KISITLAR TEORİSİ YAKLAŞIMININ YAZILIM GELİŞTİRME SÜRECİNE ETKİSİ

Görkem EREL
Yüksek Lisans Tezi
İşletme Anabilim Dalı
Danışman: Doç. Dr. Murat Selim SELVİ

2019

T.C.
TEKİRDAĞ NAMIK KEMAL ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
İŞLETME ANABİLİM DALI
YÜKSEK LİSANS TEZİ

KISITLAR TEORİSİ YAKLAŞIMININ YAZILIM GELİŞTİRME SÜRECİNE ETKİSİ

GÖRKEM EREL

İŞLETME ANABİLİMDALI

DANIŞMAN: Doç. Dr. Murat Selim SELVİ

TEKİRDAĞ – 2019

BİLİMSEL ETİK BİLDİRİMİ

Hazırladığım Yüksek Lisans Tezinin bütün aşamalarında bilimsel etiğe ve akademik kurallara riayet ettiğimi, çalışmada doğrudan veya dolaylı olarak kullandığım her alıntıya kaynak gösterdiğimi ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu, yazımda enstitü yazım kılavuzuna uygun davranıldığını taahhüt ederim.

09/01/2019

Görkem EREL

T.C.
NAMIK KEMAL ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
İŞLETME ANABİLİM DALI
YÜKSEK LİSANS TEZİ

Görkem EREL tarafından hazırlanan "Kısıtlar Teorisi Yaklaşımının Yazılım Geliştirme Sürecine Etkisi" konulu YÜKSEK LİSANS Tezinin Sınavı, Tekirdağ Namık Kemal Üniversitesi Lisansüstü Eğitim Öğretim Yönetmeliği uyarınca 26.12.2018 günü saat 10.30'da yapılmış olup, tezin KABUL /RED edilmesine OYBİRLİĞİ / OYÇOKLUĞU ile karar verilmiştir.

Jüri Başkanı: Doç.Dr. Murat Selim SELVİ	İmza:
Üye: Dr. Öğretim Üyesi: Murat TOPALOĞLU	İmza:
Üye: Dr. Öğretim Üyesi Seda Ş. GÜNGÖR	İmza:
Üye:	İmza:
Üye:	İmza:

Sosyal Bilimler Enstitüsü Yönetim Kurulu adına

...../...../20.....

Doç. Dr. Emrah İsmail ÇEVİK
Enstitü Müdürü

ÖZET

Kurum, Enstitü ABD	: Namık Kemal Üniversitesi, Sosyal Bilimler Enstitüsü, İşletme Ana Bilim Dalı
Tez Başlığı	: Kısıtlar Teorisi Yaklaşımının Yazılım Geliştirme Sürecine Etkisi
Tez Yazarı	: Görkem EREL
Tez Danışmanı	: Doç. Dr. Murat Selim SELVİ
Tez Türü, Yılı	: Yüksek Lisans Tezi, 2018
Sayfa Sayısı	: 78

Kısıtlar teorisi, bir sistemin sorunlarını tespit ederek, kendine özgü süreçlerle bu problemlerin giderilmesini amaçlayan bir yaklaşımdır. Organizasyonun en zayıf halkası kadar güçlü olduğunu savunan bu düşünce sistemi; mevcut gerçeklik ağaçlarıyla kısıtları belirleyip, bu kısıtların giderilmesi için Buharlaşan Bulut tekniğiyle çareler arar. Kısıtlar Gelecek Gerçeklik Ağacıyla giderilmekte ve istenilen durum tarif edilmektedir. Bulunan çarelerin nasıl uygulanacağı da geçiş ağaçlarıyla gösterilir. Yapılan çalışmanın amacı yazılım geliştirilirken karşılaşılan sorunların kısıtlar teorisi yöntemiyle tespit edilmesi, düzeltilmesi ve ortadan kaldırılmasının yol haritasını çizmektir. Bu tezin önemi teknik bir konu olan yazılım geliştirme süreçlerinde, sosyal bilimlerin araştırma konusu olan kısıtlar teorisini kullanarak her iki alanda çalışmalara köprü kurmaktır. Bu tez çalışmasında mevcut literatür taranarak ikincil kaynaklardan yararlanılmıştır. Bu çerçevede ilgili kitap, makale ve tezler üzerinde "Doküman Analizi" yapılmıştır. Ortaya çıkan sonuçların farklı sektörlerdeki iş kollarında görevli ve ilgililere önemli ipuçları sağlayacağı beklenmektedir. Kısıtlar teorisinin amacı en zayıf halkayı bularak güçlendirmektir. Yazılımcı bir firma için satış sonrası ortaya çıkan yazılıma ilişkin sorunlar nedeniyle müşteri zihninde kötü imaj oluşabilir. Bu ise yazılım firmasına emek ve zaman kaybına neden olur. Diğer taraftan alıcı durumundaki müşteriler için de zaman ve para kaybı söz konusudur. Kısıtlar teorisi önleyici tedbirler alarak bu sorunları yazılımın müşteriye tesliminden önce tespit etmekte ve böylece firmanın güvenilirliğini artırmaktadır. Ayrıca kısıtlar teorisi sayesinde firmanın rekabet gücü büyük ölçüde artmakta; müşterinin zaman ve para kaybı önlenmektedir.

Anahtar Kelimeler: Kısıtlar Teorisi, Zayıf Halka, Yazılım Geliştirme, Yazılım Yaşam Döngüsü, Tekirdağ

ABSTRACT

Institution, Institute, Department	: Tekirdağ Namık Kemal University, Institute of Social Sciences, Department of Business Management
Title	: The effect of constraints theory approach on the process of software development
Author	: Görkem EREL
Adviser	: Assoc. Prof. Murat Selim SELVİ
Type of Thesis, Year	: MA Thesis, 2018
Total Number of pages	: 78

The purpose of the theory of constraints is to detech and strengthen the weakest link. Due to the problems that arise after delivering the software to the customer in the software industry, the reliability of software companies is damaged. Therefore, the company's labor and time is wasted and the customer's resources are also inefficient. Due to the constraints theory, these problems are determined prior to the delivery of the software to the customer and by increasing the reliability and competitiveness of the software company to ensure that the resources of the customer are used as efficiently as possible. The theory of Constraints is an approachment which aims to solve with it’s unique process by identifying the problems of a system. This system of thought which suggest that every organization is as strong as the weakest ; The symptoms related to current reality trees seek solutions with the Evaporating Cloud technique to overcome these constraints. Constraints are solved with the Future Reality Tree and the desired situation is described. How to apply the solutions is also shown by the transition trees. The aim of the study is to determine the road map of the problems encountered in software development by the method of constraints theory, correction and elimination. The importance of this thesis is to bridge the studies in both fields by using the theory of constraints, which is the subject of research in the social sciences, in the technical development process. In this thesis, current literature was searched and secondary sources were used. within this framework, the Document Analysis was made on the related books, articles and theses. It is expected that the results will provice important clues to people who has duties and responsiblities in different sectors. The purpose of the theory of constraints is to detech and strengthen the weakest link. Due to the problems that arise after delivering the software to the customer in the software industry, the reliability of software companies is damaged. Therefore, the company's labor and time is wasted and the customer's resources are also inefficient. Due to the constraints theory, these problems are determined prior to the delivery of the software to the customer and by increasing the reliability and competitiveness of the software company to ensure that the resources of the customer are used as efficiently as possible.

Key words: Theory of Constraints, Weakest Link, Software Development, Lifecycle

ÖNSÖZ

Kısıtlar teorisi özel ve kamu sektöründe kullanıldığında oldukça yararlar sağlayan bir yaklaşımdır. Bu çalışmada bu yaklaşımın yazılım geliştirme sürecinde ne gibi etkilerinin olabileceği araştırılmıştır. Konunun ilgili ve görevli kişilerin dikkatini çekebileceği ve ortaya çıkan sonuçların ilgiyle karşılanacağı beklenmektedir.

Yüksek lisans öğrenimim boyunca akademik destek aldığım değerli hocalarıma teşekkür ederim. Tez çalışması sürecinde beni hep destekleyen sevgili eşim Kübra EREL'e, her türlü desteğini esirgemeyen Tez Danışman Doç. Dr. Murat Selim SELVİ'ye ayrıca teşekkür ederim.

Görkem EREL

İÇİNDEKİLER

ÖZET .....
ABSTRACT .....
ÖNSÖZ i
İÇİNDEKİLER ii
ŞEKİLLER TABLOSU iv
GİRİŞ 1

BİRİNCİ BÖLÜM

KISITLAR TEORİSİ

1.1. Kısıt Kavramı ve Tanımı 4
1.2. Kısıtlar Teorisinin Temel Özellikleri 6
1.3. Kısıtlar Teorisindeki Beş Ana İlke 7
1.4. Kısıt Çeşitleri 8
1.4.1. Malzeme Kısıtları 9
1.4.2. Yönetsel Kısıtlar 9
1.4.3. Kapasite Kısıtları 10
1.4.4. Lojistik Kısıtları 11
1.4.5. Pazar Kısıtları 12
1.4.6. Davranışsal Kısıtlar 13
1.4.7. Zorunlu Durumlar 14
1.4.8. Maliyet Yaklaşımı 14
1.5. Kısıtlar Teorisindeki Düşünce Süreçleri 15
1.5.1. Bugünkü Gerçeklik Ağacı 16
1.5.2. Buharlaşan Bulut 17
1.5.3. Gelecekteki Gerçeklik Ağacı 19
1.5.5. Geçiş Ağacı 22

İKİNCİ BÖLÜM

YAZILIM GELİŞTİRME SÜREÇLERİ

2.1. Yazılım Geliştirme Süreçlerinin Tanımı ve Özellikleri 25
2.2. Yazılım Geliştirme Yaşam Döngüsü 30
2.3. Yazılım Yaşam Döngüsü Temel Adımları 31
2.3.1. Planlama 32
2.3.2. Analiz 33
2.3.3. Tasarım 33
2.3.4. Gerçekleştirim 34
2.3.5. Bakım 34
2.4. Yazılım Geliştirme Süreci Modelleri 35
2.4.1. Gelişi Güzel (Disiplinsiz) Model 36
2.4.2. Şelale (Çağlayan) Modeli 36
2.4.3. Barok Modeli 38
2.4.4. V Modeli 39
2.4.5. Helezonik (Spiral) Model 40
2.4.6. Artırımsal Model 41
2.4.7. Evrimsel Model 42
2.5. Yazılım Geliştirme sürecinde karşılaşılan sorunlar 44

ÜÇÜNCÜ BÖLÜM

KISITLAR TEORİSİ YAKLAŞIMININ YAZILIM GELİŞTİRME SÜRECİNDE KULLANIMI

3.1. Mevcut Gerçeklik Ağacı' nın Kullanımı 47
3.2. Buharlaşan Bulut'un kullanılması 50
3.3. Gelecekteki Gerçeklik Ağacının Kullanılması 52
3.4. Ön Gereksinimler Ağacının Kullanılması 55
3.5. Geçiş Ağacının Kullanılması 58
SONUÇ VE ÖNERİLER 61
KAYNAKÇA 63

ŞEKİLLER LİSTESİ

Şekil 1: Bugünkü Gerçeklik Ağacı 17
Şekil 2: Çelişki Çözüm Diyagramı 18
Şekil 3: Gelecekteki Gerçeklik Ağacı 20
Şekil 4: Ön Koşul Ağacı 21
Şekil 5: Geçiş Ağacı 23
Şekil 6: Bütünleşik Düşünce Süreci Olarak 5 Mantık Aracı Arasındaki İlişki 24
Şekil 7: Yazılımın Unsurları 25
Şekil 8: Yazılım Hata Oranları Tablosu 27
Şekil 9: Yazılım Geliştirme Süreci Akış Şeması 28
Şekil 10: Yazılım Geliştirme Sistem Harcamaları Grafiği 29
Şekil 11: Yazılım Geliştirme Yaşam Döngüsü. 31
Şekil 12: Yazılım Geliştirme Süreçleri 32
Şekil 13: Şelale (Çağlayan) Modeli 37
Şekil 14: Barok Modeli 38
Şekil 15: V Modeli 39
Şekil 16: Helezonik (Spiral) Model 40
Şekil 17: Artırımsal Model 41
Şekil 18: Evrimsel Model 43
Şekil 19: Boyutlarına Göre Yazılım Sınıflandırmaları 43
Şekil 20: İşlevlerine Göre Yazılım Sınıflandırmaları 44
Şekil 21: Yazılım Geliştirme Sorun Döngüsü 45
Şekil 22: Mevcut Gerçeklik Ağacı Uygulaması 1 48
Şekil 23: Mevcut Gerçeklik Ağacı Uygulaması 2 49
Şekil 24: Buharlaşan Bulut Uygulaması 1 51
Şekil 25: Buharlaşan Bulut Uygulaması 2 52
Şekil 26: Gelecekteki Gerçeklik Ağacı Uygulaması 1 54
Şekil 27: Gelecekteki Gerçeklik Ağacı Uygulaması 2 55
Şekil 28: Ön Gereksinimler Ağacı Uygulaması 1 56
Şekil 29: Ön gereksinimler Ağacı Uygulaması 2 57
Şekil 30: Geçiş Ağacı Uygulaması 1 59
Şekil 31: Geçiş Ağacı Uygulaması 2 60

GİRİŞ

Şirketlerde oluşan sorunların, üretimi büyük ölçüde etkilemesi sebebiyle bu sorunlara çözüm bulmak için kısıtlar teorisinin geliştirilmesine ihtiyaç duyulmuştur. Şirketler, işletmeler veya küçük ticari oluşumlar, bir sistem olarak adlandırılmıştır. Bu sistemlerin verimini engelleyen her şey bir kısıt olarak düşünülmektedir ve her işletmede en az bir kısıt olduğu ileri sürülmektedir. Kısıtlar zayıf halkalar olarak tanımlanmaktadırlar. İşletmeler bir bütün olarak ele alındığında bu işletmenin içinde bulunan kısıtın tüm birimlerine tesir edebildiği görülmektedir ve işletme zayıf halkası kadar güçlü olabilmektedir. Bu yüzden işletme içindeki kısıt veya kısıtlar tespit edilerek etkili bir şekilde yönetilmelidir (Kaygusuz, 2005, s. 133-153).

Yapılan araştırmalara göre yazılım sektöründe temel olarak iki problemin göze çarptığı gözlemlenmektedir. İlki planlanan yazılımın gerçekleştirim aşamasının uzun sürmesi ve ortaya çıkan yazılımın kalitesinin istenildiği gibi olmamasıdır. Diğeri ise yazılımın istenilen standartlara uygun olmasına rağmen kullanıcı tarafından sürdürülememesidir. Bu iki problemin oluşmasına yazılım projelerindeki Planlama, Analiz, Tasarım, Gerçekleştirim ve Bakım aşamalarında bulunan tespit edilememiş kısıtların neden olduğu savunulabilir (Bank, 2014, s. 28-31).

İçinde kısıt bulunan kaynak ile içinde kısıt bulunmayan kaynak arasında bir ilişki vardır. Kısıtı olmayan kaynağın verimli bir şekilde çalışması, kısıtı olan kaynağın hemen önünde birikime neden olacağı için hem kısıtı olan kaynaktan hem de kısıtı olmayan kaynaktan çıkan parçalar, bir yerde birleştirilecekse burada kısıtı olmayan kaynağın verimli kullanılmasında dolayı, o kaynağa ait parçalarda birikme oluşacaktır. Bu düzene göre kısıt oluşmayan birimin verimi de kısıtlı kaynak tarafından belirlenmektedir. Dolayısıyla her birimin en üst seviyede kullanılması sistemin de maksimum verimde çalışacağı anlamına gelmez (Goldratt ve Cox, 1984, s. 346).

