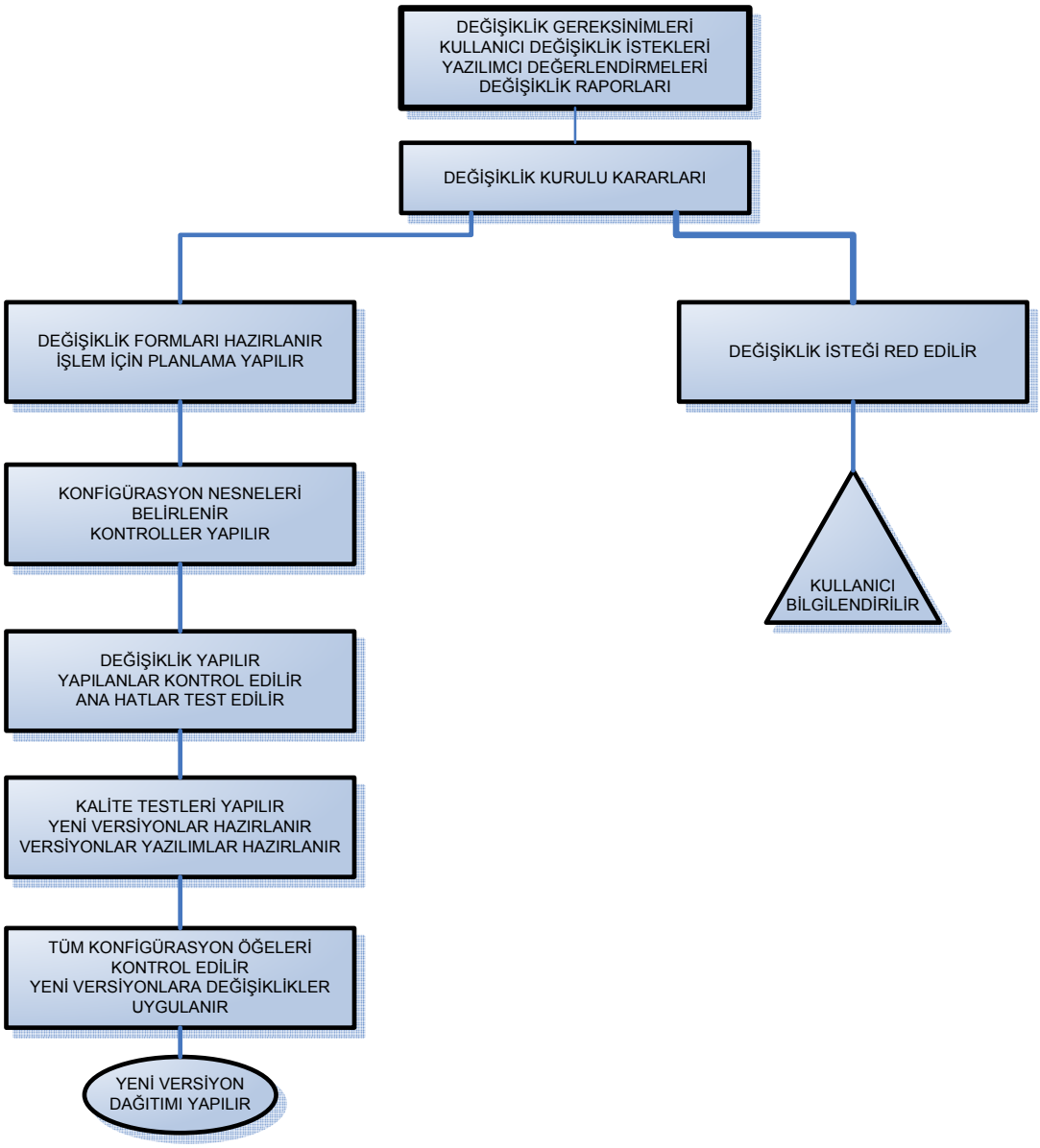
Büyük çaplı yazılım projelerinde kontrolsüz yapılan değişiklikler çok önemli kaos durumlarına düşmemize yolaçabilmektedir. Bu tarz projelerde , insani unsurlar ve otomasyon araçlarını karma olarak değişiklik kontrol sürecinde kombine ederek kullanmalıyız. Değişiklik kontrol sürecini bir ağaç yapısı şeklinde özetlememiz işimizi iş akışı açısından kolaylaştırabilir.

Değişiklik İsteklerini teknik faydaları, potansiyel etkileri, sistem fonksiyonları ve konfigürasyon öğelerine etkileri ve proje maliyeti açısından ele alırız, değerlendiririz. Değerlendirmenin sonucu değişiklik raporu olarak düzenlenir. Bu rapor Değişiklik Kontrol Kurul'una değerlendirme sonucu olarak rapor edilir. Önceliklendirmesi ve kararın durumu Kurul'a iletilir. Kurul tarafından onaylanan her değişiklik, Mühendislik Değişiklik Emri olarak yayımlanır. Bu tip onaylanan emirler değerlendirme, yapılma kriterleri ve test değerleri olarak takip sistemine kayıt edilir.

Yazılım Mühendisleri tarafından gerçekleştirilen değişiklikler bir dizine değişiklik işlemi yapıldı kodu ile eklenir. Versiyon kontrol sistemi orjinal dosya ve veri tabanlarını yapılan değişiklerle günceller.

Bu versiyon kontrol mekanizması , değişiklik kontrol süreçlerinin uygulaması açısından erişim kontrol ve senkronizasyonun sağlanması açısından çok önemli iki unsuru içerir. Erişim kontrol yazılım mühendislerine konfigürasyon öğelerine erişim yetkisini ve bunlar üzerinde değişim yapma yetkisini sağlar. Senkronizasyonda paralel olarak birçok kişinin birbirinden etkilenmeden işlemleri uyum içinde yapmasını temin eder.

Bu ağaç yapısının üzerindeki tüm süreçler adım adım bir değişiklik isteğinin uygulanması için takip edilir. Yapılan resmi değişiklik kontrol adımları sonucunda ürünün değişiklik isteklerinin üzerinde uyarlandığı şekli olan yeni ürün kullanıma ve dağıtıma hazır hale gelir . (Şekil 6.5)



Şekil 6.5 Değişiklik Kontrol Süreci Ağaç yapısı

6.5.4. Konfigürasyon Denetimi

Bundan önceki adımlarda yazılımcının yapmış oldukları gelen isteklerin biraz belirsizlik ve kaos ortamlarından kurtulup , düzen içinde çalışmalarını yapmasını sağlar. Bununla beraber etkili ve çağdaş bir

denetiminde kesinleşen değişiklik isteklerinde uygulaması gerektirdiği çok açıktır. Bu isteklerin uygun bir biçimde nasıl uyarlanabileceğinin yanıtı ise 1) resmi teknik değerlendirmelerin 2) yazılım konfigürasyon denetimlerinin tam yapılmasındadır.

Resmi teknik değerlendirmede yapılan değişikliklerin teknik olarak konfigürasyon öğelerine nasıl etki ettiğini ortaya koyarız. Değerlendiriciler teknik açıdan uygulanacak olan değişikliklerin etkileri, potansiyel yan etkilerini ele alıp ortaya koymaya çalışırlar.

Yazılım konfigürasyon denetimi ile resmi olarak aşağıdaki sorulara yanıtlar aranır.

- İstenen resmi değişiklik yapıldı mı , ek değişiklikler gerekecek mi ?
- Teknik açıdan yapılanların doğruluğu ve uygulama irdelendi mi ?
- Önerilen yazılım mühendisliği standartları ve süreçleri uygun bir biçimde değişikliklere uygulandı mı ?
- İstenen değişiklikler tam olarak etkilenen öğelere uygunlandı mı , uygulananlar vurgulanarak belgelere tam olarak yansıtıldı mı ?
- Yazılım Konfigürasyon Yönetim prosedürleri değişikler üzerinde not edilerek kayıtlar alındı ve raporlama işlemi yapıldı mı ?
- İlgili tüm yazılım konfigürasyon öğeleri yapılan değişikliklere uygun şekilde güncellendi mi ?

Bunlara verilecek arama yanıtları ise süreç ve işlemlerin denetimlerinin sağlıklı yapılması açısından kritik değerdedir.

6.5.5. Raporlama

Nelerin yapıldığına , kimin yaptığına , ne zaman yapıldığına ve nelerin etkilendiğine verdiğimiz yanıtların tamamı bir sürecin raporlama düzeyini tanımlar. Konfigürasyon durum raporlaması bu bilgi akışının yapılmasını bir belgeleme düzenine sokmasını sağlar. Bu yapmış olduğumuz değişiklik isteğinden bunların versiyonlanarak yapılmasına , test edilmesine, onaylanmasına ortamlara taşınmasına kadar olan tüm süreçte belgelenmesine ve dolayısı ile raporlanmasına yardımcı olur. Bu raporlardan elde edilecek sonuçların sisteme geri beslenmesi ile de sistemin kendini iyileştirmesi , yapılan faaliyetlerin hem mühendislik hemde maliyet açısından proje incelenmesine alınmasına neden olur. Raporlardan elde edilen değerlendirme sonuçları ile sistem geri besleme mekanizması çalışır. Bu geri besleme yolu ise sonraki adımlarda yazılım yaşam döngüsü açısından bir iyileştirme , sistem güncel tutma ve yaşama daha uyumlu bir sistem elde etmenin olmazsa olmaz koşuludur.

SONUÇ

Yazılım yaşam döngüsü ile değişiklik/ler yaşamın doğrudan bizlere yansıması gerçeğidir. Bizler yazılımları tasarlarken , gerçekleştirirken ve sonrasında onları yürütüp, yaşatırken hiç istemesek de yeni iş istekleri , sorunlar, hatalar ve her boyutta değişiklik isteği ile heran karşı karşıya kalmaktayız. Bunların yazılım geliştiriciler olarak karşımıza çıkması, tasarladığımız yazılım yada yazılım ürünlerinin potansiyel olarak sorunlu olması demek değildir. Değişiklik ve değişiklik isteği yaşam dinamiğinin içinde varolan bir gereksinim olarak karşımıza çıkmaktadır.

Bu noktada işte pozitif ve bilimsel yaklaşım olarak Yazılım Mühendisliği bilimi çıkar. Özellikle bu disiplinim amacı değişikliklerin mevcut, yaşayan yazılım sistemlerine uygulanması ile ilgili sorunlara çözümler bulmaktır.

Araştırmamızda , çalışmalarımızda yazılım ve yazılım sistemlerine gelen bu değişiklik isteklerinin uygulanması sonucundaki sorun , problem, sıkıntı , zorluklar, karmaşa , zaman sıkıntısı , hesaplanan ve hesaplanmayan ek maliyetler gibi öğeleri yönetebilecek çözümler bulabilmeyi hedefledik. Yazılım geliştirmede yazılımın toplam ömrünün % 60-70 gibi çok yüksek bir orandaki kısmını bizler yazılım geliştiriciler olarak yazılımların bakımları , destekleri, sorunlarının düzeltilmesi ve yeni gelen istekleri uygulamada harcadığımızı gözlemledik. Bu oranı eğer klasik üretim sektörleri için bir ürün geliştirme sürecine uygulamış olsaydık çok önemli bilimsel ve ticari zararlarla karşılaşmış olurduk. Bu oranın neden bu kadar çok

yüksek olduğu kadar nasıl daha iyi bir yazılım yönetimi ile yazılım geliştirmenin ele alınması gereğide kaçınılmaz bir yazılım mühendisliği sorunudur.

İşte amacımıza ulaşmak için hatalarımızı en aza indirgemek , sorunlarımızı azaltmak , yazılım üretkenliğimizi arttırmak için kayıtlı olarak tüm yazılım öğelerimizi yani yazılım konfigürasyonlarımızı bir plan dahilinde kalite kontrolü , güvencesi ile sağlayacağımızı bilmekteyiz.

Özellikle bilimsel açıdan başarılı olan bir yazılım geliştirme süreci içinde Konfigürasyon Yönetimi ve Planlaması çok önemlidir. Burada yaşayan yazılımın tüm yazılım konfigürasyon öğelerini modern araç ve gereçlerle takip etmeliyiz. Bunun için yazılım geliştirme ortamlarımızdaki yazılımlara gömülü olarak çalışan bir veri tabanı ile entegre çalışan her türlü yazılım geliştirme faaliyetimizi kayıtlı izleyen bir yardımcı yazılım aracı gereklidir. Bu araç kendi yazılım ekibimiz veya başka dış kaynaklar yardımı ile geliştirilebilir , sağlanabilir. Bu aracın varlığı demek Konfigürasyon Yönetimi yapıyoruz demek değildir. Bu araçla beraber gelen değişiklik isteklerini yönetmemizi , organize etmemizi , disiplin altında yürütmemizi, takibe almamızı sağlayacak bir yaklaşıma gereksinim duyarız. Bu yapılanmayı hem bir plan organizasyonu hemde bir yazılım geliştirme kültürü olarak yapımızda tanımlamalıyız. Ardından da özellikle tüm kurumsal yapısı ve kurumları ile tanımladığımız bu planı mutlak Üst Yönetimin desteğini sağlayarak kalite planlarımızdan da ödün vermeden uygulamamız gereklidir.

Bakınca ilk aşamada dışarıdan yazılım üretim sürecini uzatan , zorlaştıran bir yaklaşım gibi gelen bu Yazılım Konfigürasyon Yönetim planlaması aslında orta ve büyük ölçekli yazılım geliştirme takımları için kaos , belirsizlik, tanımsızlık ve ürün

kalite, kalite güvence sorunlarını yöneten bir bilimsel disipliner yazılım geliştirme üretkenlik çözümü olduğu tarafımızdan tespit edilmiştir.

Yazılım Konfigürasyon Yönetimiminin araştırma çalışmamızda bize sağladığı çözümler :

- Değişikliklerin tanımlanmasıdır,
- Değişikliklerin anlanması ve kavranmasıdır,
- Denetlenmesidir,
- Garantili olarak güvence ile teminidir,
- Raporlanıp takibidir ve kayıtlanmasıdır,
- İlişkilendirilmesidir ,
- Kaynak , üretim ve kalite planları açısından sağlıklı olarak oluşmasıdır,
- Tüm yazılım sistemleri için bir geri besleme süreci olmasıdır,
- Planlama , yönetme ve doktiriner olarak kavramdan pratiğe geçmesidir.

Yazılım Konfigürasyon Yönetimini bilimsel , modern , üretken , kurumsal , yapıcı , geliştirici , ileriye doğru götüren , dinamik , karlı ve tüm yazılım süreçlerini pozitif destekleyen bir çözüm olarak yazılım geliştirme ve sonrasında yürütme aşamalarında araştırma çalışma sonucumuz olarak görmekteyiz.

Bu kayıtlı yazılım takip sisteminin bir kavram olmaktan çok yazılım geliştirme uygulama çözümü olarak tavsiye ediyoruz. Tamamen dinamik ve kendi yazılım gereksinimleri ile gelişmeye açık her bir yazılım sistemi gereksinimine göre

konfigürasyon öğelerinin yazılıma özel uygulanabilecek yapıda olduğunu tespit etmekteyiz.

Kavramsal olarak ise kayıt , planlama , proje yönetimi, ürün – sonuç odaklı , kalite anlayışı ile nitelikli yazılım geliştiricilerle sürekli olarak Yazılım Geliştirme ekipleri için Üst Yönetimleri tarafından da çözüm olması sebebi desteğin vazgeçilmez olduğunu ifade etmekteyiz.

Sağlıklı yaşayan ve yaşayacak olan yazılımlarımız için yazılımlarımızın modern Yazılım Konfigürasyon Yönetim Planları ile değişime ayak uydurmaları vazgeçemeyeceğimiz derecede kritik ve değerli iş kavramımızdır.

KAYNAKLAR:

1.Adams, R. John , Principles of Project Management with a new Introduction, PMI Publications , 1997

2.Berczuk , S.P. , Appleton , B. , Software Configuration Management Patterns: Effective Teamwork , Practical Integration , Addison Wesley , 2002

3.Davis ,A. M. , Software Requirements : Objects, Functions &States , Prentice Hall , 1993

4.Eliens , A., Principles of Object-Oriented Software Development , Addison Wesley Publication, 2002

5.Humprey , Walter S. , A Dicipline for Software Engineering , Addison Wesley , 1995

6.Kendall Kenneth E. , Kendall , Julie E., System Analysis and Design , 2005

7.Kotonya , G. And Sommerville , I. , Requirements Engineering : Processes and Techniques , John Wiley & Sons., 1998

8.Laudon , Kenneth C. , Laudon , Jane P. , Management Information Systems – Managing The Digital Firm , 2002

9.Ould , M. A. , Business Processes : Modelling and Analysis for Re-engineering and Improvment, John Wiley & Sons. , 1995

10.Pressman , Roger S., Software Engineering A Practitioner's Approach , 2005

11.Sommerville , Ian , Software Engineering , Pearson – Addison Wesley Publication, 2004

12.Thomas , P and Weedon R. , Object-Oriented Programming in Eiffel, Addison Wesley Publication, 2003

13.White , B. A. , Software Configuration Management Strategies and Rational ClearCase , Addison Wesley , 2000

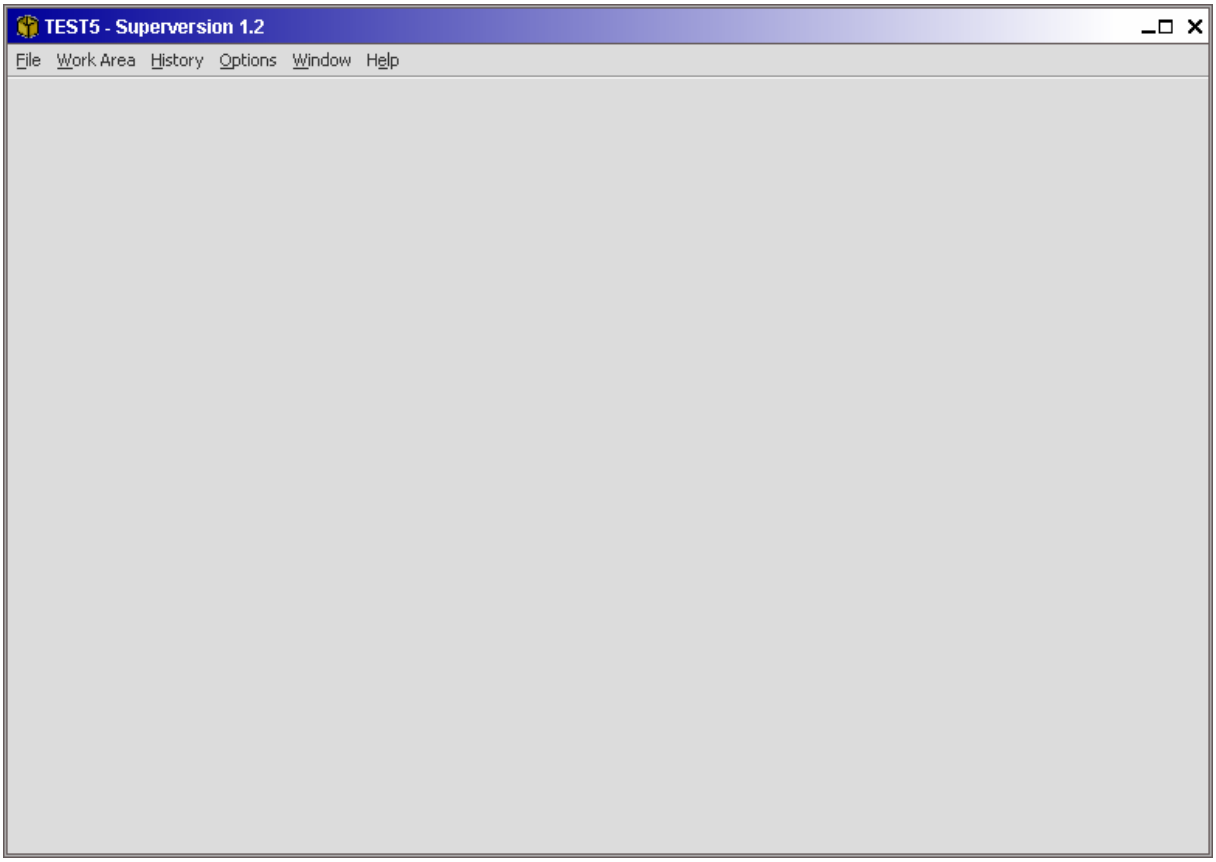
14.Whitgift , D. , Software Configuration Management : Methods and Tools , Addison Wesley , 1990

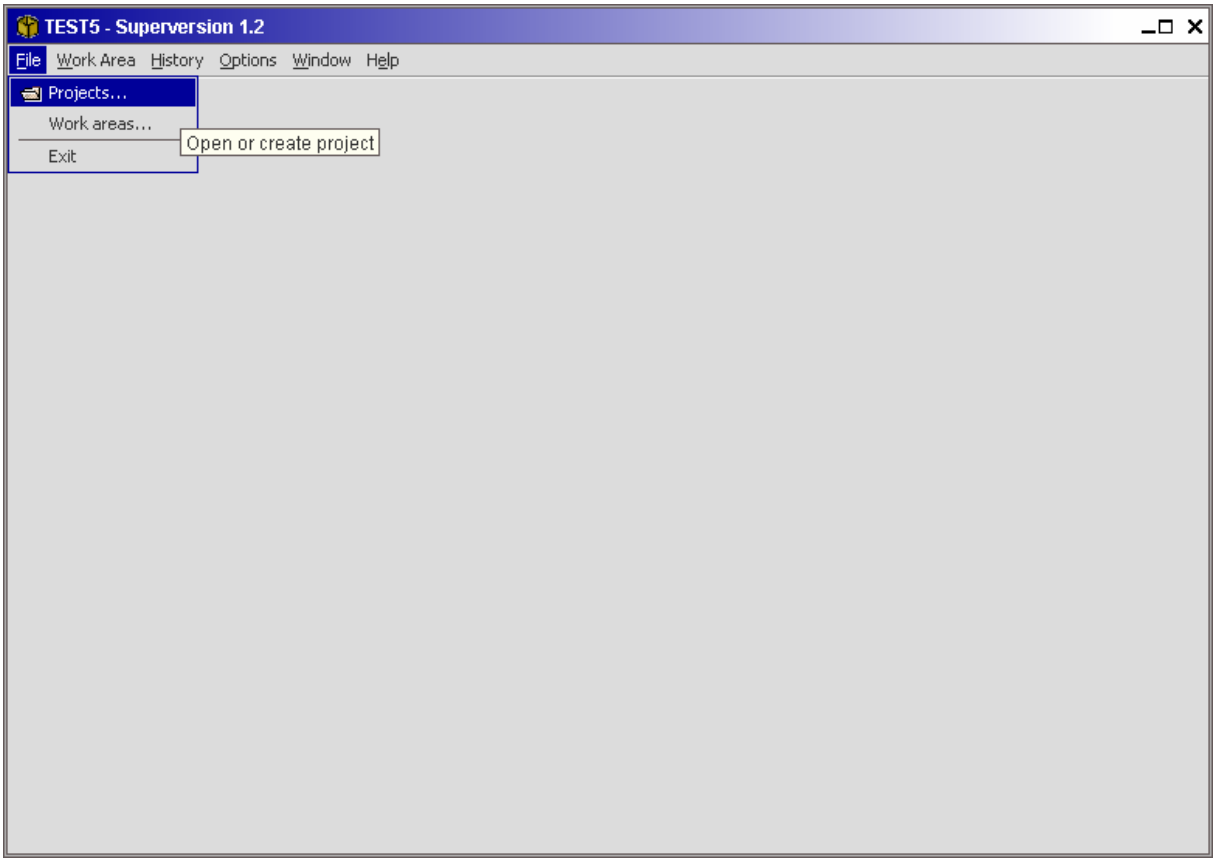
ÖZGEÇMİŞ

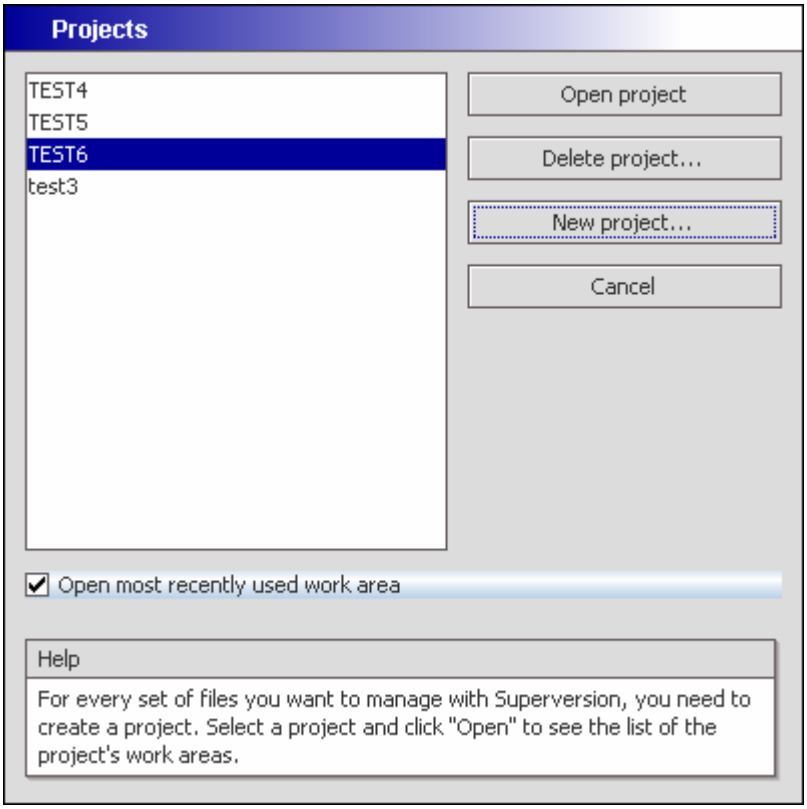
R.Reha ÇETİN, 01 Ocak 1964 yılında Gaziantep'de doğdu. 1982 yılında Kuleli Askeri Lisesi'ni bitirdi. 1986 yılında Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nden mezun oldu. 1986 yılında Ege Üniversitesi Bilgisayar Mühendisliği Bölümünde Araştırma Görevlisi ve Sistem Programcısı olarak çalışmaya başladı. Yüksek lisansını 1988 yılında aynı Bölüm'de tamamladı. Ege Üniversitesi'nde Doktora çalışmalarına başladı , ardından Trakya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Ana Bilim Dalı'nda doktora çalışmalarına devam etti. Bu arada Askerlik görevi sonrasında Türkiye'de 1992-2007 yılları arasında çeşitli finans kuruluşlarının Bilgi Teknolojileri Bölümlerinde teknik ve yönetici pozisyonlarında görev yaptı. Şu anda R.Reha ÇETİN bir finans kuruluşunda Bilgi Teknolojileri Bölüm Direktörü olarak görev yapmaktadır.