Araştırmanın Amacı ve Önemi: Bu çalışmada, kısıtlar teorisi düşünce süreçlerinin, yazılım geliştirme süreçlerinde nasıl kullanılabileceği ve oluşan problemlerin bu yöntemle nasıl ortadan kaldırılabileceği üzerinde durulmuştur. Sektör fark etmeksizin oluşan bu sorunların birçok yöntemle ortadan kaldırılması mümkünken, kısıtlar teorisinin bu sorunları ortadan kaldırış biçimi vurgulanmaktadır. Her sektörde kullanılabilen kısıtlar teorisi düşünce süreçlerinin, yazılım sektöründeki uygulamalarına örnekler verilerek anlatılmıştır. İşletme verimliliğinin artırılması hususunda önemli olan bu teori sosyal bilimlerin gelişimine katkı sağlarken yazılım geliştirme sürecinde kullanılması ise teknik bilimlere de önemli ölçüde katkı sağlamaktadır. İşletmelerde oluşan sorunlar, işletmelerin verimlerini düşürmektedir. İşletmelerin verimlerini ve karlılık seviyelerini yüksek tutabilmeleri için bu sorunları ortadan kaldırmaları kaçınılmazdır. Küçük sorunlar göz ardı edildiklerinde büyük sorunlara dönüşerek işletmelerin ciddi sıkıntılar yaşamalarına neden olmaktadır. Bu sorunların tespiti ve büyümeden önüne geçilebilmesi işletmelerin yaşamları için büyük önem arz etmektedir.

İşletmelerde kullanılan yazılımın kalitesinin artması işletmelerin verimliliğinin de artacağı anlamına gelmektedir. Bu çalışmanın amacı kısıtlar teorisi yaklaşımının yazılım geliştirme sürecinde kullanılmasının daha yazılım geliştirilirken kısıtların ortadan kaldırılmasına ve işletmelerin kaliteli yazılıma kısa sürede ve en az maliyetle ulaşmasına katkı sağlayacağına dikkat çekmektir.

Bu çalışmada kısıtlar teorisi yaklaşımının yazılım sürecine dâhil edilmesiyle organizasyonların zamanlarını daha etkin kullanabileceklerine işaret edilmektedir. Yazılım geliştirme sürecinde kısıtlar teorisi yaklaşımının uygulanmasıyla ortaya çıkan sonuçların farklı sektörlerdeki iş kollarında görevli ve ilgililere kısıtlar konusunda önemli ipuçları sağlayacağı beklenmektedir.

Kısıtlar teorisi yaklaşımı ile ilgili yapılan daha önceki çalışmalarda çeşitli sektörlerdeki uygulamalarının mevcut olduğu görülmüştür. Örneğin Ünal, Tanış ve Küçüksavaş'ın (2006) kısıtlar teorisinin bir üretim işletmesine nasıl uygulanabileceğiyle ilgili çalışmalarında yine en zayıf halkadan bahsederek kısıtlara vurgu yapmışlardır. Yine Top ve Oktay (2010) problem çözme metodolojisi olarak

kısıtlar teorisini incelemiş; yine en zayıf halka olarak kısıtlar üzerinde durmuşlardır. Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsünden bahseden çalışmasında Seker (2015), yazılım geliştirme aşamalarının tanımlarını yapmıştır. Yazılım geliştirme sürecinin iyileştirilmesi üzerine başka bir çalışma yapan GÜL (2015) yazılımda karşılaşılan problemlerden bahsederek yazılım süreç iyileştirme modellerine vurgu yapmıştır. Akman ve Karakoç'un (2005) "Yazılım Geliştirme Sürecinde Kısıtlar Teorisinin Düşünce Süreçlerinin Kullanılması" adlı çalışmasında ise kısıtlar teorisi düşünce süreçlerinin yazılım geliştirme süreçlerine uygulanışından bahsedilmektedir.

Bu tez çalışmasında mevcut literatür taranarak ikincil kaynaklardan yararlanılmıştır. Bu çerçevede ilgili kitap, makale ve tezler üzerinde "Doküman Analizi" yapılmıştır. Ortaya çıkan sonuçların farklı sektörlerdeki iş kollarında görevli ve ilgililere önemli ipuçları sağlayacağı beklenmektedir. Bu çalışmada ikincil kaynaklar kullanılarak konu ile ilgili tez, makale ve kitaplardan yararlanılarak "Belge taraması" yapılmıştır. Dolayısıyla bu çalışma Belge Tarama Modeli özelliği göstermektedir. Ayrıca konusunda uzman profesyonellerin mesleki tecrübelerinden de yararlanılarak, akademik bilgilerle sahada yer alan profesyonellerin tecrübeleri karşılaştırılarak Kısıtlar Teorisi Yaklaşımının Yazılım Geliştirme Sürecine Etkisi değerlendirilmiştir.

Çalışmanın birinci bölümünde kısıtlar teorisinin özelliklerinden bahsedilmiştir. Kısıt kavramı ve çeşitleri anlatılmaya çalışılmıştır. Kısıtlar teorisi düşünce süreçlerinden bahsedilerek, kısıt bulma ve yok etme yöntemlerinin nasıl işlediği açıklanmıştır. İkinci bölümde yazılım dünyası ve yazılım geliştirme süreçlerinden bahsedilerek yazılım geliştirme aşamaları ve modelleri açıklanmıştır. Bir yazılımın doğumundan kullanım ömrünün sonuna kadar olan bir süreçten bahsedilmektedir. Bir yazılım geliştirilirken geçtiği aşamaların açıklamaları yapılmıştır. Üçüncü bölümde kısıtlar teorisi düşünce süreçlerinin yazılım geliştirme sürecinde oluşan kısıtların bulunmasında ve bu kısıtların yok edilmesinde hangi yöntemleri kullandığı örneklerle açıklanmıştır. Kısıtlar teorisinin yazılım sektöründe kullanılmasının yazılım geliştirmeye ne gibi avantajlar sağladığından bahsedilmiştir.

BİRİNCİ BÖLÜM

KISITLAR TEORİSİ

1.1. Kısıt Kavramı ve Tanımı

Dünyada işletmeler teknolojiyi takip, etme çalışanların isteklerini artırma, bulunduğu pazarda rekabet gücünü artırma yönetim stratejilerini belirlemede bir sürü kısıtla karşılaşırlar bu gibi kısıtları ortadan kaldırmada kısıtlar teorisinin önemli rolü olduğu gözlenmektedir (Karagün ve Sözen, 2017).

Günümüz koşullarında artan nüfusun etkileriyle pazar çeşitliliği artmış durumdadır. Bu çeşitlilik sebebiyle firmaların iş kolları da genişlemiştir. Bu yeni pazarlara giriş yapan firmalar, pazarı tanıma ve taleplerine cevap verme sırasında tecrübesizliğinden de kaynaklanan birçok aksaklık ve sorunlarla karşılaşmaktadırlar. Bu sorunları çözemedikleri takdirde ya pazar payını küçültmek durumunda kalırlar ya da pazardan tamamen çekilmeyi düşünürler. Firmaların karşılaştıkları bu sorunlar birer "kısıt" olarak düşünüldüğünde, kısıtlar teorisi düşünce süreçleri bu kısıtların tespitinde ve yok edilmesinde kullanılabilir. Firmaların kısıtları ortadan kaldırmalarının, rekabet güçlerine pozitif yönde katkı sağlayacağı tartışılmazdır. Öz yetenek ve dış kaynak kullanımı konusunda da firmalara yol gösterecek olan kısıtlar teorisi düşünce süreçleri, firmaların nasıl daha rekabetçi olabilecekleri konusunda yol gösterici bir felsefe olarak yarar sağlamaktadır.

Kısıtlar teorisi sistemlerin gelişimine fayda sağlayan bir felsefe olarak da düşünülebilir (Blackstone, 2001, s. 1053-1080). Bir başka düşünce de kısıtlat teorisi, kısıtların yönetilmesi ve eş zamanlı üretim yöntemiyle devamlı gelişim hedefleyen bir yönetim felsefesi olarak adlandırılmaktadır (Atwater ve Gange, 1997, s. 14-22). Bu teori 1980'li yıllarda Dr. Eliyahu M. Goldratt tarafından geliştirilmiş olup temel kuralı sistemin verimini kısıtın direkt etkilediğidir (Rulh, 1997, s. 11-67).

Kısıtlar teorisi Toplam Kalite Yönetimi ve tam zamanlı üretime yol göstermekte olup, üretim planlamasında kullanılırken yıllar geçtikçe Goldratt tarafından geliştirilerek Maliyet Muhasebesiyle birlikte kullanılmıştır (Ronen, 2005, s. 1-2; Balderstone ve Keef, 1999, s. 26-28). Son yıllarda birçok sektör dalı kısıtlar teorisine başvurarak bu teorinin yöntemlerini verimli bir şekilde kullanmaktadır. Ticari kar güden işletmeler veya kar amacı olmayan dernekler dahi bu teoriyi uygulamaktadır. İş hacmi çeşitli ölçeklerde olan sektörler bu teori sayesinde kısa sürede önemli tecrübeler kazanarak güçlü atılımlar sağlamıştır (Ronen, 2005, s. 1-2). Kısıtlar teorisinin ne olduğunu anlayabilmemiz için öncelikle kısıtın ne anlama geldiğini bilmek gerekir.

Organizasyonlarda istenilmeyen şekilde ilerlemeyen durum veya olaylara kısıt denmektedir (MacArthur, 1993, s. 50-56). Kısıt "bir sistemin para kazanma hedefini başarmasını engelleyen herhangi bir unsur " şeklinde tanımlanmıştır (Umbela ve Srikanth, 1995, s. 50). Zaman açısından bakılırsa bir organizasyondaki en yavaş süreç tüm sistemdeki süreci etkilediği için kısıt en yavaş süreç olarak tanımlanır (Taylor ve Ortega, 2003, s. 9-14). Diğer bir değişle herhangi bir kaynağın kapasitesinden daha az kullanılan kapasite demektir. Kısıtlar karşılaşılan ve çözülmesi gereken her türlü sorun olarak tanımlanabilir (Taşçı, 2005, s. 73-74). Bir organizasyonun karşılaşabileceği iki temel kısıt vardır. Fiziksel kısıtlar ve yönetsel kısıtlar. Organizasyonlar genelde daha az fiziksel kısıtla karşılaşır. Karşılaşılan kısıtların büyük bir kısmı yönetsel kısıtlardır. Organizasyondaki kısıtların %99'u politik kısıtlardır.

Diğer taraftan Kısıtları ikiye ayırmak mümkündür. İş hacimleri, personel tutumları, stratejik politikalar v.b. gibi olgular iç kısıtlar olarak tanımlanırken, müşteri istekleri, tedarikçi firmanın kalitesi, siyasi yapı gibi olgular da dış kısıtlar olarak tanımlanabilir (Louderback ve Patterson, 1996, s. 189-196). Kısıtlar teorisinin savunduğu görüşlerden biri de her sistemde en az bir kısıtın bulunduğu ve bu kısıtın belirlenmesi ile ortadan kaldırılmasından sonra yeni bir kısıt oluşması ve bu sarmalın böyle sürmesidir (Louderback ve Patterson, 1996, s. 189-196).

Kısıtları bulabilmek için, Kısıtlar Teorisi felsefesi çeşitli akış şemaları oluşturarak kısıtın temel nedenini bulmak ve kısıtı yok etmek için yollar arar (Rudy, 1996, s. 23). Kısıtlar teorisi felsefesi sistemleri kontrol ederken hem misyonun temelini sağlamlaştırır hem de vizyonunun kılavuzu olabilir. Hedefe ulaşmakta ki engelleri tespit ederek, engelleri yok etmek için ihtiyaç olan farklılıkları hayata geçirmeyi tasarlayan bir yaklaşımdır (Schucavage, 1995, s. 15-16).

1.2. Kısıtlar Teorisinin Temel Özellikleri

Kısıtlar Teorisi'nin en önemli özelliği her sektörde ve her alanda uygulanabiliyor olmasıdır (Atay, 2009, s. 4). Yaklaşımın özü; organizasyondaki engelleri bulmak ve bu engellerin üretim süreciyle senkronize bir şekilde kullanılabilmesine hükmetmektir (Rand, 2000, s. 173-187). Kısıtlar teorisinde genelde yaygın olarak, kısıtları ortadan kaldırmak için iki yöntem kullanılmaktadır. Birincisi "parçala veya böl" olarak bilinen, sistemi parçalara ayırarak incelemek ve düzeltmek, sonra bu parçaları birleştirerek bütünde iyileşmenin yaşanmasını sağlamaktır. İkincisi ise sistemi bütün olarak değerlendirip her kademesinin performansını en üst seviyeye çıkarmak; sistemin bütün performansının da en üst seviyeye ulaşmasını sağlamaktır (Filiz, 2008, s. 235).

Dettmer; Goldratt'ın Kısıtlar Teorisinin daha iyi işleyebilmesi için birçok ilkeyi bir arada kullandığını savunarak bu ilkeleri şöyle açıklamıştır (Gürgen ve Gençyılmaz, 2008, s. 24-25).

- Kısıtı olmayan birimin geliştirilmesi kısıtı olan birime etki etmez, organizasyona faydalı olmaz.
- Önemli aksaklıklar organizasyonun bağlantıları içinde oluşan umulmayan reaksiyonlar sonunda tespit edilir.
- Önemli sorunlar gelişi güzel değildir. Organizasyonlar neden sonuç ilişkilerine dayanır.

- Kısıtlar maddesel ya da siyasal olabilir. Maddesel kısıtların belirlenmesi kolay olur ve ortadan kaldırılması da basittir.
- Siyasal kısıtların tespit edilmesi ve ortadan kaldırılması çok daha zordur. Fakat; siyasal kısıtlar ortadan kaldırıldığı takdirde organizasyon için çok daha faydalı sonuçlar doğurduğu görülmektedir.
- Kısıtı ortadan kaldıran yöntemin randımanlı olması için yöntemle ilgili sürekli gelişime yapılmalıdır.
- Değiştirilecek yönteme karar verirken organizasyonun iyi analiz edilmesi gerekmektedir. Organizasyonun Misyon - Vizyon'unun iyi anlaşılması ve organizasyonun büyüklüğü iyi bilinmelidir.
- Sorunlar genellikle saklı paradokslar sayesinde kendini gösterir. Bu tür sorunları çözebilmek için bu paradoksların hipotezinin yapılmasını gerekli kılar.
- Sabit kalmak iyileştirmenin baş düşmanıdır. Kısıtlara bulunan çareler, yapılabilecek farklılaştırmalar için öğrenme kabiliyeti sağlar.
- Hayata geçirilmemiş düşünceler organizasyonlar için çözüm olamaz.
- Organizasyonlar zincir gibi düşünüldüğünde her zincirin bir zayıf halkası olduğu da bilinir.
- Verimliliğin artırılabilmesi esas kısıtın doru şekilde tespit edilmesiyle başarılır.
- Organizasyonlarda ki verimsizliğin sebebi birden fazla kısıt olabilir.

1.3. Kısıtlar Teorisindeki Beş Ana İlke

Organizasyonlarda iki tür kaynak vardır: Bunlar "darboğazı bulunan kaynaklar ve darboğazı olmayan kaynaklardır". Darboğaz bulunan kaynak kapasitesi talebe eşit ya da talepten daha az olandır. Darboğaz olmayan kaynak ise, kapasitesi talebin üzerinde olan kaynaktır. Kapasite taleple dengelenmemelidir. Kapasite değil, akış dengelenmelidir. Organizasyonun gerçek kapasitesini darboğazlar belirler.

Bir sistemdeki sureci iyileştirmek için önce zayıf halkalar belirlenmelidir. Takip edilmesi gereken süreç aşağıda belirtildiği gibi 5 adımdan oluşmaktadır (Cox, 2004, s. 81).

1.Adim: Sistemin darboğazları belirlenmelidir.

2.Adim: Darboğazların nasıl kullanılacağı tespit edilmelidir.

3.Adim: Geri kalan her şeyin ilk iki maddeyle desteklemesi sağlanmalıdır.

4.Adim: Sistemin darboğazlarının verimliliği arttırılmalıdır.