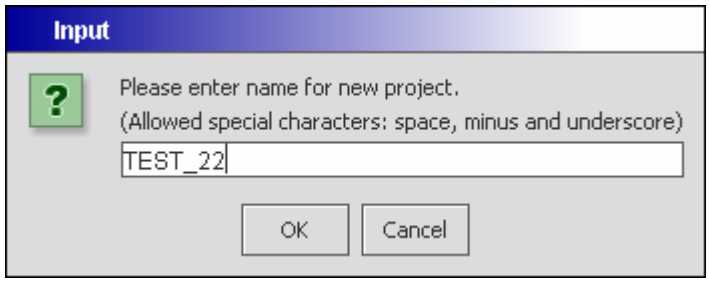
EKLER

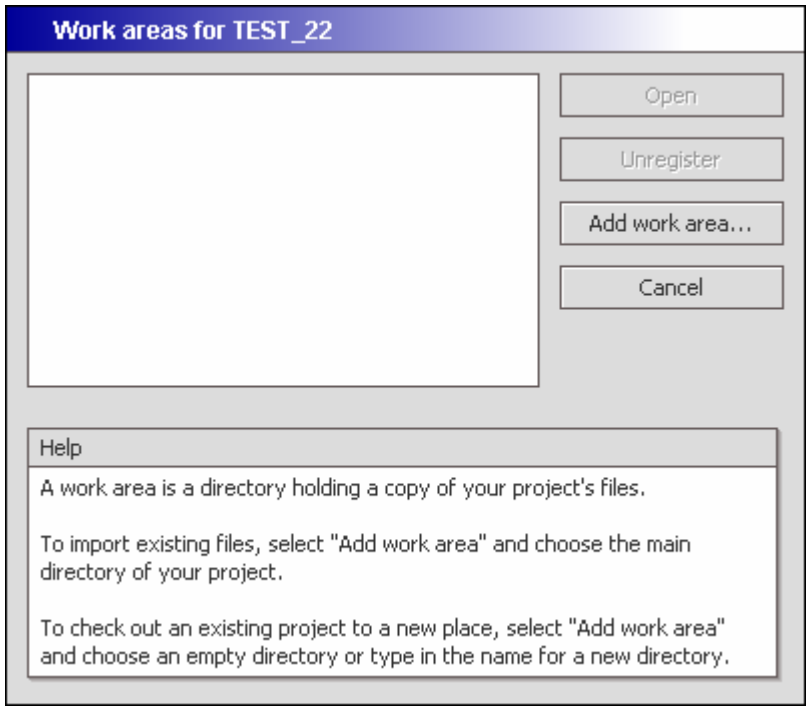
EK 1 : Ekran Örnekleri

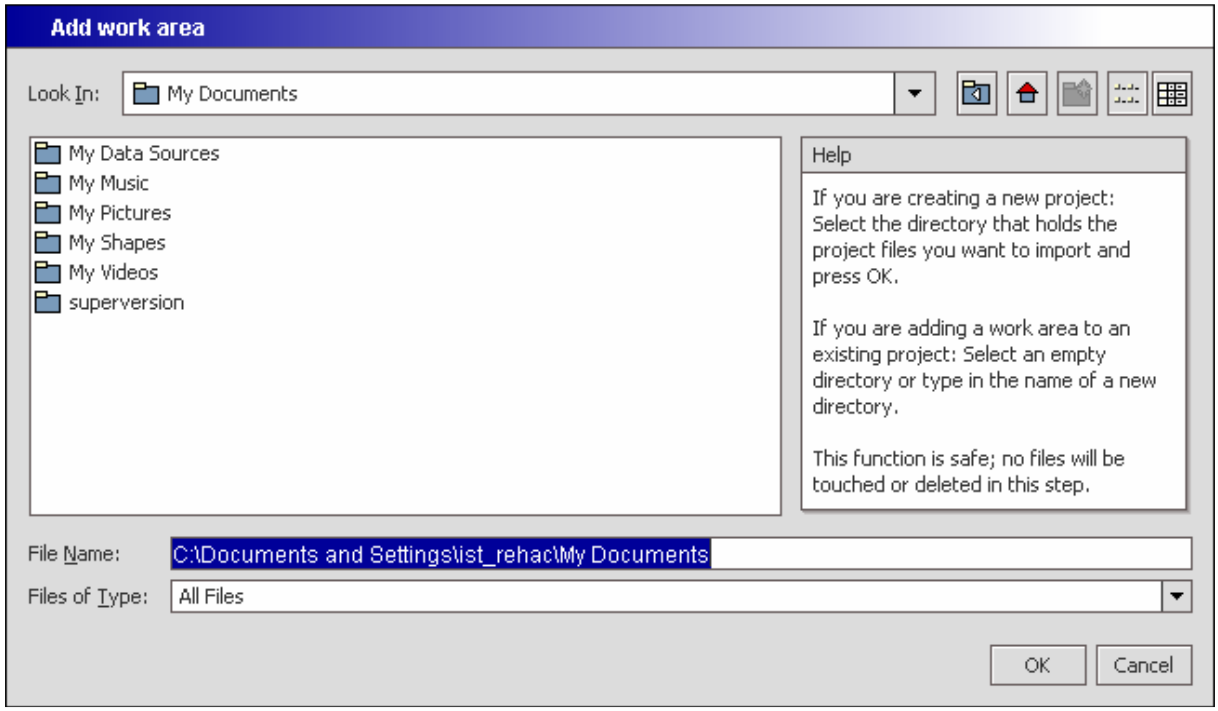


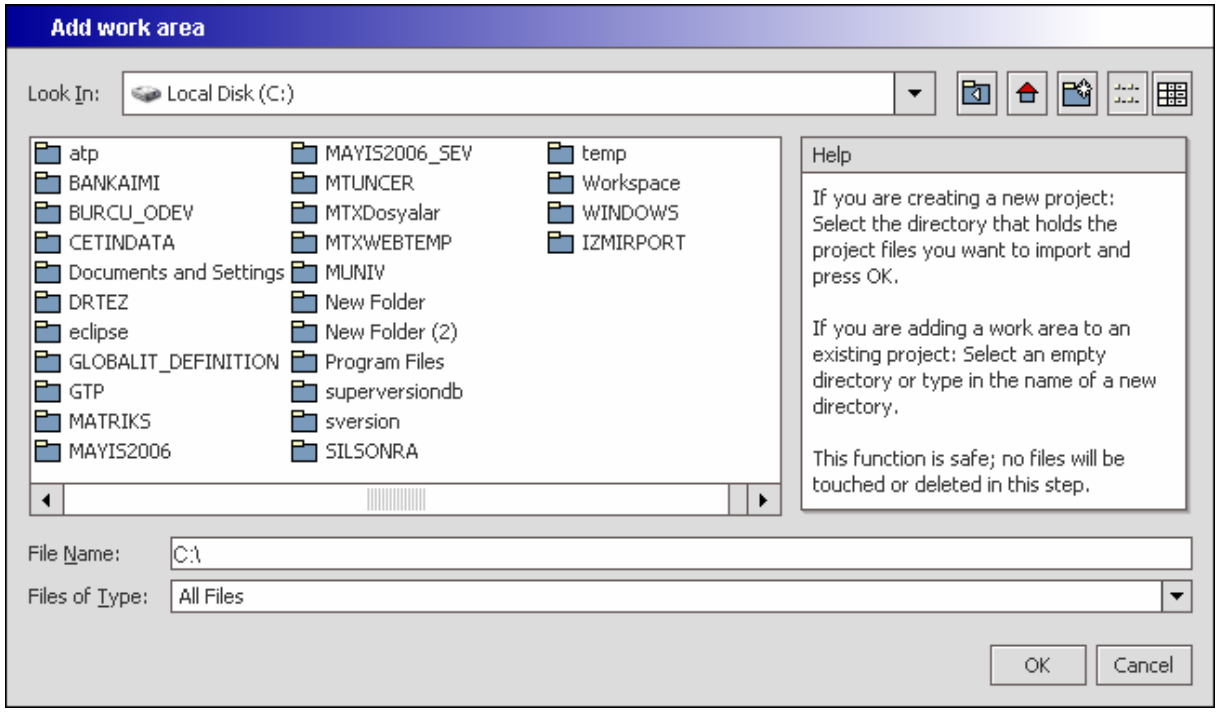

TEST5 - Superversion 1.2
File
Work Area
History
Options
Window
Help
Projects...
Work areas...
Exit
Open or create project

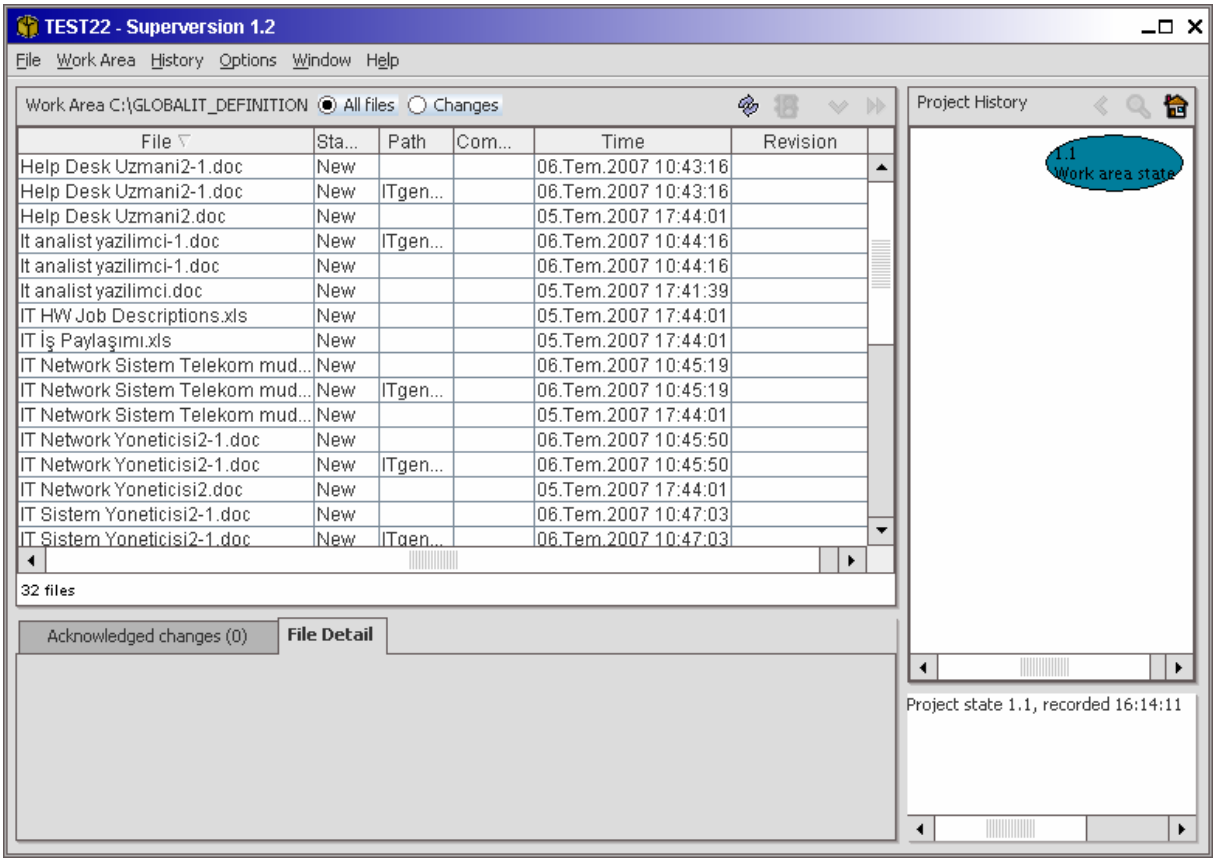

Projects
TEST4
TEST5
TEST6
test3
Open project
Delete project...
New project...
Cancel
Open most recently used work area
Help
For every set of files you want to manage with Superversion, you need to create a project. Select a project and click "Open" to see the list of the project's work areas.

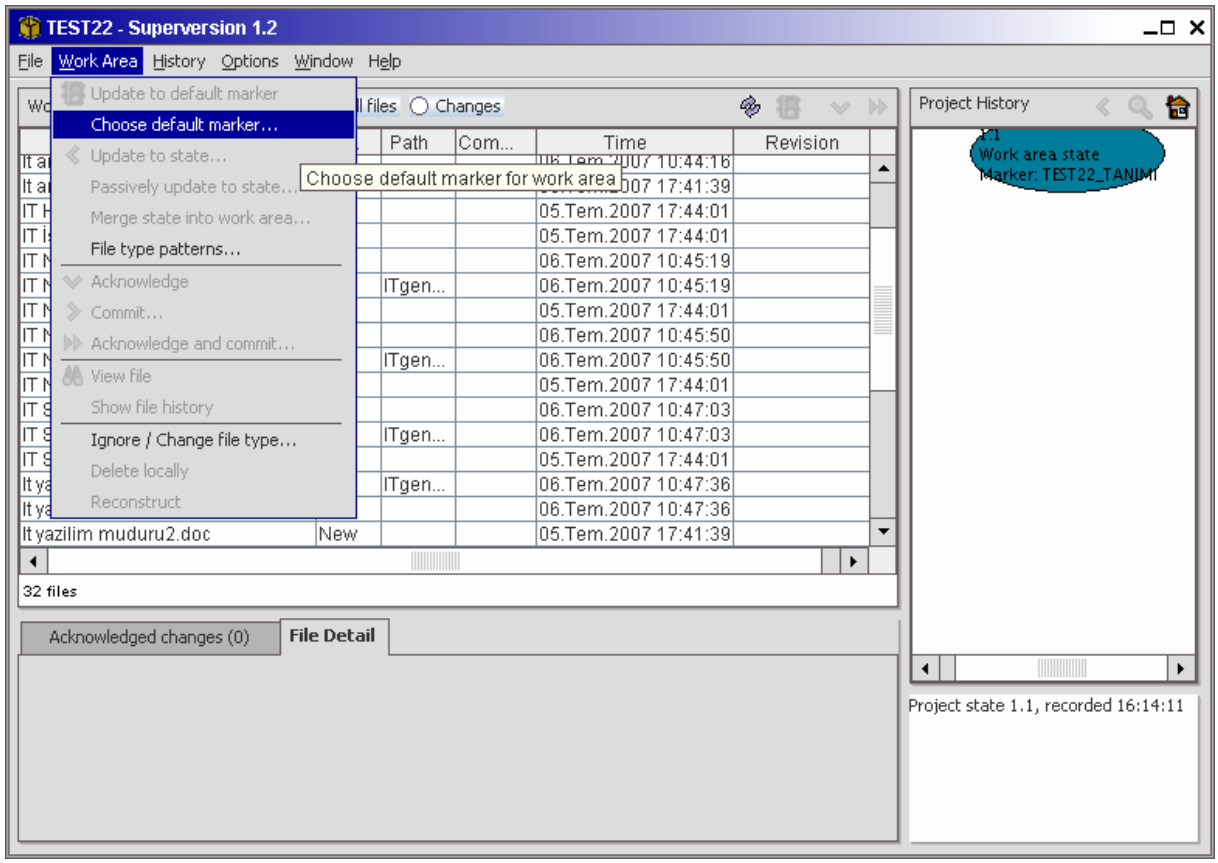

Input
Please enter name for new project.
(Allowed special characters: space, minus and underscore)
TEST_22
OK
Cancel

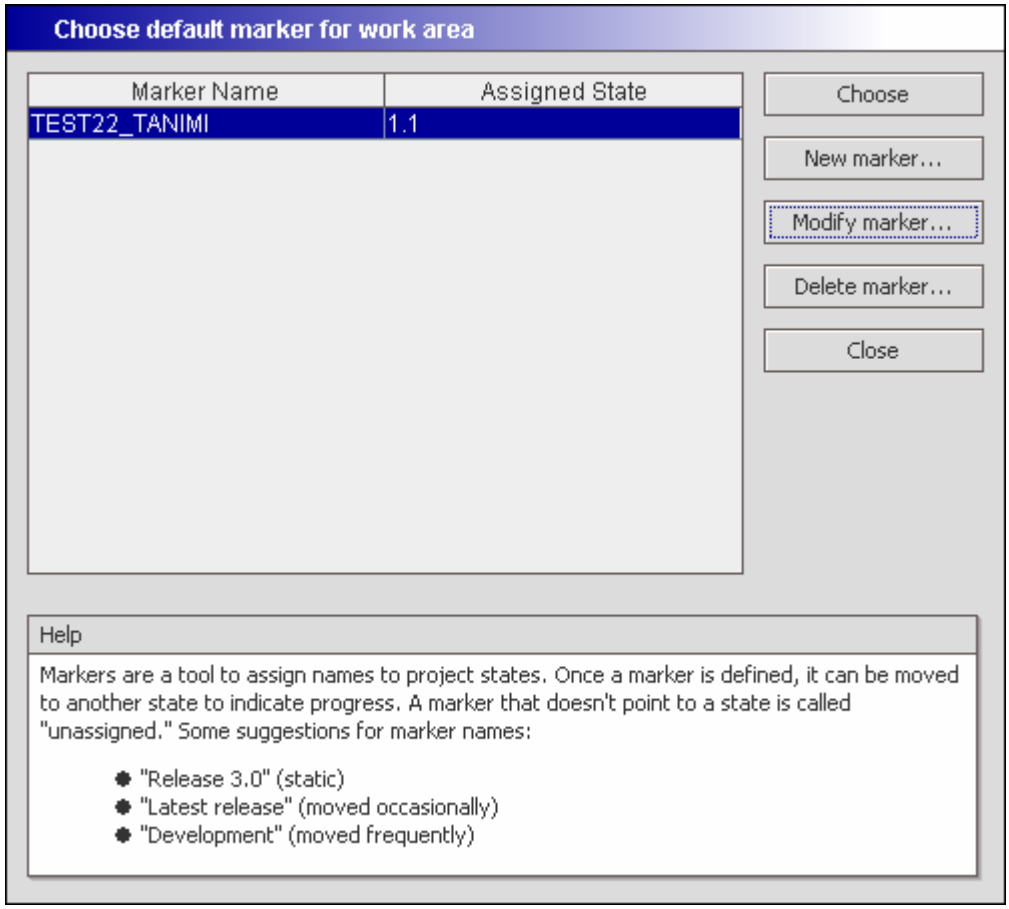

Work areas for TEST_22
Open
Unregister
Add work area...
Cancel
Help
A work area is a directory holding a copy of your project's files.
To import existing files, select "Add work area" and choose the main directory of your project.
To check out an existing project to a new place, select "Add work area" and choose an empty directory or type in the name for a new directory.

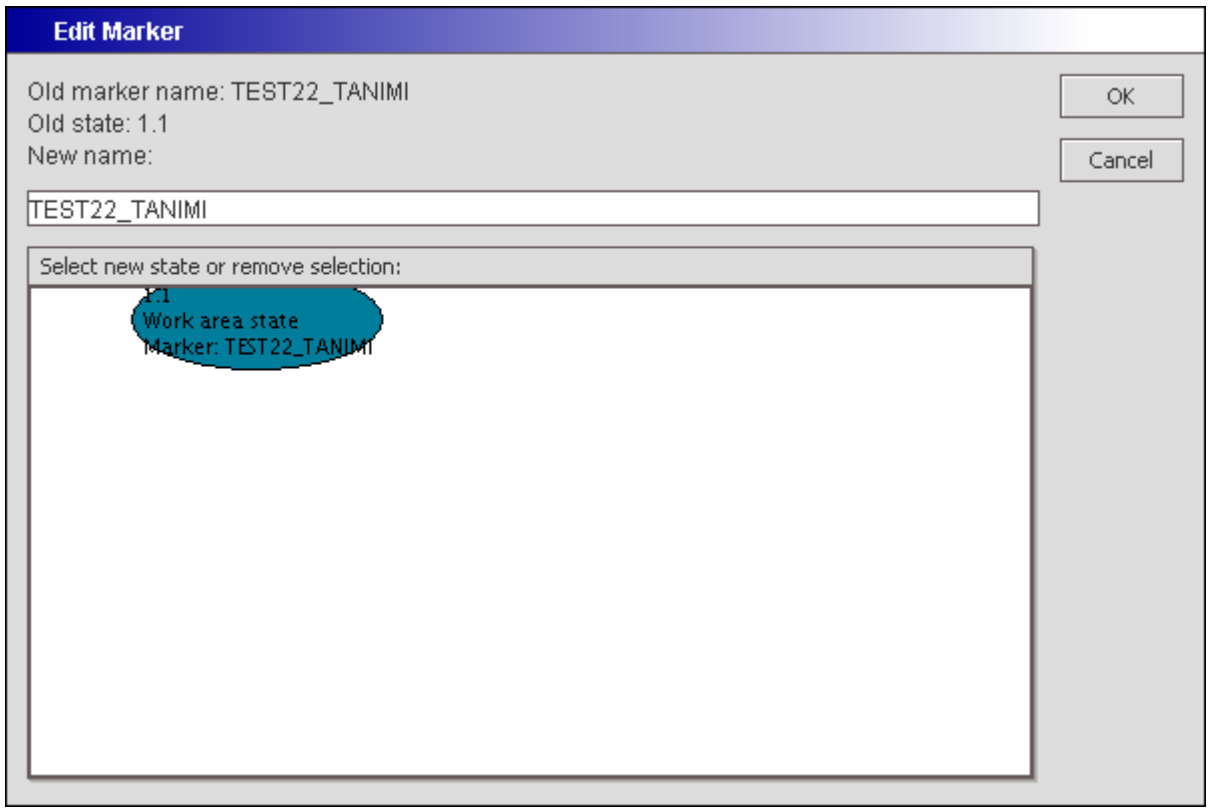

Add work area
Look In:
My Documents
My Data Sources
My Music
My Pictures
My Shapes
My Videos
superversion
Help
If you are creating a new project: Select the directory that holds the project files you want to import and press OK.
If you are adding a work area to an existing project: Select an empty directory or type in the name of a new directory.
This function is safe; no files will be touched or deleted in this step.
File Name:
C:\Documents and Settings\ist_rehac\My Documents
Files of Type:
All Files
OK
Cancel

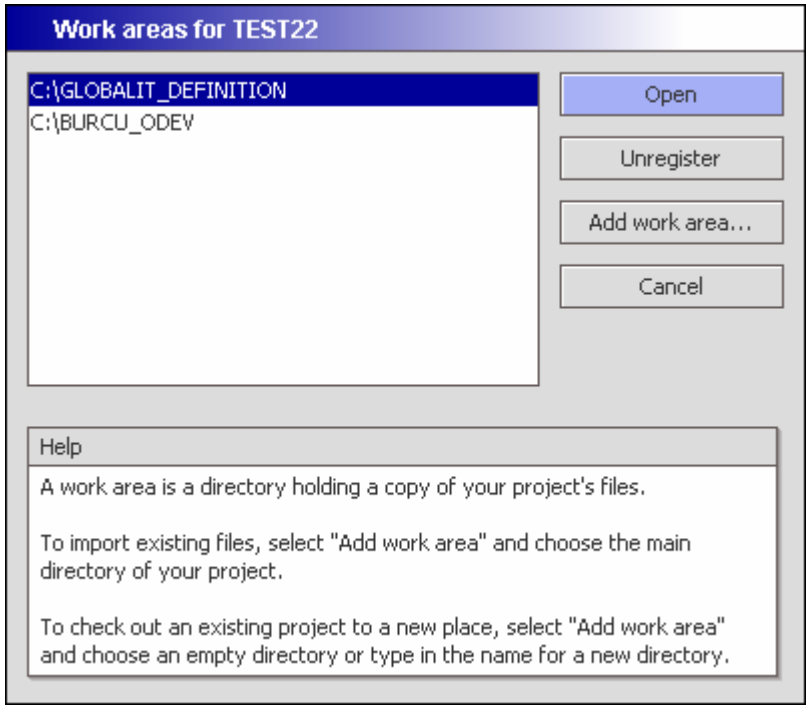

Add work area
Look In:
Local Disk (C:)
atp
BANKAIMI
BURCU_ODEV
CETINDATA
Documents and Settings
DRTEZ
eclipse
GLOBALIT_DEFINITION
GTP
MATRIKS
MAYIS2006
MAYIS2006_SEV
MTUNCER
MTXDosyalar
MTXWEBTEMP
MUNIV
New Folder
New Folder (2)
Program Files
superversiondb
sversion
SILSONRA
temp
Workspace
WINDOWS
IZMIRPORT
Help
If you are creating a new project: Select the directory that holds the project files you want to import and press OK.
If you are adding a work area to an existing project: Select an empty directory or type in the name of a new directory.
This function is safe; no files will be touched or deleted in this step.
File Name:
C:\
Files of Type:
All Files
OK
Cancel