5.Adim: Önceki adımlardan herhangi birisinde bir darboğaz oluşursa, ilk adıma geri dönülmelidir.

1.4. Kısıt Çeşitleri

Kısıt sınıflandırmasından daha önce ki paragraflarda bahsedilmişti. Hatırlanacağı üzere; iki çeşit sınıflandırma yapılmıştı. İç kısıtlar organizasyonun kontrolünde olan birimlerde oluşan kısıtlar; örneğin; çalışanlar, yöneticiler, üretim iş hacmi gibi. Dış kısıtlar organizasyonun kontrolü dışındaki faktörlerde oluşan kısıtlardır. (Ör Pazar, ham madde sağlayıcıları) (Louderback ve Patterson, 1996, s. 189-196). Kayıtlarda kısıt türlerinin aşağıda belirtildiği gibi birçok dala ayrılmıştır: (Küçükyavaş, Tanış ve Ünal, 2006, s. 17-28).

- Malzeme kısıtları
- Yönetsel kısıtlar
- Kapasite kısıtları
- Lojistik kısıtlar
- Pazar kısıtları
- Davranışsal kısıtlar
- Zorunlu durumlar
- Maliyet yaklaşımı

1.4.1. Malzeme Kısıtları

İşletmenin dışarıdan temin ettiği hammaddelerin karşılanmasında oluşan aksaklıklara malzeme kısıtları denir (Ünal, 2000, s. 8). Kısa dönemli malzeme kısıtları, uzun dönemli malzeme kısıtları olmak üzere iki farklı kategoride incelenmektedir. Kısa dönemli kısıt; üretilmiş ürünlerde beklenmedik defoların oluşması ya da istenilen ürünlerin zamanında teslim edilmemesi gibi durumlara örnek olurken, piyasalarda ki hammadde yetersizliği de uzun dönemli kısıt'a örnek olarak gösterilebilir. İşletmeler kısıtlara önlem alabilmek için yeni hammadde arayışına girerek farklı ve daha maliyetli tedarikçilerle çalışmayı göze alabilirler (Umbela ve Srikanth, 1995, s. 50).

1.4.2. Yönetsel Kısıtlar

Literatürde politik kısıtlar olarak da geçen bu tanım geleceği görme konusunda yetenekleri sınırlı olan üst yöneticiler tarafından işletmenin önündeki yatırım fırsatlarını kaçırmalarına neden olduklarında oluşmaktadır (Atwater ve Gange, 1997, s. 14-22).

Kaynakların etkin kullanılamaması yönetsel kısıtların başlıca etkilerindendir. Büyüme yeteneğini engelleyerek düzgün çalışan sistemin sağlıklı kullanılmasını engellerler (Kartal, 2006, s. 8).

İşletmelerde görülen kısıtların çoğu yönetsel kısıtlardır. Politik kısıtların dışında kalan kısıtların tespiti ve giderilmesi daha kolaydır; fakat Politik kısıtların giderilmesi işletmelere çok daha fazla yararlı olmaktadır (Rahman, 2002, s. 809-828).

Politik kısıtlar, işletmelerin işlevselliğini sekteye uğratmak ve sistemi etkileyen başka kısıtların zararlarını arttırmak gibi iki önemli unsuru ortaya çıkarmaktadır (Utku, 2007, s. 1634).

Yönetim kısıtları kurallara bağlanabilirler. Bu kurallar sözlü kurallar, yazılı kurallar olarak ikiye ayrılabilirler. Çeşitli araştırmalarda sözlü kuralların yaptırım gücünün yazılı kurallara göre daha zayıf olduğu anlaşılmıştır. Örneğin; Ankara-İstanbul arası seyahat eden bir aracın 150 km/saat hıza ulaşabilmesi mümkünken, trafik kurallarına göre bu yol üzerinde en fazla 120 km/saat hızın üzerine çıkması yasaktır. Takografi verileri üzerinden aracın hız kontrolü yapılabilmektedir. Diğer örnek ise bir işletmede toplantılarda yapılacak sunumların 15 dakikayı geçmemesini, yönetim kurulu tarafından verilmiş bir karar olarak kabul edelim, yapılan bir sunumun 15 dakika geçmesinin herhangi bir yaptırıma maruz kalmayacağı ön görülebilmektedir (Atay, 2009, s. 4).

Yöneticiler, şirketin fırsatları değerlendirmesini önleyecek kurallar koymuş olabilir. Yanlış yönetim şekilleri sistemin önüne çıkan fırsatları engelleyerek kısıta yol açabilir. Bir başka açıdan bakıldığında tasarlanan yönetim şekilleri kısıtları yönetmekte başarısız olurken düzgün çalışan bir sistemde kısıta da yol açabilir (Kaygusuz, 2005, s. 133-156).

1.4.3. Kapasite Kısıtları

Kapasite kısıtları bu teorinin en önemli kısıtlarından biridir. Genellikle Pazar taleplerinin karşılanması veya üretim esnasında çalışan yetersizliği ya da makine ve teçhizatların teknolojik yetersizliği gibi durumlar kapasite kısıtlarına örnek olabilir (Karagün ve Sözen, 2017, s. 184).

Üretim yapan işletmelerde ilk incelenmesi gereken kısıtların başında gelen kısıtlar kapasite kısıtlarıdır. Her üretim işletmesinde sistemin sağlıklı işlemesini engelleyen değişik darboğazlar bulunduğu düşünülmektedir. Sistem en zayıf halkası kadar güçlüdür mantığından yola çıkılarak herhangi bir kapasite kısıtının üretim aşamalarının tümünü tetikleyerek tam kapasite çalışmasını engelleyecektir. Eğer bir üretimin talebi arzından az ise bu durumda kapasite kısıtı oluşumu gözlenmektedir. Firmanın tesirli olabilmesi için bu kısıtları ortadan kaldırması gereklidir (Utku, 2007, s. 1634).

Kısıtlar teorisinde kapasite kısıtının önemli oluşunun nedeni; üretilen bütün ürünlerin miktarının bu kısıt tarafından belirlenmesidir. Üretim sadece bu kısıtın ortadan kaldırılması ile arttırılabilir. Sistem planlanmadan önce kısıtlı kaynakların yanında kısıtı olmayan kaynakların da belirlenmesi önemlidir. Zira sistem bu belirlemeler üzerine planlanacaktır (Ünal, 2000, s. 8).

Firmaların pazarlardan gelen isteklere yeterince cevap verememesi kapasitelerinin pazarın altında olduğunu göstermektedir. Bu da kapasite yetersizliği yani kapasite kısıtı olarak adlandırılabilir. İnsan kaynağının zayıf olması veya kullanılan makinelerin teknolojilerinin eski olması kapasite kısıtlarındandır (Kaygusuz, 2005, s. 133-156).

Düzgün işleyen sistemlerde, mevsimlik talep artışları olduğu zaman arzın talebin altında kalması gibi durumlar söz konusu olabilir. Böyle durumlarda bu sistemler kısıtı olan sistemler gibi görüne bilmektedir. Bu durumlara önlem alınmak istendiğinde firmalar için kaynak israfı oluşmaktadır. Bu durumda karşımıza öncelikli olarak; fırsat maliyeti, yarı mamul ve mamul stoku fazlası çıkmaktadır. Fırsat maliyeti; yapılmasından vazgeçilen bir durudan sağlanacak net kazancın kaybedilmesi olarak tanımlanabilmektedir (Büyükmirza, 2003, s. 9).

1.4.4. Lojistik Kısıtları

Hammadde veya diğer malzemelerin temininden ürünlerin hedef pazara ulaştırılmasına kadar yerine getirilen süreçlere lojistik denir (Erol, 2008, s. 39).

Firmalar, hammadde kısıtı, ürünlerin dağıtımı, gibi kısıtlarla yakından ilişkili olarak lojistik kısıtlarla da karşı karşıya kalmaktadır. Kısa süreli hammadde sıkıntısı ile yakından ilgisi olan lojistik sorunlar üretim için lazım olan mamullerin firmanın üretimine yetiştirilmesini içerir. Dolayısıyla zaten mevcut stok fazlaları ve üretim sıkıntıları kendisini iyice açığa çıkarır (Karamaraş, 2002, s. 12).

Firmaların kontrol ve planlama sistemlerinde oluşan problemler nedeniyle, lojistik kısıtların oluştuğu söylenebilir. Sistemde ihtiyaç olmayan malzemelerin

planlanması kaynakların koordine edilmesinde kısıtlara yol açabilir. Bu sebepten üretim ve stok problemleri ortaya çıkabilir. Örneğin: satın alınacak malzemeler için birçok farklı tedarikçiden en ucuz olanının tespit edilerek satın alınması gibi ağır işleyen bir süreç üretimin yavaşlamasını veya istenilen zamana yetiştirilememesine yol açacaktır (Kartal, 2006, s. 30).

Lojistik kısıtlarının en önemli olanlarından birsi de üretilen ürünlerin sipariş edene ulaştırılması sorunudur. Sipariş ile ifade edilen, firmaların hammadde tedariki için verdiği siparişler ve işletmenin ürettiği mamullerin diğer işletmeler veya müşteriler tarafından temini için alınan siparişlerdir. Lojistik bağlantıları işletmenin üretim süreci ile senkronize durumda değilse, işletmeler için lojistik kısıtların baş göstermesi olasıdır (Atwater ve Gange, 1997, s. 14-22; Ünal, 2000, s. 8; Karamaraş, 2002).

1.4.5. Pazar Kısıtları

Kısıtların en önemlilerinden birisi de pazar kısıtıdır. Pazar üretilen hizmet veya ürünlerin kalitesini, fiyatını, hazırlanış zamanını, inceleyerek gereken talebi ortaya koyar. İşletmenin kaynakları, pazarın talep miktarından fazla ise pir Pazar kısıtı meydana gelmektedir (Kartal, 2006, s. 30).

İşletmenin yaşamını devam ettirebilmesi ürettiği ürünleri ya da hizmeti satabileceği bir pazara bağlıdır. Firmanın mevcut pazarının talebini karşılaması önceliklidir. Bu sebepten işletmeleri için pazar kısıtı en önemli sorunlardan biridir (Büyükyılmaz ve Gürkan, 2009, s. 177-195).

İşletmelerin bünyesinde bulunan kısıtlardan değildir. Firmaların ürettiği hizmet ya da ürünlerin pazar tarafından yeterince rağbet görmemesi veya pazarın talep miktarının firmanın üretiminin altında kalması pazar kısıtına sebep olmaktadır. Pazar kısıtlarını bir nedene bağlamak yanlış olacaktır, fakat Pazar kısıtlarının ortaya çıkmasının en büyük nedenlerinden biri yönetim politikalarıdır (Atwater ve Gange, 1997, s. 14-22).

1.4.6. Davranışsal Kısıtlar

Çevrede oluşan olaylara mantıklı reaksiyonlar vermek davranış olarak tanımlanabilir. Davranışlar, insanların eğitim seviyeleri, ailelerinden aldığı kültür ve zihinsel kapasitelerine göre gelişir. Oluşan bir davranış realite ile zıtlaşma içerisinde ise ve organizasyonun global ölçümleri üzerinde tam tersi bir etki oluştururursa davranışsal kısıt ortaya çıkar (Kartal, 2006, s. 30).

Karşılaşılan durumlara verilen spesifik tepkilere davranış denilebilir. Bu tepkiler çeşitli kriterlere göre değişimler gösterebilirler (Örneğin; eğitim, mantık, tecrübe). Davranış eğer gerçeklik ile ters düşer ve organizasyonun işleyişinde sıkıntıya yol açarsa davranışsan kısıt oluşturabilir. Davranışsal kısıtların organizasyon üzerindeki etkisini kontrol etmek zordur. Organizasyonun karına ve organizasyonun verimine ne ölçüde etki ettiğini tespit etmek oldukça önemlidir (Kaygusuz, 2005).

"Bana beni nasıl ölçeceğini söyle, ben de sana nasıl davranacağımı söyleyeyim", "Beni mantıksız bir şekilde ölçersen, mantıksız davranışlarımdan şikâyetçi olma", "Benim ölçütlerimi tam olarak kavrayamadığım yenileri ile değiştirirsen, kimse nasıl davranacağını bilmeyecektir. Ben bile…". Goldratt'ın bu cümleleri ile özetlediği performans değerlendirme yöntemlerinin, davranışsal kısıtlar ile yakından alakalı olduğu savunulabilir.

Davranışsal kısıtlar, organizasyonun içindeki diğer sıkıntıların majör sebebi değilseler bile yok edilmeleri işletmenin performans artırımı açısından önemidirler. Bu sebepten bu tür kısıtları temel bir engel olarak nitelendirmek mümkündür. Ortadan kaldırılmaları oldukça zordur (Utku, 2007, s. 1634).

En zararlı tutumlarda birisi de kaynakların aralıksız kullanılmasıdır. Çalışanların sürekli işe odaklandırılmaları üretkenlik olarak nitelendirildiği için bu tutum oluşmaktadır. Bu tutumun savunduğu durum tüm kaynaklar en yüksek seviyede kullanılırsa istenilen kar elde edilir düşüncesinin kullanım yüzdesi ölçümlerinle de desteklenmektedir. Bu tutum hem yönetim hem de çalışanlar tarafından kabul edilmiştir. Bu yüzden malzeme açıkları, fazla stok, dengesiz ürün

karmaları, çizelgelerin kayması gibi durumlar oluşur. Başka bir örnek vermek gerekirse; hazırlık sırasında sonuçların ayrıntılarını değerlendirmeden tasarruf tedbirlerinin gerçekleştirimle düşüncesidir. Hazırlıkların girdiler, stok, çıktılar gibi işlemlerin üzerindeki ayrıntılı etkisini dikkate almamak, toplam kar üzerinde bir eksilmeye neden olabilir. İşletmelerin karları üzerindeki etkisi genellikle tahmin edilebilmektedir (Kartal, 2006, s. 30).

1.4.7. Zorunlu Durumlar

Organizasyonlardaki işleyişlerin düzenlenmesine, bir takım zorunlu durumlar getirilebilir. Bunlar kişilere veya bölümlere getirilmiş sınırlamalar olabilir. Örneğin çevre temizliği için kanuni düzenlemeler, kaliteli üretim, şeffaflık gibi durumlar olabilir. Yöneticiler çalışanlarına kurallar koyabilir, hisse sahipleri ise kar payı dağıtımı gibi şartlar koşabilir. Bu gibi şartlar yerine getirilmediği zaman çeşitli sıkıntılar baş gösterebilir. Kısıtlar ile zorunlu durumları birbirinden ayırmak gereklidir. Zorunlu durumlardan bir tanesi yerine getirilmezse kısıta dönüşme ihtimali oldukça yüksektir. Bu zorunlu durumların karşılanması süreklilik arz etmek durumundadır, aksi halde işletmenin karlılığındaki iyileşme önünde kısıt olmaya devam edecektir (Kartal, 2006, s. 30).

1.4.8. Maliyet Yaklaşımı

Firmaların genelinde, sistemi bölerek, yerel ölçüde en iyi şekilde sonuçlanan ölçümler geliştirme konusunda eğilimler vardır. Yerel ölçümlerin nasıl kullanılacağı okullarda öğretilmekte ve iş dünyasında da desteklenmektedir. Buna rağmen, olumlu küresel bir etkiyle sonuçlanmayan yerel ölçümler haricinde, yerel ölçüm sistemleri geçersiz olarak görülmelidir. Maliyet yaklaşımı çoğu kısıtın çıkış nedeni olan ve belki de işletmeler için en büyük tehlikelerden biridir (Stein, 1996, s. 2).

1.5. Kısıtlar Teorisindeki Düşünce Süreçleri

Kısıtların tespiti için kısıtlar teorisi tarafından bir takım mantıksal düşünce süreçleri geliştirilmiştir. Goldratt "Mantıksal Düşünme araçları" adını verdiği aşamalar geliştirerek bunlara aşağıdaki isimleri vermiştir.