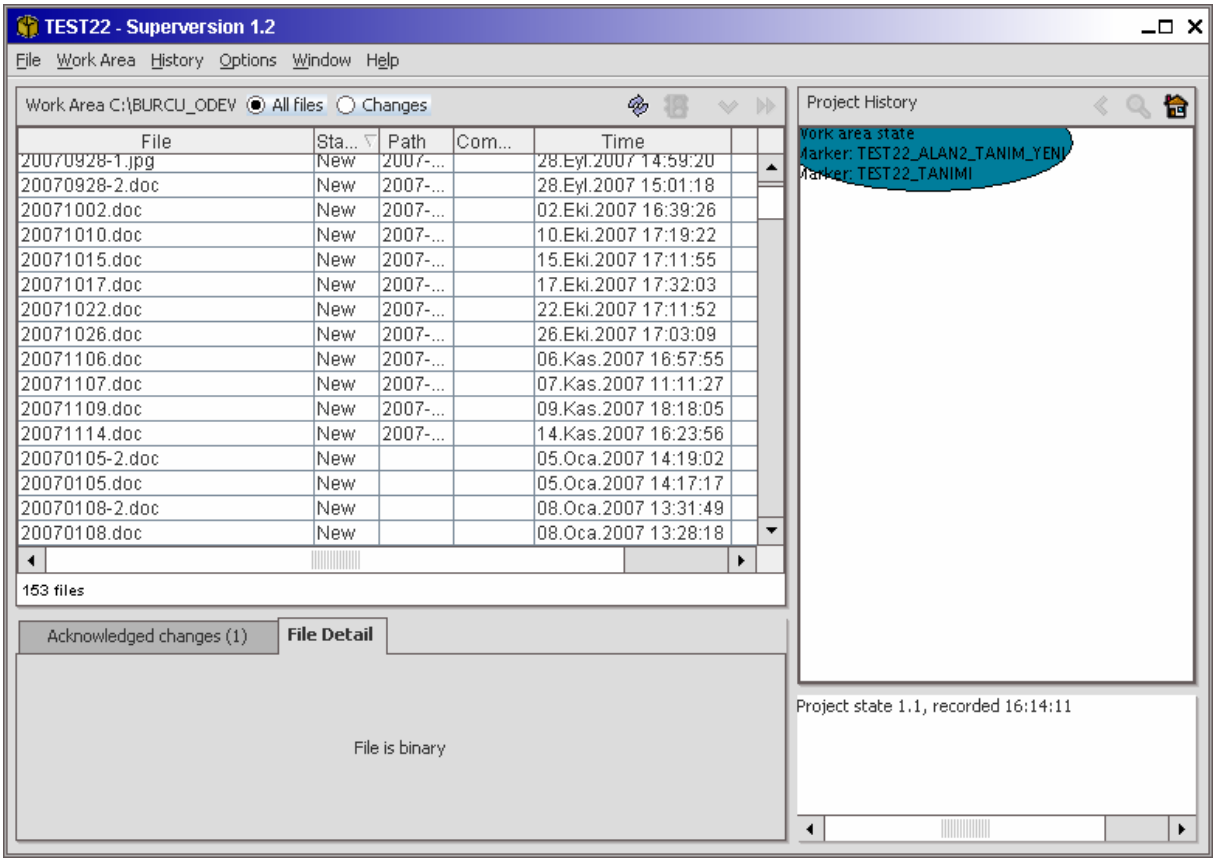

TEST22 - Superversion 1.2
File Work Area History Options Window Help
Work Area C:\GLOBALIT_DEFINITION All files Changes
File Sta... Path Com... Time Revision
Help Desk Uzmani2-1.doc New 06.Tem.2007 10:43:16
Help Desk Uzmani2-1.doc New ITgen... 06.Tem.2007 10:43:16
Help Desk Uzmani2.doc New 05.Tem.2007 17:44:01
It analist yazilimci-1.doc New ITgen... 06.Tem.2007 10:44:16
It analist yazilimci-1.doc New 06.Tem.2007 10:44:16
It analist yazilimci.doc New 05.Tem.2007 17:41:39
IT HW Job Descriptions.xls New 05.Tem.2007 17:44:01
IT İş Paylaşımı.xls New 05.Tem.2007 17:44:01
IT Network Sistem Telekom mud... New 06.Tem.2007 10:45:19
IT Network Sistem Telekom mud... New ITgen... 06.Tem.2007 10:45:19
IT Network Sistem Telekom mud... New 05.Tem.2007 17:44:01
IT Network Yoneticisi2-1.doc New 06.Tem.2007 10:45:50
IT Network Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:45:50
IT Network Yoneticisi2.doc New 05.Tem.2007 17:44:01
IT Sistem Yoneticisi2-1.doc New 06.Tem.2007 10:47:03
IT Sistem Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:47:03
32 files
Acknowledged changes (0)
File Detail
Project History
1.1
Work area state
Project state 1.1, recorded 16:14:11

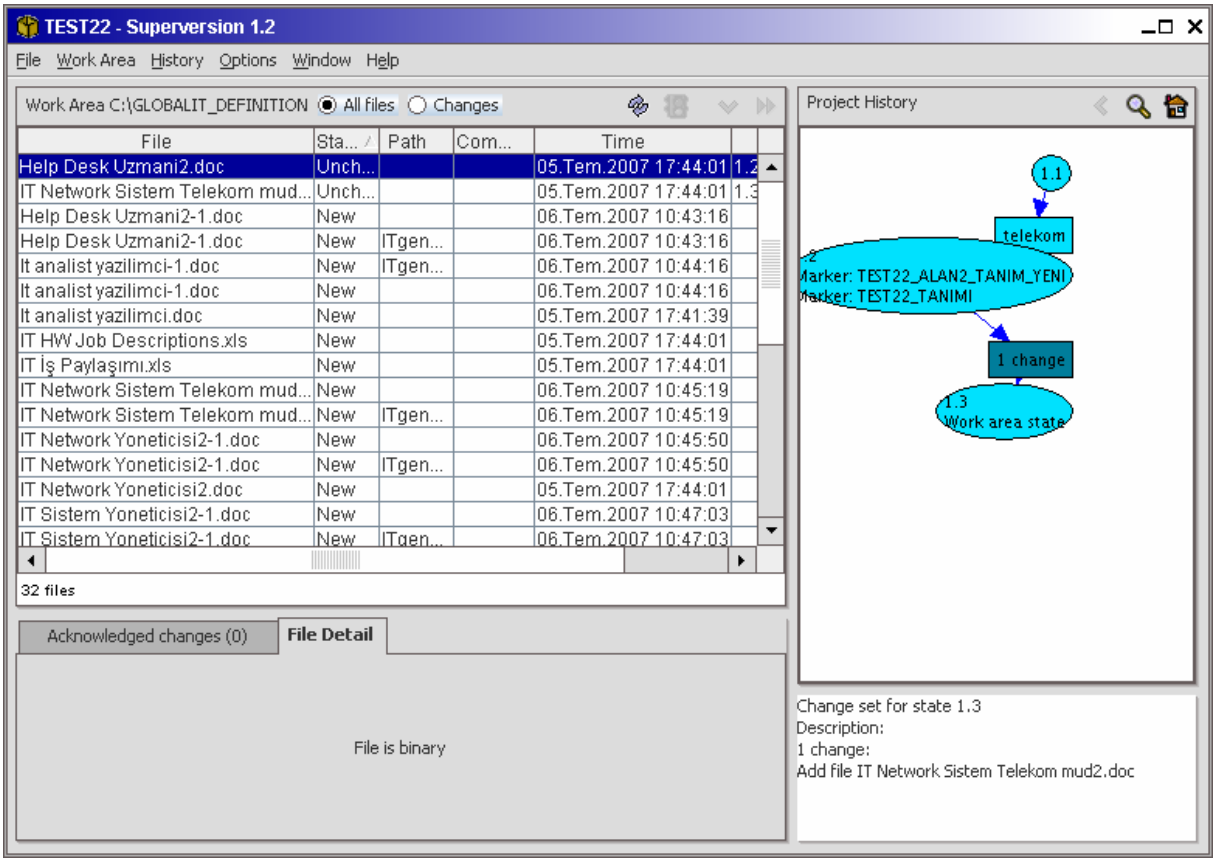

TEST22 - Superversion 1.2
File
Work Area
History
Options
Window
Help
Update to default marker
Choose default marker...
Update to state...
Passively update to state...
Merge state into work area...
File type patterns...
Acknowledge
Commit...
Acknowledge and commit...
View file
Show file history
Ignore / Change file type...
Delete locally
Reconstruct
Choose default marker for work area
Path
Com...
Time
Revision
06.Tem.2007 10:44:16
05.Tem.2007 17:41:39
05.Tem.2007 17:44:01
05.Tem.2007 17:44:01
06.Tem.2007 10:45:19
06.Tem.2007 10:45:19
05.Tem.2007 17:44:01
06.Tem.2007 10:45:50
06.Tem.2007 10:45:50
05.Tem.2007 17:44:01
06.Tem.2007 10:47:03
06.Tem.2007 10:47:03
05.Tem.2007 17:44:01
06.Tem.2007 10:47:36
06.Tem.2007 10:47:36
It yazilim muduru2.doc
New
05.Tem.2007 17:41:39
32 files
Acknowledged changes (0)
File Detail
Project History
1.1
Work area state
Project state 1.1, recorded 16:14:11

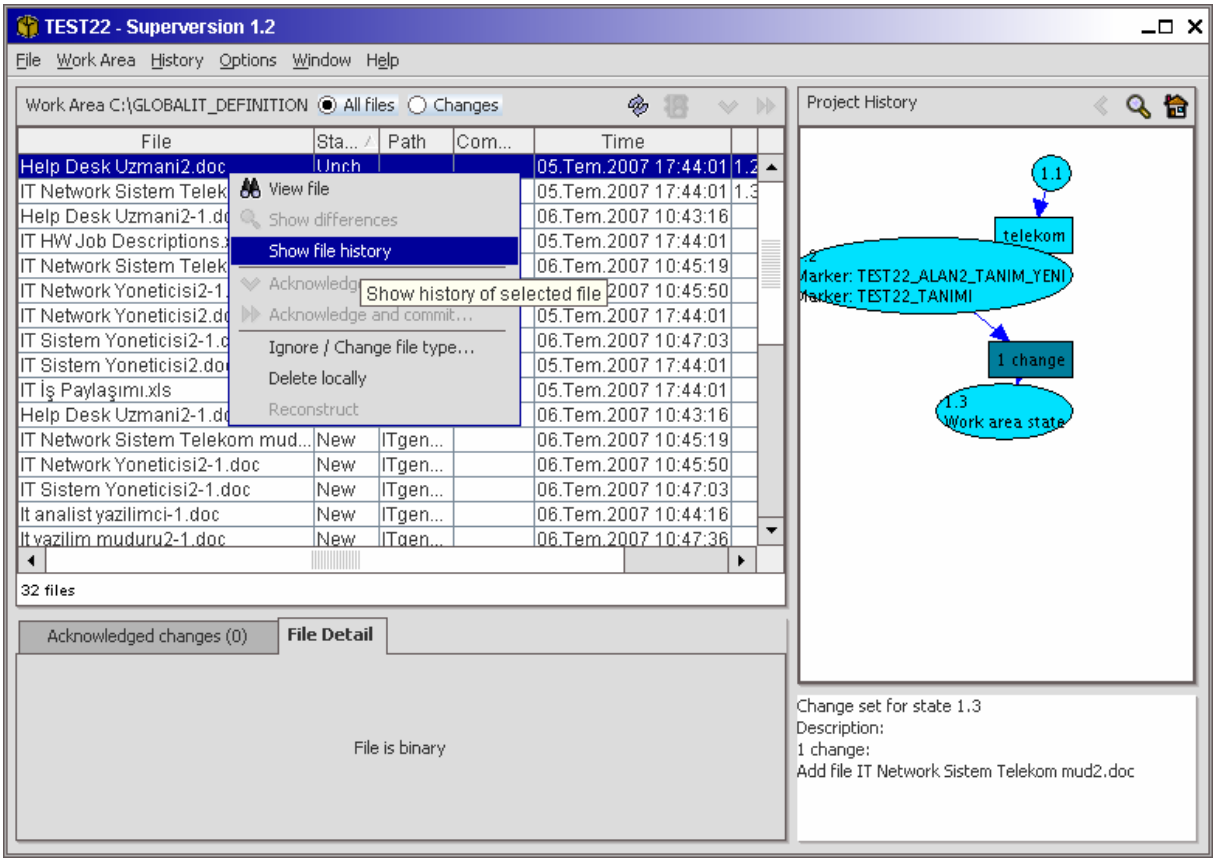

Choose default marker for work area
Marker Name
Assigned State
TEST22_TANIMI
1.1
Choose
New marker...
Modify marker...
Delete marker...
Close
Help
Markers are a tool to assign names to project states. Once a marker is defined, it can be moved to another state to indicate progress. A marker that doesn't point to a state is called "unassigned." Some suggestions for marker names:
"Release 3.0" (static)
"Latest release" (moved occasionally)
"Development" (moved frequently)

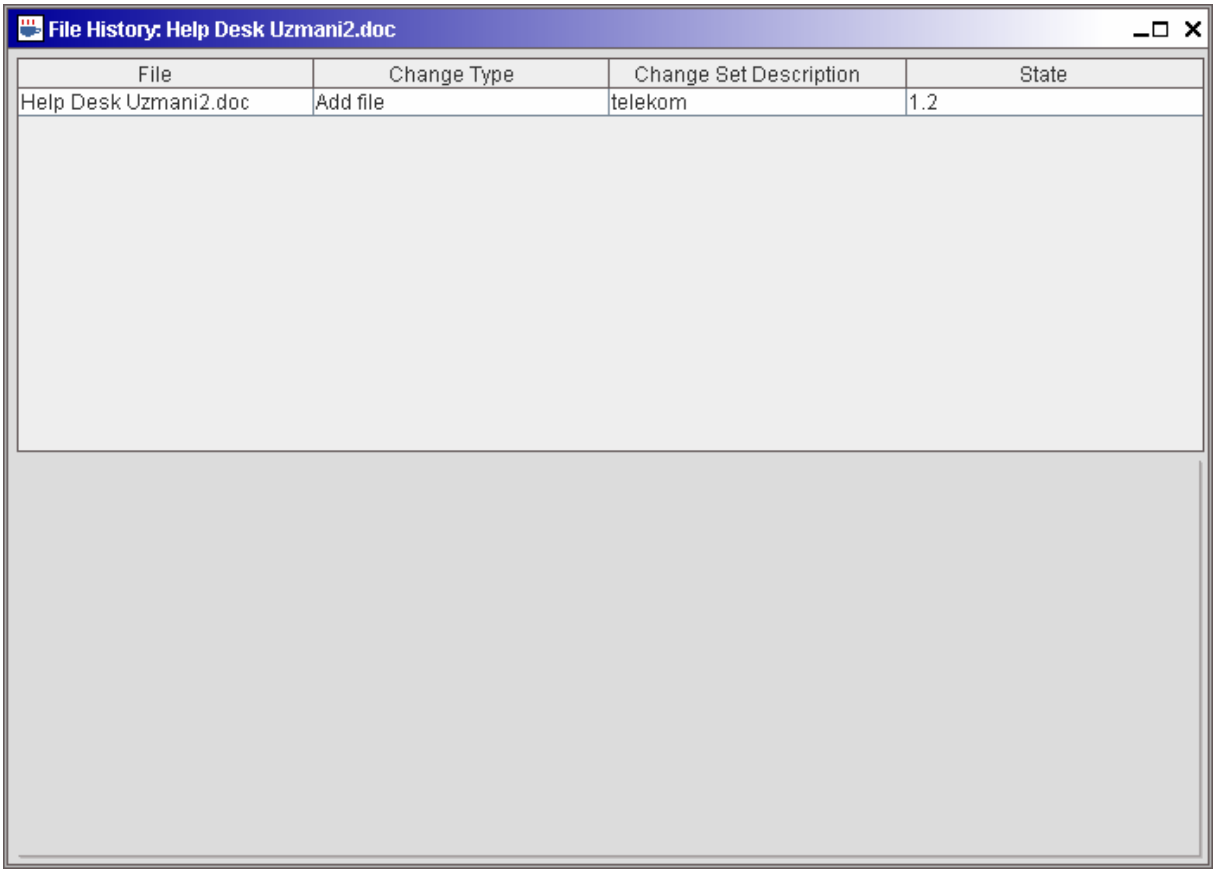

Edit Marker
Old marker name: TEST22_TANIMI
Old state: 1.1
New name:
OK
Cancel
TEST22_TANIMI
Select new state or remove selection:
1.1
Work area state
Marker: TEST22_TANIMI

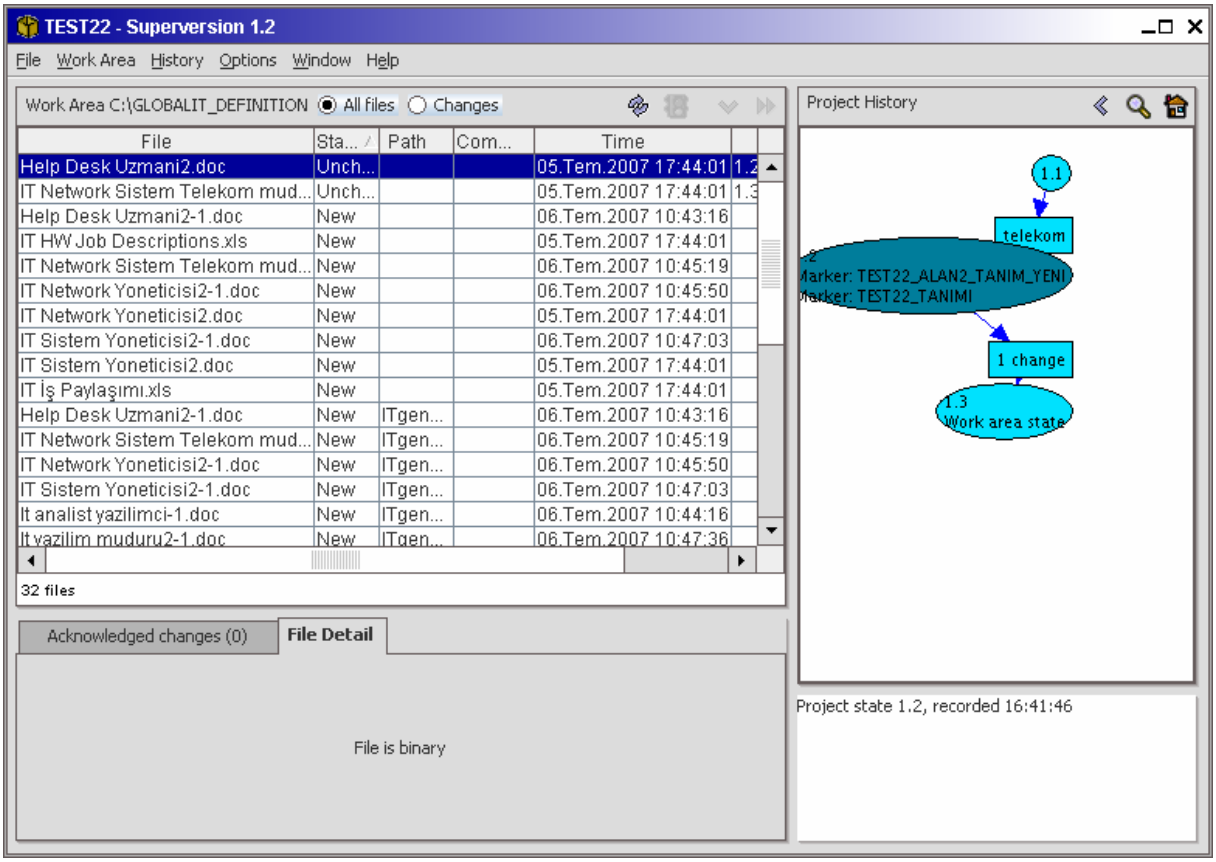

Work areas for TEST22
C:\GLOBALIT_DEFINITION
C:\BURCU_ODEV
Open
Unregister
Add work area...
Cancel
Help
A work area is a directory holding a copy of your project's files.
To import existing files, select "Add work area" and choose the main directory of your project.
To check out an existing project to a new place, select "Add work area" and choose an empty directory or type in the name for a new directory.

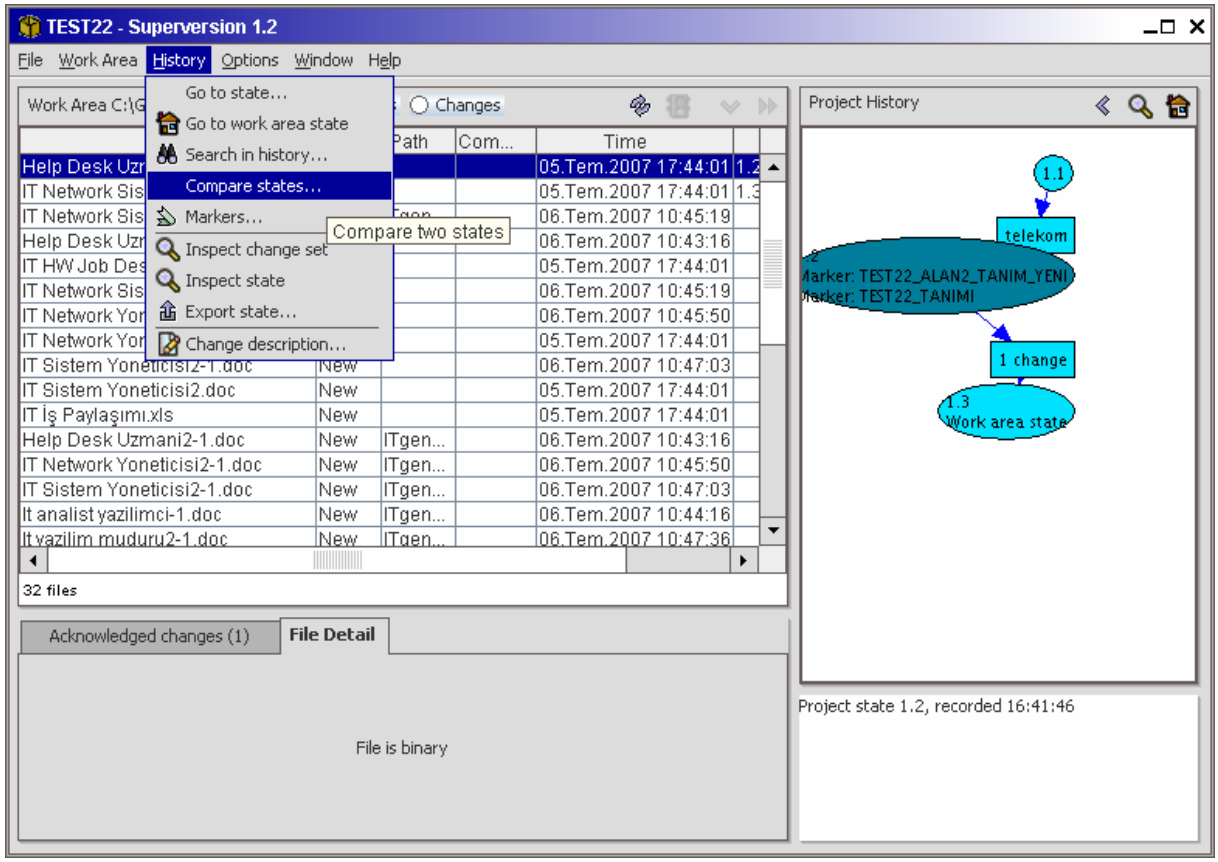

TEST22 - Superversion 1.2
File Work Area History Options Window Help
Work Area C:\BURCU_ODEV All files Changes
File Sta... Path Com... Time
20070928-1.jpg New 2007-... 28.Eyl.2007 14:59:20
20070928-2.doc New 2007-... 28.Eyl.2007 15:01:18
20071002.doc New 2007-... 02.Eki.2007 16:39:26
20071010.doc New 2007-... 10.Eki.2007 17:19:22
20071015.doc New 2007-... 15.Eki.2007 17:11:55
20071017.doc New 2007-... 17.Eki.2007 17:32:03
20071022.doc New 2007-... 22.Eki.2007 17:11:52
20071026.doc New 2007-... 26.Eki.2007 17:03:09
20071106.doc New 2007-... 06.Kas.2007 16:57:55
20071107.doc New 2007-... 07.Kas.2007 11:11:27
20071109.doc New 2007-... 09.Kas.2007 18:18:05
20071114.doc New 2007-... 14.Kas.2007 16:23:56
20070105-2.doc New 05.Oca.2007 14:19:02
20070105.doc New 05.Oca.2007 14:17:17
20070108-2.doc New 08.Oca.2007 13:31:49
20070108.doc New 08.Oca.2007 13:28:18
153 files
Acknowledged changes (1)
File Detail
File is binary
Project History
Work area state
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
Project state 1.1, recorded 16:14:11