- Bugünkü Gerçeklik Ağacı
- Buharlaşan Bulut
- Gelecekteki Gerçeklik Ağacı
- Ön Şart Ağacı
- Geçiş Ağacı

Bu araçları karmaşık sistemleri analiz edebilmek için belirli bir düzen için de kullanmak gerekir (Utku, 2007, s. 1634). Kısıtlat teorisi üç soru ve bunların diyagramları üzerinde duraktadır (Dettmer, 1997, s. 23; Scheinkopf, 1999, s. 5-37). Bunlar "Ne Değişecek", "Neye Dönüşecek", "Dönüşüm Nasıl Gerçekleşecek" dir.

Ne değişecek: "Mevcut Gerçeklik Ağacı" ve "Buharlaşan Bulut" diyagramları bu soruya cevap ararken kullanılır.

Neye dönüşecek: "Gelecek Gerçeklik ağacı" diyagramı ile cevaplandırılır. Sistemin gelecekte olmasının istenildiği durumdur.

Dönüşüm nasıl gerçekleşecek: "Ön şart ağacı" ve "geçiş ağacı" diyagramları kullanılırken, iyileştirmenin nasıl yapılacağı sorusuna cevap aranır (Scheinkopf, 1999, s. 5-37).

1.5.1. Bugünkü Gerçeklik Ağacı

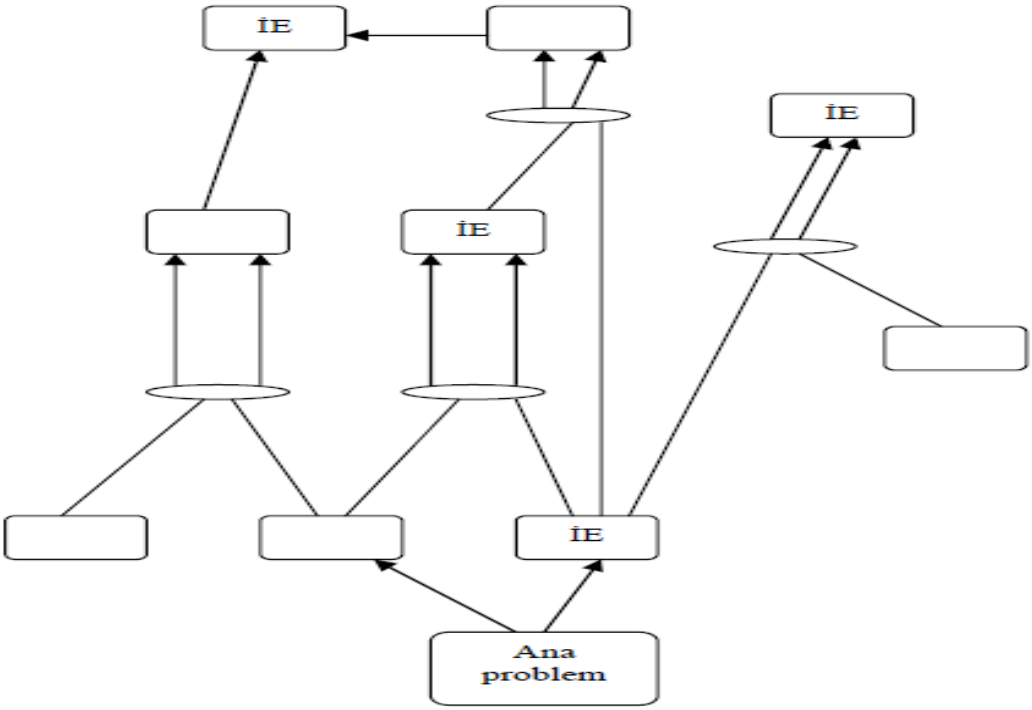
Literatürde "Mevcut Gerçeklik Ağacı" olarak da geçen "Bugünkü Gerçeklik Ağacı" organizasyonların temel kısıtlarını bulmak için kullanılan bir yöntemdir. Temel kısıt organizasyonun verimini azaltan ve istenmeyen durumların ortaya çıkmasındaki en önemli etkendir. Temel kısıtların fiziksel veya politik kısıtlara dayanması "Bugünkü Gerçeklik Ağacı" yöntemi için bir fark yaratmaz ve bu kısıtların tespiti için oldukça tesirli bir yöntemdir (Scoggin ve Segelhorst, 2003, s. 767-797).

Sistemin şu an ki durumunu görmek için kullanılan "Bugünkü Gerçeklik Ağacı" neden-sonuç ilişkilerini ortaya koyar. "Mevcut Gerçeklik Ağacı" karmaşık problemleri anlaşılabilir hale getirerek temel kısıtlara odaklanılmasını sağlamaktadır. Bu yöntem organizasyonun içinde bulunduğu durumu tespit etmek kısıtları çözebilmek için kullanılır (Sadıç, Özdemir ve Gözlü, 2006, s. 99-118).

"Mevcut Gerçeklik Ağacı" arzu edilmeyen bir durumdan kök nedene ulaşıncaya kadar neden-sonuç ilişkilerinin bağlandığı yöntemdir (Kartal, 2006, s. 30).

"Bugünkü Gerçeklik Ağacı"nın başarmak istediği amaçları Dettmer (1997) şu şekilde özetlemiştir (Utku, 2007, s. 1634):

- İstenmeyen faktörlerin tespitini sağlamak
- Temel bir problemin teşhisini sağlamak
- Organizasyonun en iyi duruma ulaşması için kısıtları ortadan kaldırmak
- Organizasyon üzerindeki en olumlu etki için en basit iyileştirmeyi yapmak
- Etki alanı dışında kalan kök sorunlara neyin neden olduğunu anlamak
- Neden-sonuç mantığıyla istenmeyen etkilerin ilişkisini tespit etmek



İE: İstenmeyen Etki
Şekil 1: Bugünkü Gerçeklik Ağacı
Kaynak : (Dettmer, 1997)

1.5.2. Buharlaşan Bulut

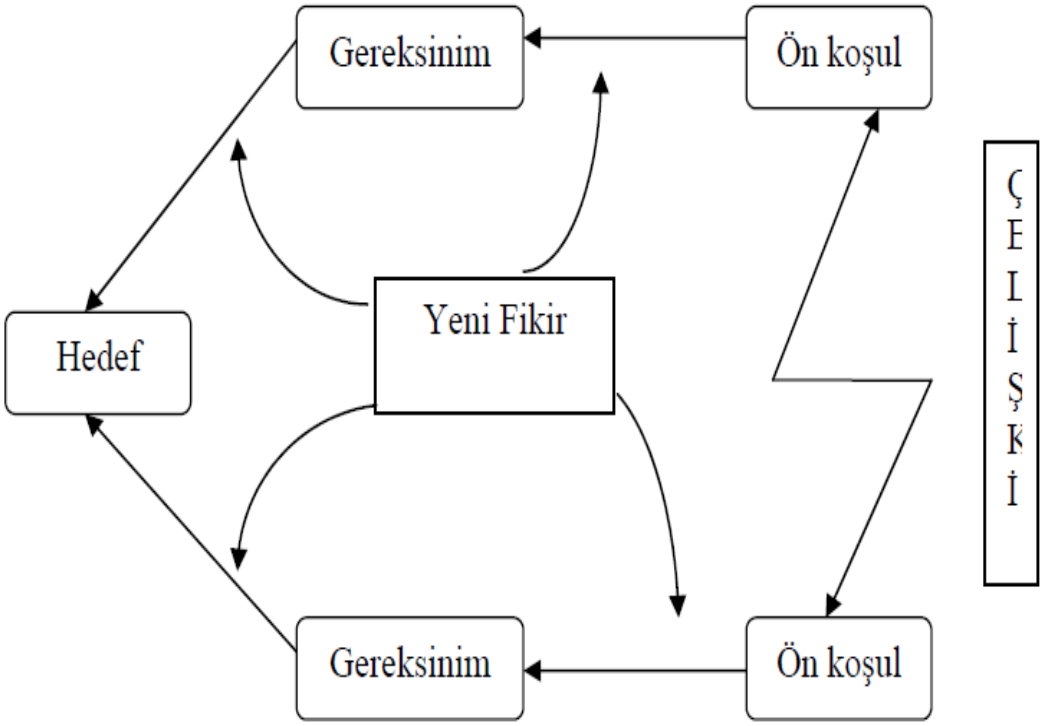
Literatürde çelişki çözüm diyagramı olarak ta geçen Buharlaşan Bulut organizasyonların karşılaştığı problemleri tek tek inceleyerek sorunların tespit edilmesinde önemli rol oynar. Bu yöntem kısıtı ve yarattığı problemi ortadan kaldırarak istenilen aşamaya geçişi sağlar (Rahman, 2002, s. 809-828).

Bu yöntemde kısıtı yok edebilmek için çözüm önerileri ve gereksinimler belirlenir. Bu çözümler arasındaki çelişkiler gidermek için müdahalelerde bulunulur. Karşılaşılan çatışmalara çözüm amacı ile yaklaşılır. Buharlaşan bulut sıkıntılı durumdan ideal duruma geçiş için kısıtın ortadan kaldırılmasına katkı sağlar ve diğer aşamalara geçişe bir bağ oluşturur. Temel kısıtlara etkili ve kesin çözümler üretilir. Bu yaklaşımda kaç tane sorun varsa o kadar buharlaşan bulut üretilmelidir.

Buharlaşan bulut çıkan çatışmaları birleştirerek çatışmalarda ortaya atılan varsayımlardan, çatışan tarafların yanlışlarını fark ettirerek bu çatışmaların bir buhar gibi dağılmasını sağlar (Atay, 2009, s. 4).

Dettmer'e göre Buharlaşan Bulut Yöntemi bazı amaçlar için yaratılmıştır. (Utku, 2007, s. 1634) Bunlar aşağıda belirtilmiştir:

- Çatışmanın olduğunu ispatlamak
- Bir çatışmanın ana problemini belirlemek
- Çatışmayı çözüme kavuşturmak
- Çatışanlar için uygun çözümler bulmak
- Sorunun çıkış nedenini anlamak
- Çatışmaların tüm varsayımlarını incelemek.



Şekil 2: Çelişki Çözüm Diyagramı
Kaynak: (Scheinkopf, 1999)

1.5.3. Gelecekteki Gerçeklik Ağacı

Olması istenen etkinin oluşmasına sağlamaya yönelik geliştirilmiş bir yöntem olan Gelecekteki Gerçeklik Ağacı bu etkinin nasıl oluşacağı sorunsalını çözer. Bu sebepten Bugünkü Gerçeklik Ağacına göre farklıdır. Uygulamaya başlanmadan önce nelere sahip olunduğunun ve neler gerektiğinin anlaşılabilmesi için gerekli yapıyı oluşturur. Bu yöntem sistemin geleceğinde etkili olacak yöntemi belirleyerek olumsuz etkiler için uyarılarda bulunur. Gelecekteki Gerçeklik Ağacı bir eğer-ne çatışmasıdır. Gelecekteki Gerçeklik Ağacı ile Bugünkü Gerçeklik Ağacı arasındaki farklar ise şunlardır (Atay, 2009, s. 41);

- Bugünkü Gerçeklik Ağacı şu anki durumu incelerken Gelecekteki Gerçeklik Ağacı istenilen etki üzerinde yoğunlaşır.
- Gelecekteki Gerçeklik Ağacı eksikliklerin ortaya çıkmasını sağlarken üretilen çözümlerin geliştirilmesini sağlar.
- İstenilene ulaşmadaki bütün pozitif unsurları tanımlar başarısızlığa uğranıldığında ise sebepleri ortaya çıkarır.

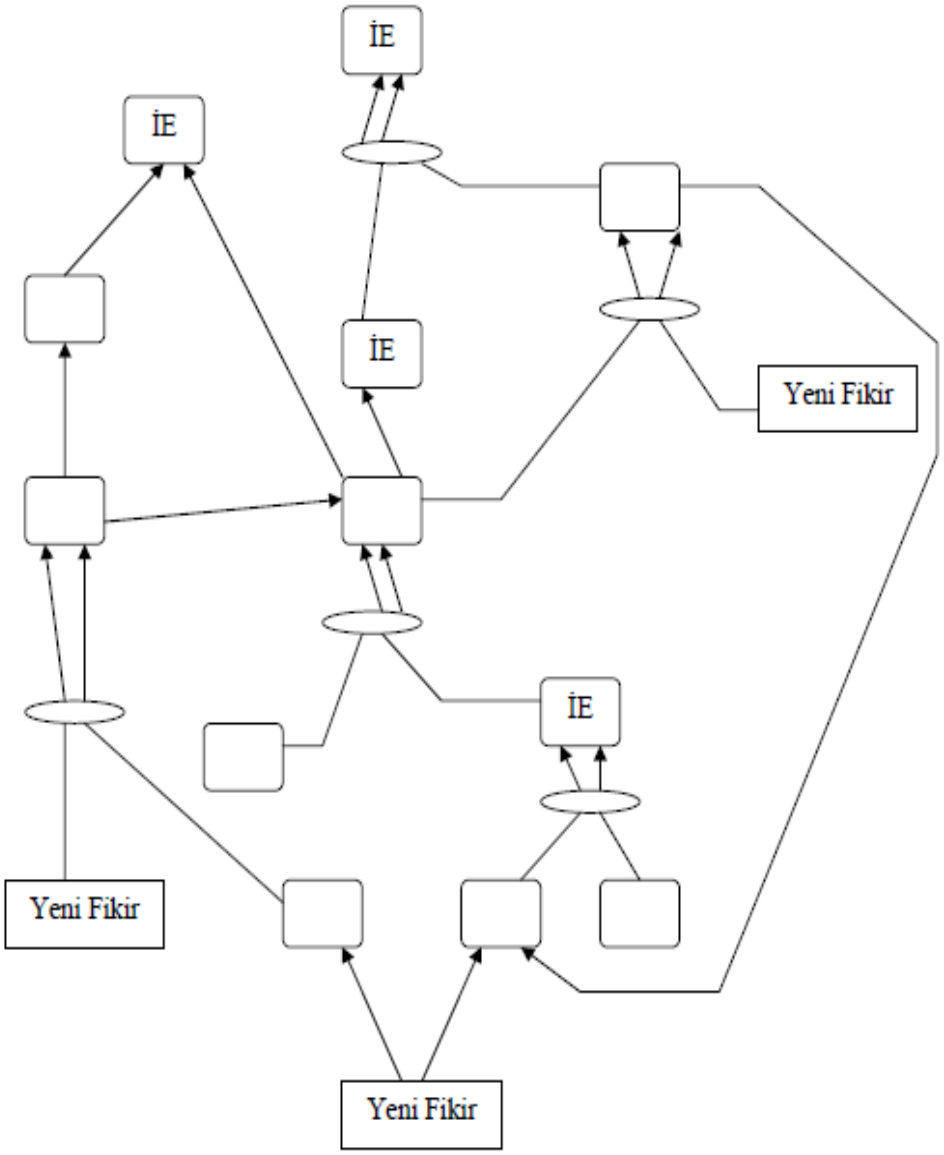
Gelecekteki Gerçeklik Ağacı organizasyondaki değişiklikler ile oluşan sonuçlar arasındaki ilişkiyi gösterir. Bu yöntem işletmenin ileri görüşlülüğünün gelişmesine büyük katkı sağlar. Düşünülen değişimin getireceği fayda ve zararları tespit ederek bu etkilere nasıl tepkiler verilmesi gerektiğini gösterir. Bugünkü gerçeklik ağacı ile gelecekteki gerçeklik ağacını birbirinde ayıran en belirgin farklılık ise bugünkü gerçeklik ağacının istenmeyen sonuçları ilişkilendirirken gelecekteki gerçeklik ağacı belirlediğimiz çözüm yolu ile istenilen etkileri ilişkilendirir (Kartal, 2006, s. 30).

Dettmer'in (1997) söylemine göre Gelecekteki Gerçeklik Ağacı aşağıdaki aşamalar için düşünülmüştür (Utku, 2007, s. 1674):

- Organizasyondaki istenilen etkilerin negatif bir sonuç olmadan sağlanabileceğine karar vermek
- Organizasyondaki içsel kararların etkilerini incelemek

- İstenilen etkinlikler için bir araç bulmak
- Etkili bir planlama aracı sunmak

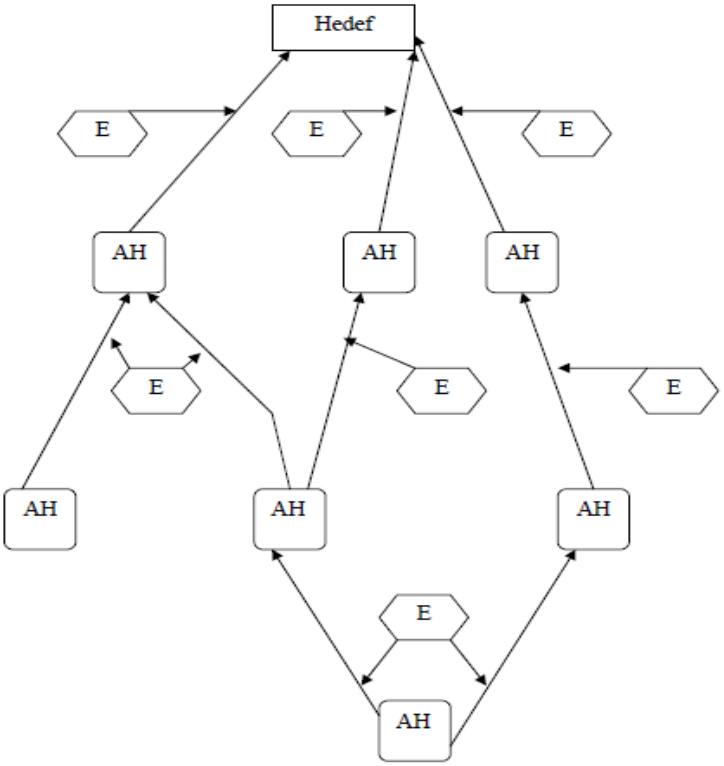
Gelecekteki Gerçeklik Ağacı alınan kararların sonuçlarının ne olacağının anlaşılmasını sağlayarak bu sayede negatif etkileri erkenden önlenebilmesini sağlayabilir. Bu yöntem var olan kısıtın neyle değiştirileceğini karar bağlar.



İE: İstenen Etki
Şekil 3: Gelecekteki Gerçeklik Ağacı
Kaynak: (Dettmer, 1997)

1.5.4. Ön Şart Ağacı

Literatürde ön koşul ağacı olarak da geçen bu yöntem organizasyondaki değişimin nasıl olacağını belirler.



AH: Ara Hedef, E: Engel
Şekil 4: Ön Koşul Ağacı
Kaynak :(Scheinkopf, 1999)

Bu sistem çözüm önerilerini engelleyen unsurları ortadan kaldırılmasında kullanılabilecek olan çözümlerin oluşturulması mantıksal bir yol arar. Ön şart ağacı istenilen sonuçlara engel olan sebepleri belirler. Bu engelleri kaldırmayı sağlayacak amaçları belirler (Kartal, 2006, s. 30).

Bir planın hazırlanmasından sonra bunların uygulamasında oluşacak engelleri ortadan kaldırmak için kullanılacak adımların bir sırasını belirlemektedir. Adım adım bir planın uygulanış şeklidir (Utku, 2007, s. 1634).

Dettmer'e (1997) göre, ön şart ağacı ile şu amaçlar başarılabilir;

- İstenilen faaliyetlerin başarılmasındaki engelleri belirlemek
- Sorunlara çözüm aramak.

1.5.5. Geçiş Ağacı

Geçiş ağacı adım adım uygulanır. Bu yöntemle sistemin bulunduğu durumdan arzu edilen duruma geçişi sağlanır. Geçiş Ağacı'nı oluşturabilmek için, Ön Şart Ağacı'nda ara amaçlara ulaşmak için gereken eylemleri o anda bulunduran ortam dikkate alınır. Planlanan olayların nedenlerinin belirtilmesi planın alt kademelere aktarılması sırasında ihtiyaç duyulan doğrulamaları otomatikman yapmasıdır. Bu yöntemin en önemli özelliği değişime karşı gelme direncinin aşılmasına yardımcı olmaktır. Bu yöntem "değişim nasıl gerçekleştirilecek?" sorusuna verilecek cevaptır (Kartal, 2006, s. 30).

Gerekli işlemleri tanımlayan Geçiş Ağacı, sistemin istenilen duruma gelmesinde temel unsur olarak karşımıza çıkar. Sistemin başından sonuna kadar değişim sürecindeki olumlu olumsuz bütün sonuçları sebep – sonuç şeklinde gösteren bir zincirler bütündür. Değişimlerin yerinde ve zamanında yapılmasında Geçiş Ağacı kullanılır (Atay, 2009, s. 4).

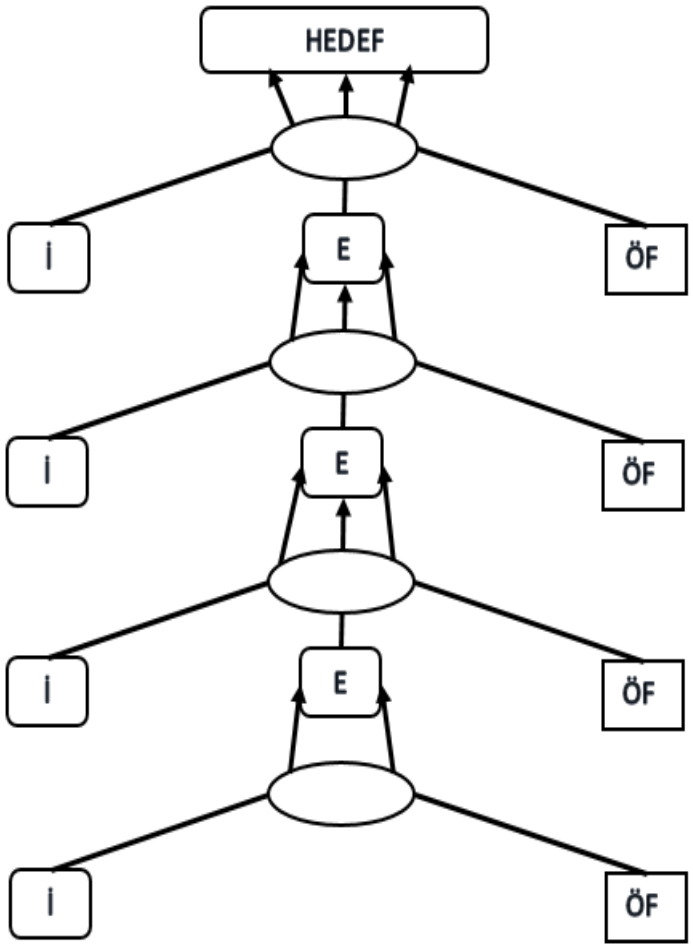
"Nasıl değiştirilecek?" sorusunun ikinci kısım cevabı Geçiş Ağacı'dır. Bu yöntem sayesinde ihtiyaçlar, eylemler ve bunların etkileri gözlemlenmektedir. Tanımlanan hedeflere ulaşılabilmesi için verilen kararlara nasıl ulaşılacağına adım adım rehberlik yaparak yol gösterir. Geçiş ağacı projenin tamamlanmasında bir kontrol listesi gibi işlevsellik gösterir.

Geçiş Ağacının birtakım amaçları başarmak işin tasarlanmış olup Dettmer (1997) bunları şöyle sıralamıştır (Utku, 2007, s. 1634):

- Adım adım uygulanan yöntemler çalışmaları yerine getirmek,
- Gelecekteki Gerçeklik Ağacı ya da Buharlaşan Bulut'da ki gelişmeleri uygulamak,
- Ön Şart Ağacı'ndaki amaçlara ulaşmayı sağlamak,

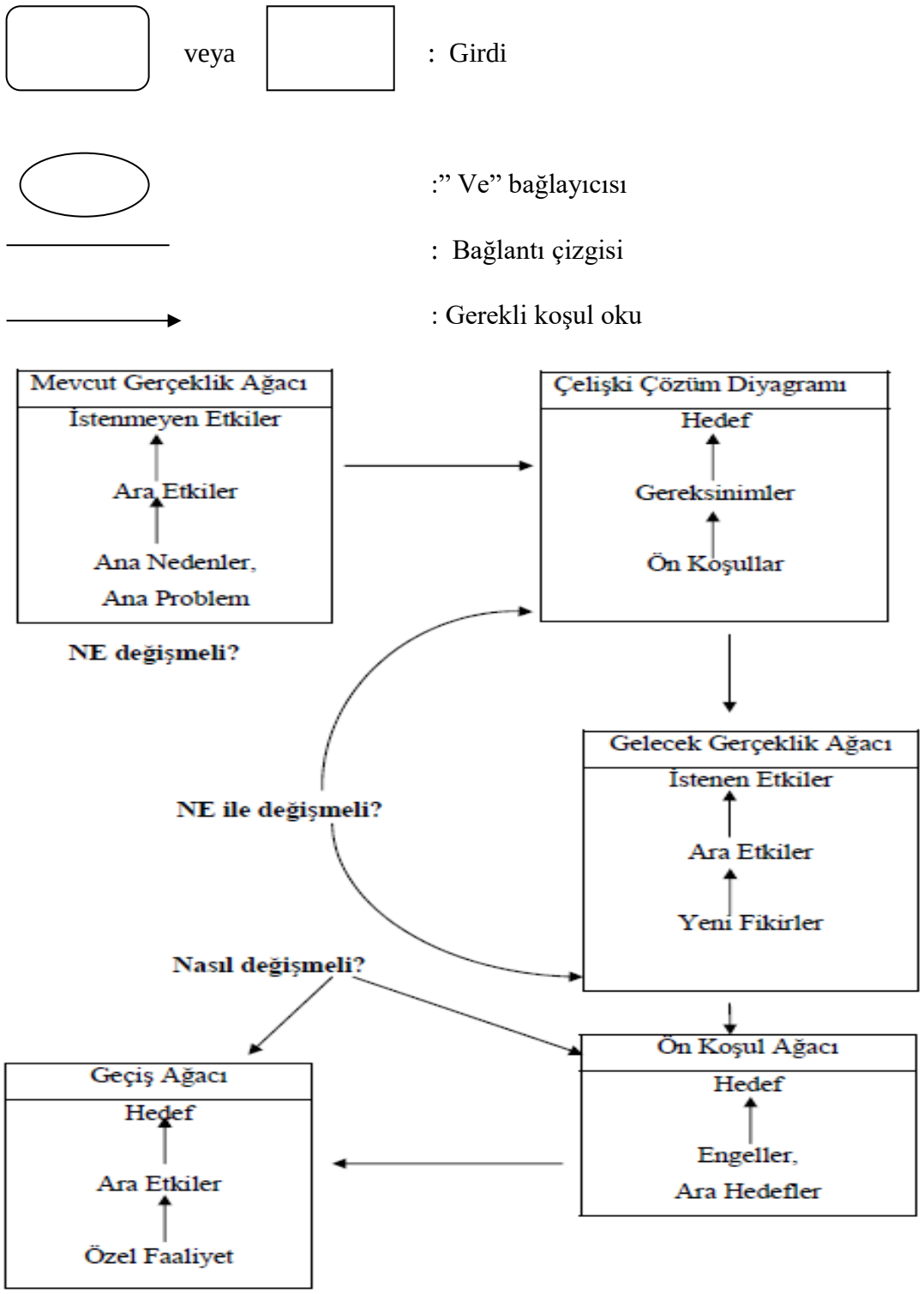
- İstenmeyen etkileri engellemek,
- Stratejik planlar için taktik faaliyet planları geliştirmek,
- Sınırlı bir amaca karşı, süreçte ortaya çıkan sapmayı belirlemek,

Geçiş Ağacı karar süreci sonunda uygulama sırasında tüm basamakları adım adım gösterir. Kısıtın nasıl giderileceği hakkındaki kararlara yardımcı olur.



ÖF: Özel Faaliyet, İ: İhtiyaç, E: Etki
Şekil 5: Geçiş Ağacı
Kaynak: (Dettmer, 1997)

Ağaç diyagramlarında kullanılan semboller şu şekilde ifade edilmektedir;



Şekil 6: Bütünleşik Düşünce Süreci Olarak 5 Mantık Aracı Arasındaki İlişki
Kaynak: (Dettmer 1997).

İKİNCİ BÖLÜM

YAZILIM GELİŞTİRME SÜREÇLERİ

2.1. Yazılım Geliştirme Süreçlerinin Tanımı ve Özellikleri

Bilgisayar donanım bileşenleri dışında kalan her şeye yazılım denir. Mantık, veri, belge, insan ve program bileşenlerinin belirli bir üretim amacına yönelik olarak bir araya getirilmesi ve yönetilebilmesi için kullanılabilecek ve üretilen, yöntem, araç, bilgi ve belgelerin tümünü yazılım içerir.

Yazılım	Mantık - (Algoritma)
	Veri - (Test verisi, Bilgi)
	Belge - (Dokümanlar)
	İnsan - (Kullanıcı, Geliştirici)
	Program - (Kodlama)

Şekil 7: Yazılımın Unsurları

Mantık: Bilgisayarlaştırılmak istenen iş için mevcut mantığın yazılıma yansıtılması durumudur. Yazılım haricinde günlük hayatımızda bile çeşitli algoritmalar kurmak zorundayız, örneğin; okula gitmek için ilk olarak uyanmak, evden dışarı çıkmak daha sonra ne şekilde ulaşım sağlanacak, gibi sorulara cevaplar üretmek suretiyle sıralı algoritmalar oluştururuz. Günlük hayatımızda nasıl bu algoritmaları kurmadan işlerimizi yürütemiyorsak, yazılım geliştirilirken de mantık (algoritma) kurumadan yazılımın işlevselliği etkin kılmak mümkün olamamaktadır.

Veri: Her tür yazılım mutlaka bir veri üzerinde çalışmak durumundadır. Veri dış ortamdan alınabileceği gibi, yazılım içerisinde de üretilebilir. Yazılımın temel amacı "veri"yi "bilgi "ye dönüştürmektir. Her şey veri olarak düşünülebilir resim, ses, yazı, rakam, amaç bunların hepsinden bilgi elde edebilmektir. Veri boyutlarına ve hafızada kapladığı yere göre kategorize edilerek düzenlenir. Bu düzenleme hafızanın etkin kullanılmasında önem arz etmektir.

Belge: Yazılım geliştirmek bir mühendislik disiplini gerektirmektedir. Mühendislik çalışmalarında izlenen yol ya da kullanılan yaklaşımlar yazılım üretimi için de geçerlidir. Yazılım üretimi sırasında, birçok aşamada yapılan ara üretimlere ait bilgiler (planlama, analiz, tasarım, gerçekleştirim, vb. bilgileri) belli bir düzende belgelenmelidirler. Yazılımın kullanma kılavuzu olarak tabir edilebilecek belgeleme safhası, yazılımı geliştiren mühendislere yol göstererek yapılan hataların tespitinin kolaylaşmasını sağlamaktadır.

İnsan (Kullanıcı, Geliştirici): İnsan faktörü yazılım parçalarında iki boyutlu olarak karşımıza çıkmaktadır. Yazılımı geliştirenler ve yazılımı kullanalar. Günümüz koşullarında artık tek kişi olarak yazılım geliştirmek mümkün olamamaktadır. Komplike yazılımları üretebilmek için bir ekip oluşturup çeşitli teknikler kullanmak gerekmektedir. Kurulan ekipte bir yönetici önderliğinde koordinasyon kurularak yazılımlar geliştirilmektedir.