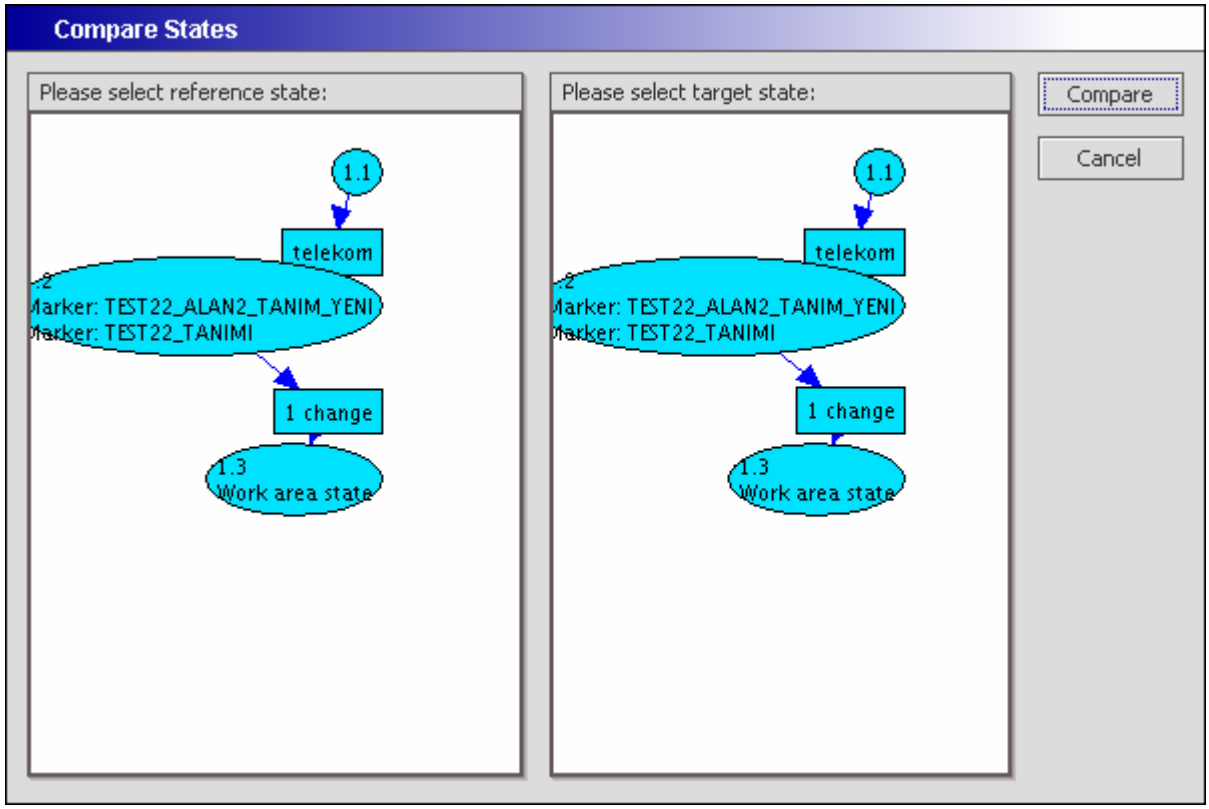

TEST22 - Superversion 1.2
File Work Area History Options Window Help
Work Area C:\GLOBALIT_DEFINITION All files Changes
File Sta... Path Com... Time
Help Desk Uzmani2.doc Unch... 05.Tem.2007 17:44:01
IT Network Sistem Telekom mud... Unch... 05.Tem.2007 17:44:01
Help Desk Uzmani2-1.doc New 06.Tem.2007 10:43:16
Help Desk Uzmani2-1.doc New ITgen... 06.Tem.2007 10:43:16
It analist yazilimci-1.doc New ITgen... 06.Tem.2007 10:44:16
It analist yazilimci-1.doc New 06.Tem.2007 10:44:16
It analist yazilimci.doc New 05.Tem.2007 17:41:39
IT HW Job Descriptions.xls New 05.Tem.2007 17:44:01
IT İş Paylaşımı.xls New 05.Tem.2007 17:44:01
IT Network Sistem Telekom mud... New 06.Tem.2007 10:45:19
IT Network Sistem Telekom mud... New ITgen... 06.Tem.2007 10:45:19
IT Network Yoneticisi2-1.doc New 06.Tem.2007 10:45:50
IT Network Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:45:50
IT Network Yoneticisi2.doc New 05.Tem.2007 17:44:01
IT Sistem Yoneticisi2-1.doc New 06.Tem.2007 10:47:03
IT Sistem Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:47:03
32 files
Acknowledged changes (0)
File Detail
File is binary
Project History
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state
Change set for state 1.3
Description:
1 change:
Add file IT Network Sistem Telekom mud2.doc

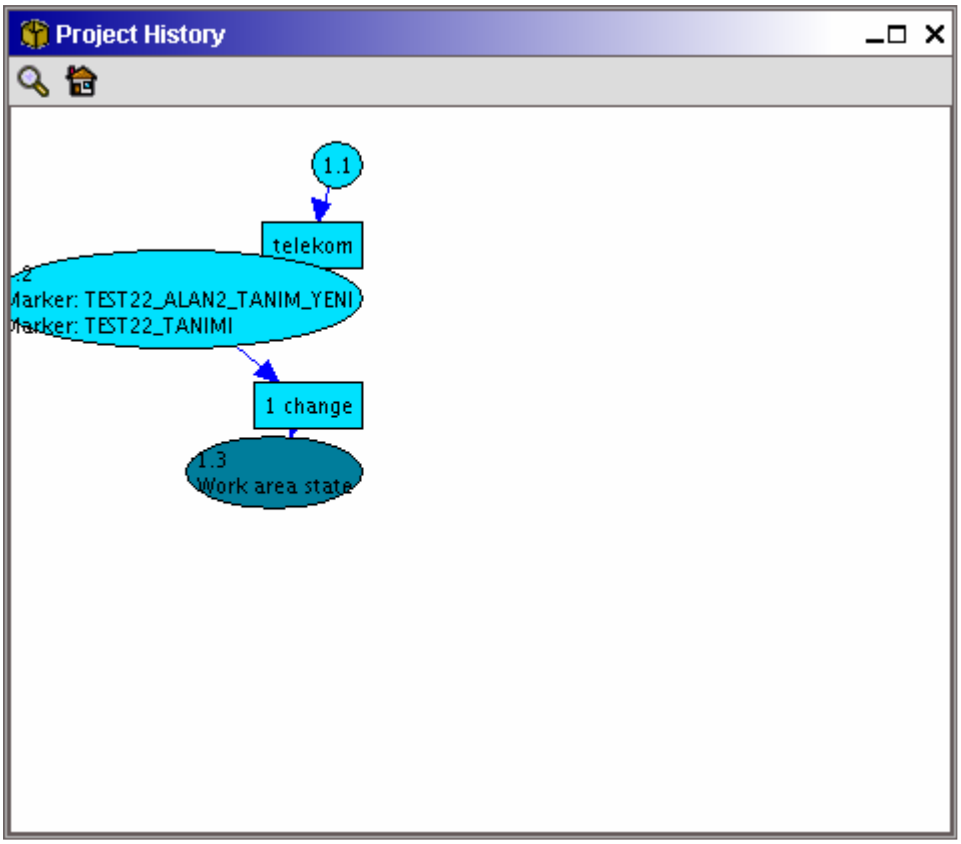

TEST22 - Superversion 1.2
File Work Area History Options Window Help
Work Area C:\GLOBALIT_DEFINITION All files Changes
File Sta... Path Com... Time
Help Desk Uzmani2.doc Unch 05.Tem.2007 17:44:01
IT Network Sistem Telek 05.Tem.2007 17:44:01
Help Desk Uzmani2-1.d 06.Tem.2007 10:43:16
IT HW Job Descriptions.x 05.Tem.2007 17:44:01
IT Network Sistem Telek 06.Tem.2007 10:45:19
IT Network Yoneticisi2-1 2007 10:45:50
IT Network Yoneticisi2.d 05.Tem.2007 17:44:01
IT Sistem Yoneticisi2-1.c 06.Tem.2007 10:47:03
IT Sistem Yoneticisi2.doc 05.Tem.2007 17:44:01
IT İş Paylaşımı.xls 05.Tem.2007 17:44:01
Help Desk Uzmani2-1.d 06.Tem.2007 10:43:16
IT Network Sistem Telekom mud... New ITgen... 06.Tem.2007 10:45:19
IT Network Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:45:50
IT Sistem Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:47:03
It analist yazilimci-1.doc New ITgen... 06.Tem.2007 10:44:16
It yazilim muduru2-1.doc New ITgen... 06.Tem.2007 10:47:36
View file
Show differences
Show file history
Acknowledg
Acknowledge and commit...
Ignore / Change file type...
Delete locally
Reconstruct
Show history of selected file
32 files
Acknowledged changes (0)
File Detail
File is binary
Project History
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state
Change set for state 1.3
Description:
1 change:
Add file IT Network Sistem Telekom mud2.doc


File History: Help Desk Uzmani2.doc
File
Change Type
Change Set Description
State
Help Desk Uzmani2.doc
Add file
telekom
1.2


TEST22 - Superversion 1.2
File Work Area History Options Window Help
Work Area C:\GLOBALIT_DEFINITION All files Changes
File Sta... Path Com... Time
Help Desk Uzmani2.doc Unch... 05.Tem.2007 17:44:01
IT Network Sistem Telekom mud... Unch... 05.Tem.2007 17:44:01
Help Desk Uzmani2-1.doc New 06.Tem.2007 10:43:16
IT HW Job Descriptions.xls New 05.Tem.2007 17:44:01
IT Network Sistem Telekom mud... New 06.Tem.2007 10:45:19
IT Network Yoneticisi2-1.doc New 06.Tem.2007 10:45:50
IT Network Yoneticisi2.doc New 05.Tem.2007 17:44:01
IT Sistem Yoneticisi2-1.doc New 06.Tem.2007 10:47:03
IT Sistem Yoneticisi2.doc New 05.Tem.2007 17:44:01
IT İş Paylaşımı.xls New 05.Tem.2007 17:44:01
Help Desk Uzmani2-1.doc New ITgen... 06.Tem.2007 10:43:16
IT Network Sistem Telekom mud... New ITgen... 06.Tem.2007 10:45:19
IT Network Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:45:50
IT Sistem Yoneticisi2-1.doc New ITgen... 06.Tem.2007 10:47:03
It analist yazilimci-1.doc New ITgen... 06.Tem.2007 10:44:16
It yazilim muduru2-1.doc New ITgen... 06.Tem.2007 10:47:36
32 files
Acknowledged changes (0)
File Detail
File is binary
Project History
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state
Project state 1.2, recorded 16:41:46


TEST22 - Superversion 1.2
File
Work Area
History
Options
Window
Help
Go to state...
Go to work area state
Search in history...
Compare states...
Markers...
Inspect change set
Inspect state
Export state...
Change description...
Compare two states
Changes
Project History
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state
32 files
Acknowledged changes (1)
File Detail
File is binary
Project state 1.2, recorded 16:41:46


Compare States
Please select reference state:
Please select target state:
Compare
Cancel
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state


Project History
1.1
telekom
Marker: TEST22_ALAN2_TANIM_YENI
Marker: TEST22_TANIMI
1 change
1.3
Work area state

EK 2 : Uygulama Kaynak Kod Örnekleri

```
// Random Access File Read/Write Procedure
// Source File Name:
/home/superversion/src/com/nothome/delta/RandomAccessFile2.java

package com.nothome.delta;

import java.io.*;

// Referenced classes of package com.nothome.delta:
//            IRandomAccess

public class RandomAccessFile2 extends RandomAccessFile
    implements IRandomAccess
{

    public RandomAccessFile2(String name, String mode)
        throws FileNotFoundException
    {
        super(name, mode);
    }

    public RandomAccessFile2(File file, String mode)
        throws FileNotFoundException
    {
        super(file, mode);
    }
}
```

```
// Calculate Delta Diffence Data Procudure
// Decompiler options: packimports(3)
// Source File Name:
/home//superversion/src/com/nothome/delta/Delta.java

package com.nothome.delta;

import java.io.*;

// Referenced classes of package com.nothome.delta:
//            RandomAccessFile2, Checksum, TooShortException,
DebugDiffWriter,
//            GDiffWriter, DiffWriter, IRandomAccess

public class Delta
{

    public Delta()
    {
    }

    public void computeDelta(File sourceFile, File targetFile,
DiffWriter output)
        throws IOException, Exception
    {
        BufferedInputStream targetStream = new
BufferedInputStream(new FileInputStream(targetFile));
        int targetLength = (int)targetFile.length();
        RandomAccessFile2 source = new
RandomAccessFile2(sourceFile, "r");
        try
        {
            computeDelta(((IRandomAccess) (source)), ((InputStream)
(targetStream)), targetLength, output);
        }
        finally
        {
            source.close();
            targetStream.close();
        }
    }

    public PushbackInputStream computeDelta(IRandomAccess source,
InputStream targetStream, int targetLength, DiffWriter output)
        throws TooShortException, IOException
    {
        int targetidx = 0;
        Checksum checksum = new Checksum();
        checksum.generateChecksums(source);
        PushbackInputStream target = new
PushbackInputStream(targetStream, 1024);
        boolean done = false;
        byte buf[] = new byte[16];
        long hashf = 0L;
        byte b[] = new byte[1];
        byte sourcebyte[] = new byte[16];
```

```
        if(targetLength - targetidx <= 16)
        {
            target.close();
            output.close();
            throw new TooShortException("Unable to compute delta,
input file is too short");
        }
        target.read(buf, 0, 16);
        targetidx += 16;
        hashf = Checksum.queryChecksum(buf, 16);
        long alternativehashf = hashf;
        boolean sourceOutofBytes = false;
        do
        {
            if(done)
                break;
            int index = checksum.findChecksumIndex(hashf);
            if(index != -1)
            {
                boolean match = true;
                int offset = index * 16;
                int length = 15;
                source.seek(offset);
                if(!sourceOutofBytes && source.read(sourcebyte, 0,
16) != -1)
                {
                    for(int ix = 0; ix < 16; ix++)
                        if(sourcebyte[ix] != buf[ix])
                            match = false;

                } else
                {
                    sourceOutofBytes = true;
                }
                if(match & (!sourceOutofBytes))
                {
                    boolean ok = true;
                    byte sourceBuff[] = new byte[1024];
                    byte targetBuff[] = new byte[1024];
                    int source_idx = 0;
                    int target_idx = 0;
                    do
                    {
                        source_idx = source.read(sourceBuff, 0,
1024);
                        if(source_idx == -1)
                        {
                            sourceOutofBytes = true;
                            b[0] = sourcebyte[15];
                            break;
                        }
                        target_idx = target.read(targetBuff, 0,
source_idx);
                        if(target_idx == -1)
                            break;
```

```
                        int read_idx = Math.min(source_idx,
target_idx);
                        int i = 0;
                        do
                        {
                            targetidx++;
                            length++;
                            ok = sourceBuff[i] == targetBuff[i];
                            i++;
                            if(!ok)
                            {
                                b[0] = targetBuff[i - 1];
                                if(target_idx != -1)
                                    target.unread(targetBuff, i,
target_idx - i);
                            }
                        } while(i < read_idx && ok);
                        b[0] = targetBuff[i - 1];
                    } while(ok && targetLength - targetidx > 0);
                    output.addCopy(offset, length);
                    if(targetLength - targetidx <= 15)
                    {
                        buf[0] = b[0];
                        int remaining = targetLength - targetidx;
                        target.read(buf, 1, remaining);
                        targetidx += remaining;
                        for(int ix = 0; ix < remaining + 1; ix++)
                            output.addData(buf[ix]);

                        done = true;
                    } else
                    {
                        buf[0] = b[0];
                        target.read(buf, 1, 15);
                        targetidx += 15;
                        alternativehashf = hashf =
Checksum.queryChecksum(buf, 16);
                    }
                    continue;
                }
            }
            if(targetLength - targetidx > 0)
            {
                target.read(b, 0, 1);
                targetidx++;
                output.addData(buf[0]);
                alternativehashf =
Checksum.incrementChecksum(alternativehashf, buf[0], b[0]);
                for(int j = 0; j < 15; j++)
                    buf[j] = buf[j + 1];

                buf[15] = b[0];
                hashf = Checksum.queryChecksum(buf, 16);
            } else
            {
                for(int ix = 0; ix < 16; ix++)
```

```
                    output.addData(buf[ix]);

                done = true;
            }
        } while(true);
        output.close();
        return target;
    }

    public static void main(String argv[])
    {
        Delta delta = new Delta();
        if(argv.length != 3)
        {
            System.err.println("usage Delta [-d] source target
[output]");
            System.err.println("either -d or an output filename
must be specified.");
            System.err.println("aborting..");
            return;
        }
        try
        {
            DiffWriter output = null;
            File sourceFile = null;
            File targetFile = null;
            if(argv[0].equals("-d"))
            {
                sourceFile = new File(argv[1]);
                targetFile = new File(argv[2]);
                output = new DebugDiffWriter();
            } else
            {
                sourceFile = new File(argv[0]);
                targetFile = new File(argv[1]);
                output = new GDiffWriter(new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(new File(argv[2])))));
            }
            if(sourceFile.length() > 0x7fffffffL ||
targetFile.length() > 0x7fffffffL)
            {
                System.err.println("source or target is too large,
max length is 2147483647");
                System.err.println("aborting..");
                return;
            }
            delta.computeDelta(sourceFile, targetFile, output);
            output.flush();
            output.close();
        }
        catch(Exception e)
        {
            System.err.println("error while generating delta: " +
e);
        }
    }
```

```
    public static final int S = 16;
    public static final boolean debug = false;
    public static final int buff_size = 1024;
}
```

```
// Detect Files Diffrence Procedure
// Decompiler options: packimports(3)
// Source File Name:
/home/superversion/src/com/nothome/delta/DeltaTest.java

package com.nothome.delta;

import drjava.superversion.tools.FileUtil;
import java.io.*;

// Referenced classes of package com.nothome.delta:
//            Delta, GDiffWriter, RandomAccessFile2, GDiffPatcher

public class DeltaTest
{

    public DeltaTest()
    {
    }

    public static void main(String args[])
        throws Exception
    {
        Delta delta = new Delta();
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        DiffWriter diffWriter = new GDiffWriter(new
DataOutputStream(out));
        File file1 = new File("stuff/realocc2");
        byte data1[] = FileUtil.loadFile(file1);
        File file2 = new File("stuff/realocc1");
        byte data2[] = FileUtil.loadFile(file2);
        delta.computeDelta(file1, file2, diffWriter);
        byte patch[] = out.toByteArray();
        System.out.println("xdelta: => " + patch.length);
        out = new ByteArrayOutputStream();
        GDiffPatcher.patch(new RandomAccessFile2(file1, "r"), out,
new ByteArrayInputStream(patch));
        System.out.println("length: " + data2.length + " / " +
out.size());
        byte outBytes[] = out.toByteArray();
        for(int i = 0; i < data2.length; i++)
            if(data2[i] != outBytes[i])
                System.out.println("mismatch #" + i + ": " +
data2[i] + "/" + outBytes[i]);

    }
}
```

```
// Merging Entity Procedure
// Decompiler options: packimports(3)
// Source File Name:
/home/superversion/src/drjava/superversion/tools/Merger.java

package drjava.superversion.tools;

import java.util.List;

// Referenced classes of package drjava.superversion.tools:
//            MergeResult, ExplodedLine, BlockDiffer

public class Merger
{

    public Merger()
    {
    }

    public static MergeResult merge(List original, List a, List b)
    {
        List explodedA = BlockDiffer.explode(original, a);
        List explodedB = BlockDiffer.explode(original, b);
        MergeResult result = new MergeResult();
        int iA = 0;
        int iB = 0;
        do
        {
            if(iA >= explodedA.size() && iB >= explodedB.size())
                break;
            if(iA >= explodedA.size())
            {

result.addLine(getLine((ExplodedLine)explodedB.get(iB)));
                iB++;
            } else
            if(iB >= explodedB.size())
            {

result.addLine(getLine((ExplodedLine)explodedA.get(iA)));
                iA++;
            } else
            {
                ExplodedLine eA = (ExplodedLine)explodedA.get(iA);
                ExplodedLine eB = (ExplodedLine)explodedB.get(iB);
                if(eA.getType() == 3)
                {
                    if(eB.getType() == 3)
                    {
                        if(eA.getRight().equals(eB.getRight()))
                        {
                            result.addLine(getLine(eA));
                            iA++;
                            iB++;
                        } else
                        {
```

```
                                result.addLine(getLine(eA),
getLine(eB));
                                iA++;
                                iB++;
                            }
                        } else
                        {
                            result.addLine(getLine(eA));
                            iA++;
                        }
                    } else
                    if(eA.getType() == 2)
                    {
                        if(eB.getType() == 3)
                        {
                            result.addLine(getLine(eB));
                            iB++;
                        } else
                        if(eB.getType() == 1)
                        {
                            result.addLine(null, getLine(eB));
                            iA++;
                            iB++;
                        } else
                        {
                            iA++;
                            iB++;
                        }
                    } else
                    if(eA.getType() == 0)
                    {
                        if(eB.getType() == 2)
                        {
                            iA++;
                            iB++;
                        } else
                        {
                            result.addLine(getLine(eB));
                            iB++;
                            if(eB.getType() != 3)
                                iA++;
                        }
                    } else
                    if(eB.getType() == 3)
                    {
                        result.addLine(getLine(eB));
                        iB++;
                    } else
                    if(eB.getType() == 2)
                    {
                        result.addLine(getLine(eA), null);
                        iA++;
                        iB++;
                    } else
                    if(eB.getType() == 0)
                    {
```

```
                    result.addLine(getLine(eA));
                    iA++;
                    iB++;
                } else
                if(eA.getRight().equals(eB.getRight()))
                {
                    result.addLine(getLine(eA));
                    iA++;
                    iB++;
                } else
                {
                    result.addLine(getLine(eA), getLine(eB));
                    iA++;
                    iB++;
                }
            }
        } while(true);
        return result;
    }

    private static String getLine(ExplodedLine explodedLine)
    {
        return explodedLine.getType() != 2 ?
explodedLine.getRight() : explodedLine.getLeft();
    }
}
```

```
// Scanning of Files from directory Procedure
// Decompiler options: packimports(3)
// Source File Name:
/home/superversion/src/drjava/superversion/tools/Scanner.java

package drjava.superversion.tools;

import drjava.superversion.DownloadFailedException;
import drjava.superversion.api.*;
import drjava.superversion.filesystem.Directory;
import drjava.superversion.filesystem.File;
import java.io.PrintStream;
import java.util.*;

// Referenced classes of package drjava.superversion.tools:
//            PassAllFilter, PathFilter, FingerprintCache

public class Scanner
{

    public Scanner()
    {
    }

    public static List findChanges(Directory a, Directory b,
PathFilter filter, FingerprintCache bCache)
        throws DownloadFailedException
    {
        ArrayList list = new ArrayList();
        addChanges(list, a, b, "", 0, filter, bCache);
        return list;
    }

    private static void addChanges(List list, Directory a,
Directory b, String path, int depth, PathFilter filter,
FingerprintCache bCache)
        throws DownloadFailedException
    {
        if(depth >= 32)
        {
            System.err.println("Warning: Max scan depth reached: "
+ path);
            return;
        }
        final Map aFiles = new TreeMap();
        Set bFiles = new TreeSet();
        if(a != null)
        {
            File _file;
            for(Iterator i = a.getFiles().iterator(); i.hasNext();
aFiles.put(_file.getName(), _file))
                _file = (File)i.next();

        }
        if(b != null)
        {
```

```
            File _file;
            for(Iterator i = b.getFiles().iterator(); i.hasNext();
bFiles.add(_file.getName()))
                _file = (File)i.next();

        }
        if(b != null)
        {
            Iterator i = b.getFiles().iterator();
            do
            {
                if(!i.hasNext())
                    break;
                final File _file = (File)i.next();
                if(filter.match(_file.getName()))
                {
                    final File aFile =
(File)aFiles.get(_file.getName());
                    if(aFile == null)
                    {
                        list.add(new NewFileChange(path +
_file.getName()) {

                            public Data getContents()
                            {
                                return _file.getContents();
                            }

                        }
);
                    } else
                    {
                        String filePath = path + _file.getName();
                        if(aFile.getContents().getFingerprint() !=
bCache.getFingerprint(filePath, _file))
                            list.add(new ModifyFileChange(filePath)
{

                                public Data getPreviousContents()
                                {
                                    return aFile.getContents();
                                }

                                public Data getContents()
                                {
                                    return _file.getContents();
                                }

                            }
);
                    }
                }
            } while(true);
        }
        Iterator i = aFiles.keySet().iterator();
        do        {
```

```
            if(!i.hasNext())
                break;
            final String _file = (String)i.next();
            if(!bFiles.contains(_file))
                list.add(new RemoveFileChange(path + _file) {

                    public Data getPreviousContents()
                    {
                        return
((File)aFiles.get(_file)).getContents();
                    }

                }
);
        } while(true);
        Map aDirs = collectDirNames(a);
        Map bDirs = collectDirNames(b);
        if(b != null)
        {
            Iterator i = b.getDirectories().iterator();
            do
            {
                if(!i.hasNext())
                    break;
                Directory dir = (Directory)i.next();
                if(!autoIgnoreDirs.contains(dir.getName()) &&
filter.match(dir.getName()))
                {
                    Directory aDir =
(Directory)aDirs.get(dir.getName());
                    addChanges(list, aDir, dir, path +
dir.getName() + "/", depth + 1, filter.enterSubdir(dir.getName()),
bCache);
                }
            } while(true);
        }
        if(a != null)
        {
            Iterator i = a.getDirectories().iterator();
            do
            {
                if(!i.hasNext())
                    break;
                Directory dir = (Directory)i.next();
                Directory bDir =
(Directory)bDirs.get(dir.getName());
                if(bDir == null &&
!autoIgnoreDirs.contains(dir.getName()) &&
filter.match(dir.getName()))
                    addChanges(list, dir, bDir, path +
dir.getName() + "/", depth + 1, filter.enterSubdir(dir.getName()),
bCache);
            } while(true);
        }
    }
    private static Map collectDirNames(Directory d)
```

```
    {
        Map dirs = new TreeMap();
        if(d != null)
        {
            Directory dir;
            for(Iterator i = d.getDirectories().iterator();
i.hasNext(); dirs.put(dir.getName(), dir))
                dir = (Directory)i.next();

        }
        return dirs;
    }

    public static List collectFilePaths(Directory dir)
    {
        return collectFilePaths(dir, ((PathFilter) (new
PassAllFilter())));
    }

    public static List collectFilePaths(Directory dir, PathFilter
filter)
    {
        List list = new ArrayList();
        collectFilePaths(list, dir, "", filter);
        return list;
    }

    static void collectFilePaths(List list, Directory dir, String
path, PathFilter filter)
    {
        Iterator i = dir.getFiles().iterator();
        do
        {
            if(!i.hasNext())
                break;
            File f = (File)i.next();
            if(filter.match(f.getName()))
                list.add(path + f.getName());
        } while(true);
        Iterator i = dir.getDirectories().iterator();
        do
        {
            if(!i.hasNext())
                break;
            Directory d = (Directory)i.next();
            if(!autoIgnoreDirs.contains(d.getName()))
                collectFilePaths(list, d, path + d.getName() + "/",
filter.enterSubdir(d.getName()));
        } while(true);
    }
    static final List autoIgnoreDirs = Arrays.asList(new String[] {
        "CVS"
    });
    static final int MAXDEPTH = 32;

}
```