Program (Kod): Yazılımın ana çıktısı olan bilgisayar kodları çeşitli programlama dilleri (C+, C++, C#, JavaScript vb.) kullanılarak yazılırlar. Yazılım kurulduktan sonra yeni ihtiyaçlar veya ortaya çıkan hatalar nedeniyle sürekli olarak bakıma ihtiyaç duymaktadır. Temel olarak iki nedenden oluşan bu durumlar şunlardır;

- Hiçbir program tümüyle her olasılık göz önüne alınarak test edilemez.
- İşletmeler dinamik bir yapıya sahip oldukları için sürekli yeni ihtiyaçlar ortaya çıkarabilmektedir.

Tipik bir yazılım projesinin geliştirilmesi ortalama 1 ila 2 yıl zaman alıyor ve en az 500.000 kod satırı içeriyor. Bu yazılım projelerinin %75 i başarı ile tamamlanabiliyor. Her çalışan günde 10 satırdan daha az kod yazımı gerçekleştirebiliyor. Yazılımın geliştirilmesi sürecinde her 1000 kod satırında 50 ila 60 hata olasılığı bulunuyor. Tamamlanmış ve satışa sunulmuş olan yazılımlar da bile 4/1000 hata olasılığı bulunuyor. Teslimi planlanan zamanın gerisinde kalma, bütçeyi aşma, yazılımın kalitesinin düşük olması (Güvenilir olmayan yazılım, kullanıcı ihtiyaçlarının karşılanmasında yetersiz kalınması, sürekliliğin sağlanamaması, vs.) gibi durumların oluşması da yazlım krizlerinin en önemli unsurları olarak göze çarpıyor.

Bir programı tüm ayrıntıları ile test etmek teorik olarak mümkün olmakla birlikte, uygulamada bu mümkün değildir. Yazılım ancak sınırlı sayıda veri ile sınanabilir. Sonuç olarak, işletmeye alınan ve üretimi tamamlanmış her yazılım %100 hatasız çalışıyor anlamına gelmez. Her zaman bir hatanın ortaya çıkma olasılığı vardır.

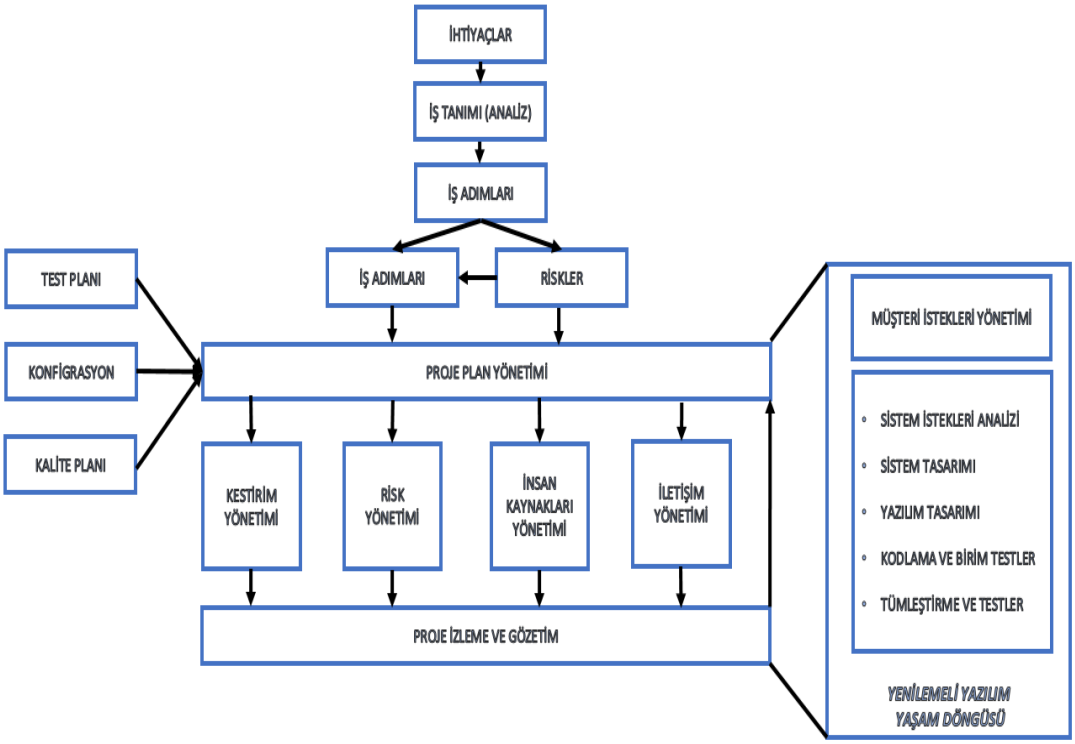
MANTIKSAL TASARIM HATA PAYI	20%
İŞLEVSEL TASARIM HATA PAYI	15%
KODLAMA HATA PAYI	30%
BELGELEME VE DİGER HATALAR PAYI	35%

Şekil 8: Yazılım Hata Oranları Tablosu

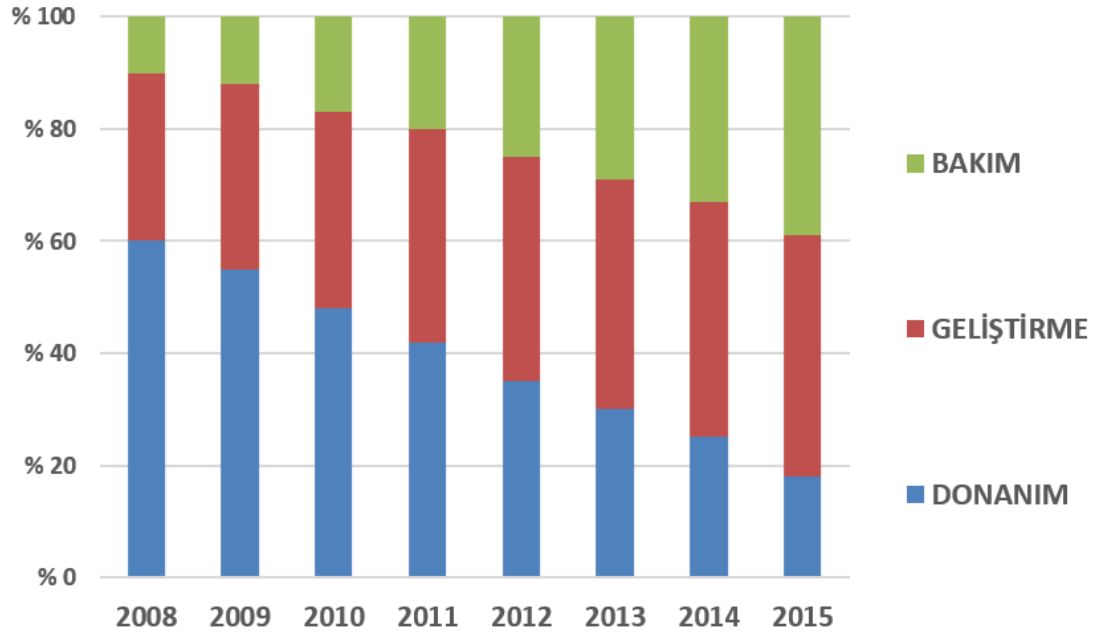
Tipik bir yazılım geliştirme sürecinde aşağıdaki gibi bazı olumsuzluklar kolayca gözlemlenebilir. Bir bakıma bu tür olumsuzlukların giderilmesi amacıyla Yazılım Mühendisliği disiplini geliştirilmiştir.

- Farklı yeteneklere sahip birçok personel (analist, programcı, test uzmanı, vs.),
- Yazılım çıktısı ile ilgilenen kullanıcılar,

- Yeniliğe tepki gösteren kullanıcılar ve yöneticiler,
- Yeterince tanımlanmamış kullanıcı beklentileri
- Personel değişim oranının yüksekliği,
- Yüksek eğitim maliyetleri,
- Dışsal ve içsel kısıtlar (zaman, maliyet, işgücü, vs),
- Standart ve yöntem eksiklikleri,
- Verimsiz kaynak kullanımı,
- Mevcut yazılımlardaki kalitesizlik,
- Yüksek üretim maliyeti,



Şekil 9: Yazılım Geliştirme Süreci Akış Şeması



Şekil 10: Yazılım Geliştirme Sistem Harcamaları Grafiği

Yazılım, belirli görevler için tasarlanmış makinaları, işlevsel hale getirebilmek, birbirleriyle haberleşmelerini ve uyumunu sağlayabilmek için oluşturulan elektronik komutlar bütünüdür. Başka bir ifadeyle ortaya çıkan problemleri çözmek amacıyla makine dilinde oluşturulan anlamlı ifadelerdir. İşletim sistemi yazılımları ve uygulama yazılımları olarak ikiye ayrılabilecek olan yazılımların işletim sistemi olarak tasarlanmış olanları bilgisayar donanımlarının bütünlük içerisinde çalışabilmelerini sağlarken uygulama yazılımları ise belirli amaçlara yönelik olarak tasarlanmaktırlar (Steinmuller, 1995, s. 1).

Yazılım sektörü, hizmet ve ürün sektörü arasında geçişi sağlamaktadır. Yazılım firmaları teknolojik firmalar olduğu için kazandıkları teknolojik bilgi ve tecrübeleri teknik çözümlerde kullanmaktadırlar. Bu doğrultuda yazılım firmalarının kapasiteleri ve bilgi birikimleri yazılım geliştirme sürecinde çok önemlidir. (Igel ve Islam, 2001, s. 157-166). Yazılım dünyasında, yazılım yaşam döngüsü genel olarak haftalar ve aylar la ölçülür. Kısa yazılım ömürleri kısa firma ömürlerin beraberinde getirir (Nambisan, 2002, s. 141-165). Yazılım sektörünün bu özellikleri göz önüne alındığında yazılım firmalarının ayakta kalabilmeleri için sürekli yeni yazılım

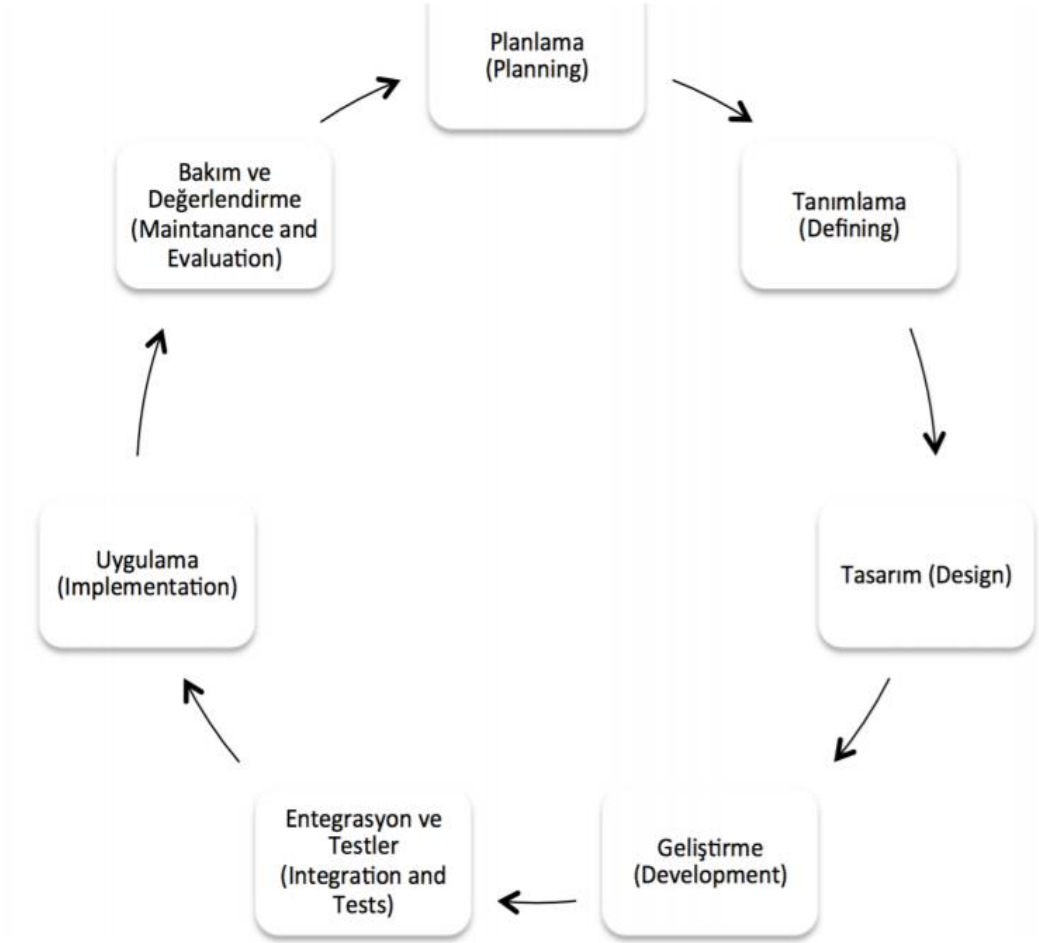
geliştirmeleri gerekmektedir (Igel ve Islam, 2001, s. 157-166). Yazılım geliştirme sürecinde ihtiyaçların yazılıma dönüşebilmesi için yapılması gereken çalışmaların tanımlanması gerekmektedir (Schwalbe, 2000, s. 52-60İ; Boutquin, 2000, s. 15-22).

Yazılım geliştirme süreci tekrarlayan bir süreç değildir. Yazılım geliştirilirken yazılım ekibinin kelitesi ürünün kalitesini oldukça etkilemektedir. Yazılım geliştirilirken sonraki aşama önceki aşamaya bağımlıdır. (Büyüközkan ve Feyzioğlu, 2003, s. 27-45).

2.2. Yazılım Geliştirme Yaşam Döngüsü

Yazılımın planlanması, geliştirilmesi ve kullanılması süreci boyunca geçtiği tüm aşamalar yazılım yaşam döngüsü olarak tanımlanmaktadır. Yazılımdan istenen işlevsellik sürekli değiştiği için aşamalar da sürekli bir döngü oluşturur. Bu döngüleri içinde geriye dönüşlerle tekrar ilerlemek mümkündür. Yazılım yaşam döngüsü doğrusal ve tek yönlü değildir. Bu nedenle sürekli tekrarlanan bir olaydır. Yazılımın kullanım ömrü dolduğunda bu yaşam döngüsü de sonlanmış olur. Yazılımın kullanım ömrünün dolması ise o yazılımın hem donanımsal hem de kodsal anlamda isteklere cevap verememesi anlamına gelmektedir. Bakım ve onarım işlemlerinden çok yazılımın yeniden revize edilmesi gerekliliği doğduğunda da yazılımın kullanım ömrünün dolduğu söylenebilir.

Yeni geliştirilen yazılımların yaşam döngüleri de yeni başlamış olmaktadır. Kimi zaman yazılımların yaşam döngüleri planlı bir şekilde belirlenmektedir, sadece ihtiyaca yönelik yazılımlar geliştirilerek belirli bir süre kullanılmak üzere planlanabilirler. Bunlar genelde tek kullanımlık yazılımlar olurlar ve bir programı çalıştırmak için kullanılırlar.



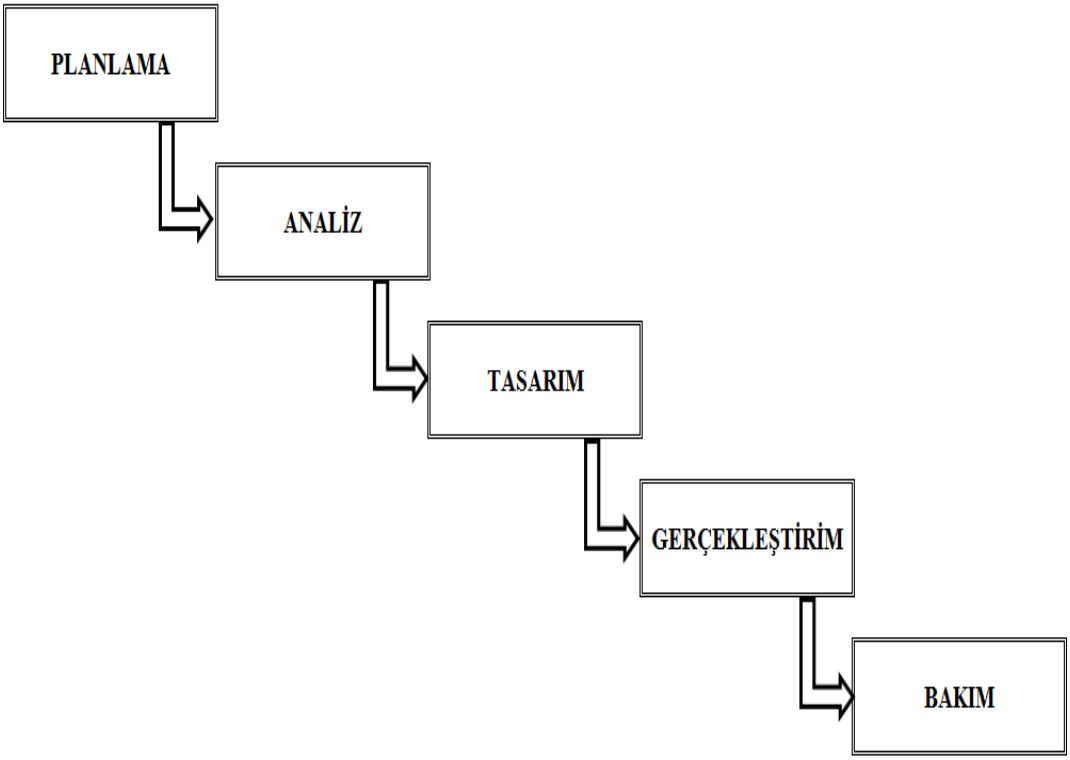
Şekil 11: Yazılım Geliştirme Yaşam Döngüsü.
Kaynak: (Seker, 2015, s. 20)

2.3. Yazılım Yaşam Döngüsü Temel Adımları

Yazılım geliştirme süreci sadece kod yazılımı olarak düşünülmemelidir. Yazılım geliştirilmeden önce belgeleme, test etme, kullanıcı eğitimi, bakım, idame gibi çalışmaların planlaması gerekmektedir (Balaban, 2004).

Şelale metodu bilinen en eski ve en yaygın yazılım geliştirme metotlarındandır. Bu metot ile yazılım geliştirilirken doğrusal bir çizelge izlenir.

Her aşama tek tek tamamlanarak geçilmek zorundadır geriye dönüş ile test yapılamaz (Balaban, 2004). Yazılım geliştirme süreci beş aşamalıdır. Bunlar: planlama, analiz, tasarım, gerçekleştirim, bakım.



Şekil 12: Yazılım Geliştirme Süreçleri
Kaynak: (Arifoğlu ve Doğru, 2000, s. 30)

2.3.1. Planlama

Bu aşamada yazılımın tüm gereksinimlerini kapsar. Kaynakları yönetimi, personelin yönetimi, müşterinin bilgi seviyesi ve yeni modelin kullanılabilirliği, ileriye dönük yazılımın kullanılabilirliği ve bakımı gibi durumlar bu aşamada tamamen incelenir. İncelemelerin bitiminde bir rapor hazırlanarak müşterilere sunulur ve yazılımın maliyetinin yapacağı işe değip değmeyeceği analiz edilir (Seker ve Diri, 2010, s. 881-887).

Örneğin: Bir yerleşim yeri kurulacak, inşaatlar yapılacak ve bu yer yaşayan bir yer olacak. Bu yerin içinde yeniler, eskiler, doğanlar ölenler olacak. Tüm bu döngünün organize edilişinin ilk aşamasının ve en önemli aşamasının planlama aşaması olduğu aşikârdır. Bütün yapının başlangıç aşamasıdır ve ihtiyacın belirlendiği fikirlerin tartışıldığı aşamadır (Seker, 2015, s. 19).

2.3.2. Analiz

Analiz aşaması projenin tüm ihtiyaçlarını ve ihtiyaçlara cevap verecek işlevlerin ayrıntılı olarak çalışıldığı aşamadır. Bu aşamada organizasyondaki ihtiyaçlar belirlenir, sorunlar ortaya çıkarılır ve yazılımın bu sorunlara nasıl çözüm getireceği tartışılır. Bu aşamada amaç yazılım mühendisi tarafından sorunların ve ihtiyaçların tam anlaşılmasını sağlamak ve yazılımın kusursuza yakın olmasını sağlamaktır (Akman ve Karakoç, 2005, s. 111).

Planlama aşamasının bitiminde yazılımın tanımlarının yapılacağı bu aşama ihtiyacın ve ihtiyaca yönelik istenilen çözümün ne olacağı ile ilgili tanımların konuşulduğu aşamadır. Eğer ki aynı düşüncede birleşilememiş ise ortaya çıkacak sonuçların da farklı olması kaçınılmazdır. Yapılacak bu tanımlamaların üzerine bir yazılım inşa edilecektir. Tasarım aşaması olarak da düşünülebilir(Seker, 2015, s. 20).

2.3.3. Tasarım

Analiz aşamasının bitiminde ortaya çıkan ihtiyaçlara çözüm getirecek yazılımın işlevleri tasarlanmaya başlanır. Bu aşamada yazılımın bütün modüllerinin genel hatları tasarlanır (Seker, 2014, s 4). Mantıksal tasarım ve fiziksel tasarım olarak iki aşama vardır. Mantıksal tasarımda teklif edilen sistemin şekli anlatılır. Fiziksel tasarımda ise yazılımın bölümleri ve detayları tartışılır (Akman ve Karakoç, 2005, s. 112).

Planlama ve analize göre bir proje hazırlanır. Belirli kararlar verilir ve bu kararlara doğru orantılı seçimler yapılır. Gereken tüm donanım ekipmanları belirlenir. Sayfalar arası geçişler ya da modüllerin nasıl olması gerektiği kullanılacak veri tabanı gibi unsurlar belirlenir. Gerçekleştirim aşamasına bu bölümden kalan bir karar aktarılmaması gerekir (Seker, 2015, s. 20).

2.3.4. Gerçekleştirim

Gerçekleştirim yazılım aşamalarından kodlama kısmına geçiş aşamasıdır. Artık yazılımın işlevsel özelliklerinin ilk adımları atılmaya başlanmıştır. Tasarım aşamasında planlanan her şey burada hayata geçirilmektedir (Seker, 2015, s. 20).

Kodların yazılması, modüllerin test edilmesi ve kurulumun yapıldığı aşamadır (Atasoy, 2018, s.1). Kodlama kısmı yazılım geliştirme süreçlerinin en fazla teknik bilgi gerektiren kısmıdır. Bu kısımda yazılım personelinin yetenekleri ön planda olmaktadır. Donanıma verilecek komutların donanımı en az seviyede yoracak şekilde kodlanması yazılımın daha istikrarlı çalışmasını sağlayarak donanımın daha uzun ömürlü olmasını sağlamaktadır. Yazılım personelinin tecrübesi ve bilgisi ne kadar yüksek ise oluşturulan kodlar bilgisayar donanımın en yüksek verimlilikte kullanılmasını sağlar.

2.3.5. Bakım

Bakım aşaması yazılım faal olduktan sonra bu yazılımın üzerinde değişiklikler yapılması olarak adlandırıla bilir (Soylu, 2017, s.6). Yazılımın müşteriye teslimi aşamasıdır. Güncelleme ve desteğinin verilmesidir (Koç, 2017, s. 13). Yazılımın bakımı esnasında yeni tasarımlar ve tanımlar ortaya çıkmakta; bu durum bir döngü şeklinde sürekli devam etmektedir. Böylece yazılımın yaşam döngüsü oluşmaktadır. Yazılım da bir tüketim ürünüdür ve belirli ihtiyaçları karşılayana kadar yaşamları devam eder. Bununla birlikte yazılım bir mühendislik

ürünüdür ve uzun ömürlü olması beklenir. Yazılımın hiç müdahale olmadan ya da küçük müdahalelerle uzun süreler var olması beklenir. Yazılım Yaşam Döngüsü mühendislik ürünü ve tüketim ürünü arasında yer almaktadır. Yazılımın bir planlanması ve yönetimi vardır (Seker, 2015, s. 20). Yazar yazılımın uzun ömürlü olmasının istendiğini, ancak bunun için sürekli bakımlarla düzeltmelerinin yapılması gerektiğini belirtmektedir. Aynı yazara göre Yazılım Yaşam Döngüsü bu bakımların nasıl ve nerede yapılacağına kılavuzluk etmektedir. Yazılımcı belli aşamalardan geçtikçe kazandığı tecrübeleri diğer aşamalarda kullanmaya başlar.

2.4. Yazılım Geliştirme Süreci Modelleri

İdeal bir yazılım süreci olmadığından yazılım geliştiren her firma kendi yaklaşımlarını geliştirmektedirler. Yöntemleri yazılımı kullanacak insanların yeteneklerine ve yazılımdan istenen özelliklere göre yöntemleri geliştirilmektedir. Dolayısıyla aynı organizasyonda olsa bile farklı yazılımlar için farklı yöntemleri kullanılabilir (Sommervile, 2000, s. 45-63).

Yazılım oluşturulurken seçilecek model sürecin en önemli kararların başında gelir (Kettunen ve Laanti, 2005, s. 587-608). “Bir sürecin olması hiç olmamasından çok daha iyidir ve çoğu durumda, bir sürecin varlığı nasıl uygulandığından daha fazla önem arz etmektedir” (Natarajan, 2004, s. 20-30). Doğru yöntemin seçilmesi problemlerin önüne erkenden geçileceği anlamına gelir (Kettunen ve Laanti, 2005, s. 587-608).

Süreçler farklı olsa bile hepsinde ortak etkinlikler vardır (Sommervile, 2000). Bunlar:

- Projenin özelliklerinin belirlenmesi ve kısıtlarının saptanması
- Yazılımın tasarımı ve kodlanması
- Yazılımın test edilmesi ve ihtiyaçları karşılayıp karşılamadığının belirlenmesi
- Yazılımın yeni ihtiyaçlara göre göncenlenmesi ve çıkan sorunların giderilmesi için bakımının yapılması.

Geri dönüş ve izleniş sırasına göre farklı yazılım yaşam döngüsü modelleri geliştirilmiştir (Kettunen ve Laanti, 2005, s. 587-608).

Bunlar:

- Gelişi Güzel (Disiplinsz) Model.
- Şelale (Çağlayan) Modeli.
- Barok Modeli.
- V Modeli.
- Helezonik (Spiral) Model.
- Artırımsal Model.
- Evrimsel Model.

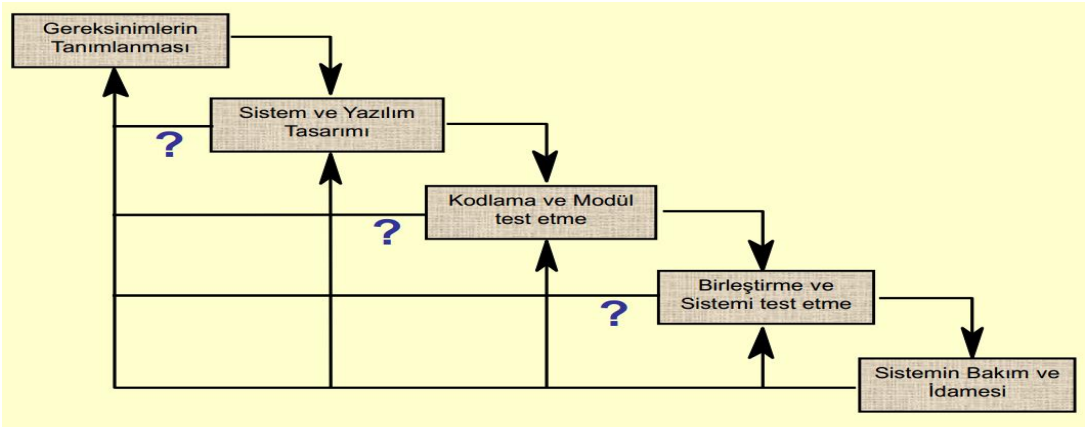
2.4.1. Gelişi Güzel (Disiplinsiz) Model

Herhangi bir düzen ya da yöntem olmadan, geliştiren kişiye bağımlı olarak oluşan bu model, çoğu zaman geliştiren kişinin dahi yazdığı kodların karmaşıklığı içinde kaybolmasına neden olmaktadır. 1960'lı yıllarda kullanılan bu model günümüzde kullanılabilirliğini yitirmiştir. Genellikle tek kişilik üretimden oluşan bu yöntemin en basit programlama yöntemi olduğu söylenebilir. İzlenmesi ve bakımı oldukça zordur.

Bu yöntem genel olarak düzensiz ve plansızdır. Tahminler ve zamanlama gibi planlamalar yapılsa bile uygulamada çok az kullanılırlar (Karadağ, 2010, s. 2).

2.4.2. Şelale (Çağlayan) Modeli

Bu yöntem statiktir (Soylu, 2017, s.11). Yazılım geliştirme süreçlerinin ilkidir. Her aşama tek tek çalışılır ve uygulanır.



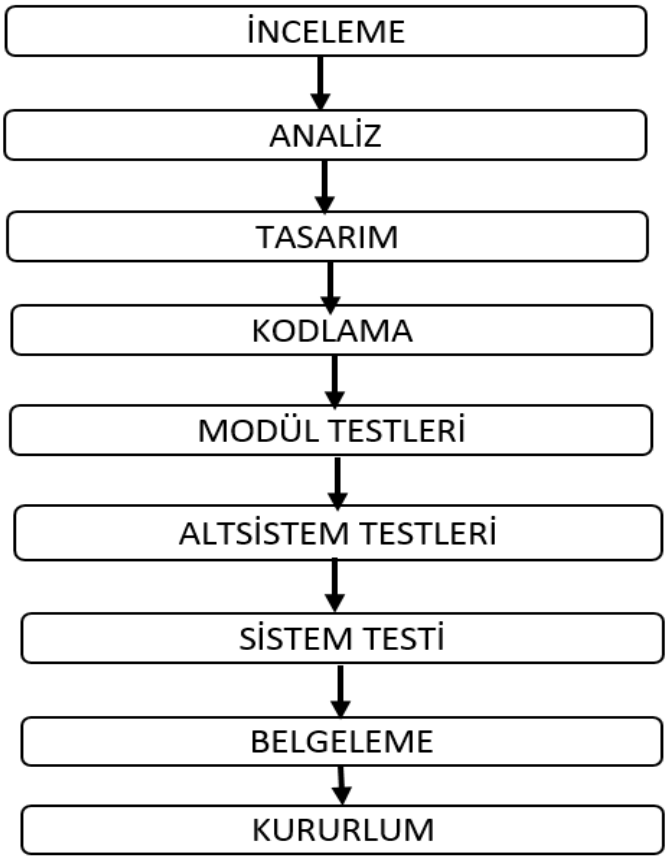
Şekil 13: Şelale (Çağlayan) Modeli
Kaynak: (Seker, 2015, s. 19)

Her aşama gözden geçirilerek onaylanır. Süreç bu şekilde devam etmektedir. Bir aşama bitmeden sonraki aşama ya geçilmez. Gereksinim bölümünün bitip tasarım aşamasına geçildiğinde ihtiyaçlar belirlenmiş ve sabitlenmiş kabul edilir. Belgeleme odak noktası olarak gelişmiştir (Pauca, 2003, s. 331-361). Bu yöntemde belgeleme yapmak, tekrar işler çok belirgin olduğundan maliyeti yüksek olmaktadır. Sorunlar ertelenerek en sona bırakılır. Dolayısıyla gereksinimlere cevap vermesi oldukça zor olmaktadır. Bu yöntemin uygun olduğu koşullar gereksinimler tam anlaşıldığı koşullardır. Aksi takdirde projeyi bu yöntemde olduğu gibi esnek olmayan şekilde tasarlamak doğru olmamaktadır. Büyük projeler ve mühendislik disiplini gerektiren işlerde kullanılması uygun olmaktadır (Sommervile, 2000, s. 45-63).

Yaşam döngüsünün temel adımları en az bir kez izleyerek gerçekleştirilir. İyi tanımlı projeler ve üretimi az zaman gerektiren yazılım projeleri için uygun bir modeldir. Geleneksel model olarak da bilinen bu modelin kullanımı günümüzde giderek azalmaktadır. Barok modelin aksine belgeleme işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür. Barok modele göre geri dönüşler iyi tanımlanmıştır. Yazılım tanımlamada belirsizlik yok (ya da az) ise ve yazılım üretimi çok zaman almayacak ise uygun bir süreç modelidir.

2.4.3. Barok Modeli

Yaşam döngüsünün adımlarının doğrusal şekilde ilerlediği bu model 1970'li yollarda kullanılmaya başlanmıştır. Belgelemeyi ayrı bir süreç olarak ele alır, belgelemenin yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür. Günümüzde belgeleme işlemleri yazılımın doğal bir ürünü olarak görülmektedir. Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değil. Gerçekleştirim aşamasına daha fazla ağırlık veren bir model olup, günümüzde kullanımı önerilmemektedir (Yılmaz, 2007, s. 9).



Şekil 14: Barok Modeli
Kaynak: (Seker, 2015, s. 20)

2.4.4. V Modeli

Belirsizliklerin az iş tanımlarının fazla olduğu bir model olan V Modeli kullanıcının projeye katkısını arttırmaktadır. Projenin iki aşamalı olarak ihale edilmesini sağlamaktadır (Yılmaz, 2007, s. 15). Bunlar:

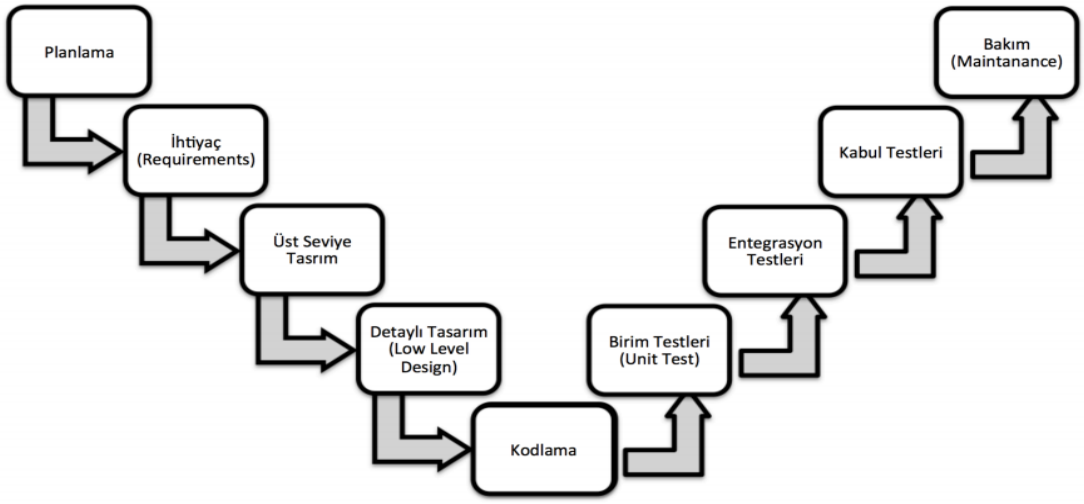
- İlk ihalede kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta,
- İkinci ihalede ise ilkinde elde edilmiş olan kullanıcı modeli tasarlanıp, gerçeklemektedir.

Kullanıcı modeli: Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.

Mimari Modeli: Sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevler.

Gerçekleştirim modeli: Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar.

Sol taraf üretim, sağ taraf sınama işlemleridir.



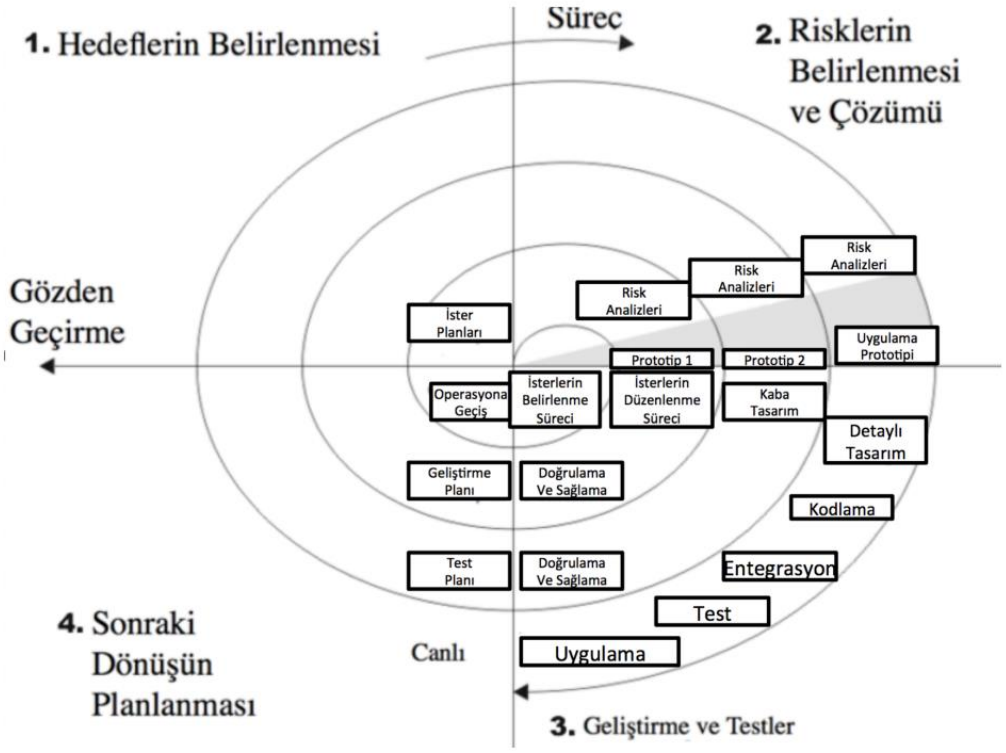
Şekil 15: V Modeli
Kaynak: (Seker: 2015, s. 21)

2.4.5. Helezonik (Spiral) Model

Bu yöntem riskler üzerine odaklanarak ortadan kaldırılmasını çabalar. Risk yönetimin kullanması bu yöntemin farkını ortaya koyar. Modelin oluşumunu sağlayan döngüleri bir takım aşamalardan geçerler (Sommervile, 2000, s. 45-63). Bunlar:

- Risklerin belirlenmesi.
- Bir sonraki aşamanın planlanması.
- Gelişim ve geçerliliğin sağlanması.
- Amaç ve sınırların belirlenmesi.

Yenilemeli artırımsal bir yaklaşım sergileyen helezonik modelin prototip oluşturarak denemeler yapar Her döngü bir fazı ifade eder. Doğrudan tanımlama, tasarım,.vb. gibi bir faz yoktur. Risk Analizi Olgusu ön plana çıkmıştır (Yılmaz, 2007, s. 20).

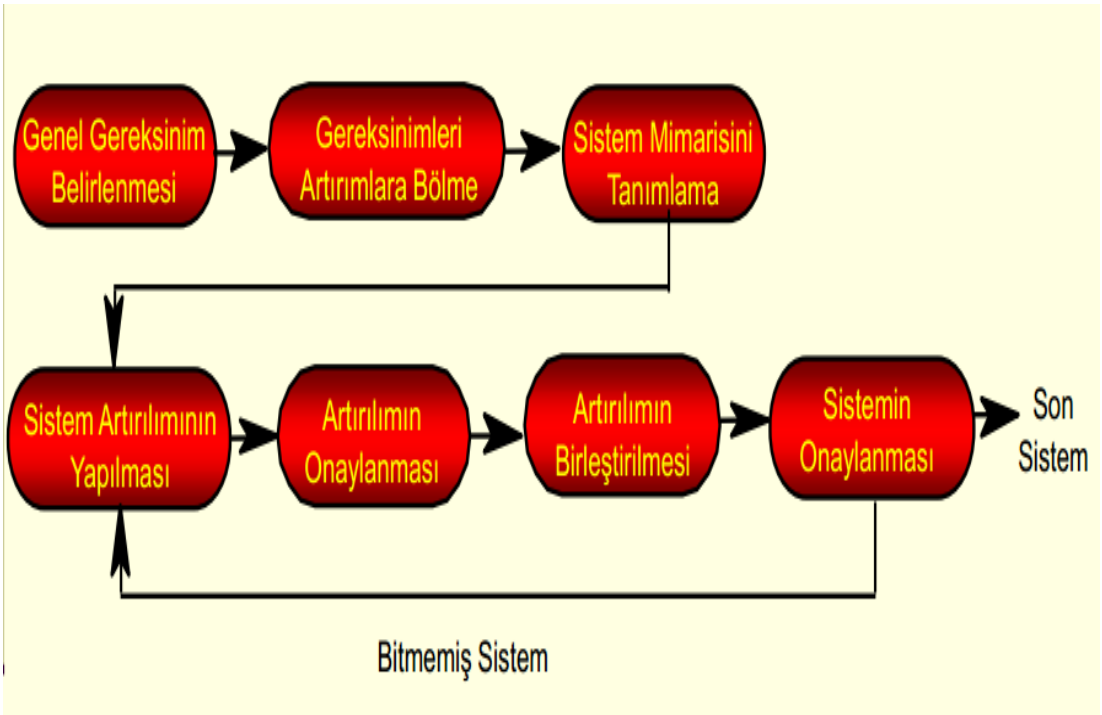


Şekil 16: Helezonik (Spiral) Model
Kaynak: (Seker: 2015, s. 21)

Üretim süreci boyunca, ara ürün üretme ve üretilen ara ürünün kullanıcı tarafından sınanması temeline dayanır. Yazılımı kullanacak personelin sürece erken katılması ileride oluşabilecek istenmeyen durumları engeller. Gerek proje sahibi, gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme ve hak ediş planlaması yapılır. Yazılımı geliştiren mühendisler için yazılımın kodlanması ve sınanması daha erken başlar (Yılmaz, 2007, s. 19).

2.4.6. Artırımsal Model

Artırımsal modelde yazılımın tamamını bitirip müşteriye vermektense, yazılım fonksiyonlarına göre kategorize edilerek parça parça teslim edilmesi yaklaşımı mevcuttur.



Şekil 17: Artırımsal Model
Kaynak: (Seker: 2015, s. 19)

Müşterinin yazılımı kullanabilmesi için tüm projenin bitmesini beklemek zorunda kalmamaktadır. Önem sırasına göre ihtiyaçları belirleyerek ilk uygulama ve geliştirmeye bu sıradan başlanabilir. Prototipler kullanılarak tecrübe kazanılır ve sistem öğrenilir. Kullanıcı ihtiyaçlarını daha kolay belirleyerek, yazılım geliştirme sürecinin içinde yer alır. Aksaklıkların tüm yazılımı etkilemesi engellenerek risk en az seviyeye indirilir. Önemli olan ihtiyaçlara yönelik yazılım parçaları önceden teslim edilerek geriye kalan bölümler sonradan bütünleştirilir. Müşterinin temel ve en önemli bölümlerde hata bulma olasılığı oldukça düşük olmasının sebebi önemli bölümlerin en çok sınamadan geçilmiş olmasıdır (Sommervile, 2000, s. 45-63).

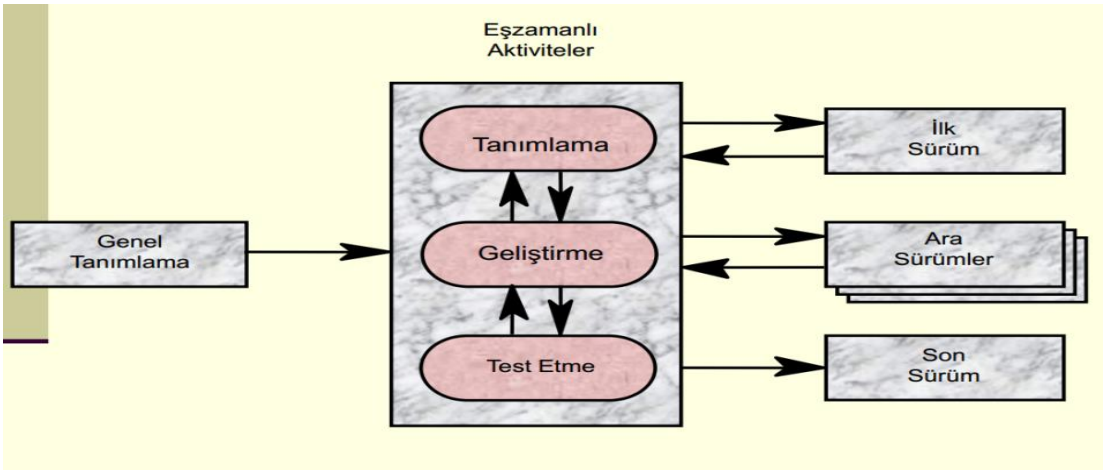
Üretilen her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler bu modele uygun olabilir. Bir taraftan kullanım, diğer taraftan üretim yapılır (Yılmaz, 2007, s. 25).

2.4.7. Evrimsel Model

İlk tam ölçekli model olarak öne çıkan bu model, coğrafik olarak geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir (banka uygulamaları). Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedirler. Pilot uygulama kullan, test et, güncelle diğer birimlere taşı olarak çalışılan bu modelin başarısı ilk evrimin başarısına bağımlıdır (Yılmaz, 2007, s. 22-23).

Örnek:

- Çok birimli banka uygulamaları
- Önce sistem geliştirilir ve Şube-1'e yüklenir.
- Daha sonra aksaklıklar giderilerek geliştirilen sistem Şube-2'ye yüklenir.
- Daha sonra geliştirilen sistem Şube-3'e yüklenir.
- Belirli aralıklarla eski şubelerdeki güncellemeler yapılır.



Şekil 18: Evrimsel Model
Kaynak: (Seker: 2015, s.19)

Birçok yazılım sürecinde bu yöntem kullanılmaktadır. Aşamalar tekrar edilerek yazılım sürekli güncellenmektedir. Yazılımın küçük bir bölümü oluşturularak aralıkları uzun olmayan sürelerle eklentileri monte edilmektedir. Müşteriler yazılımın durumunu görmek için sürekli raporlar istediğinde yazılım çabuk geliştirildiği için her parça için ayrı raporlar hazırlamak maliyetli ve zahmetli olmaktadır. Yazılımın devamlı değişime maruz kalması mimariyi bozarak birleştirmeyi zorlaştırmaktadır. Özel teknik ve araçlar gerektirdiğinden bu tekniklerin diğerleriyle uyumlu olmaması durumunda bu uyumu sağlayabilecek mühendis yeteneğine ihtiyaç duyulmaktadır (Sommervile, 2000, s. 45-63).

KÜÇÜK (KOD SATIR SAYISI <= 2.000)	BİLGİSAYAR OYUNLARI ÖĞRENCİ PROJELERİ
ORTA (KOD SATIR SAYISI 2.000 İLE 100.000 ARASI)	CAD (COMPUTER AİDED DESİGN) BDE (BİLGİSAYAR DESTEKLİ EĞİTİM) YAZILIMLARI
BÜYÜK (KOD SATIR SAYISI 100.000 İLE 1.000.000 ARASI)	İŞLETİM SİSTEMLERİ
ÇOK BÜYÜK (KOD SATIR SAYISI >= 1.000.000)	KOMUTA KONTROL SİSTEMLERİ HAVA TAHMİNİ SİSTEMLERİ YILDIZ SAVAŞLARI SİSTEMLERİ

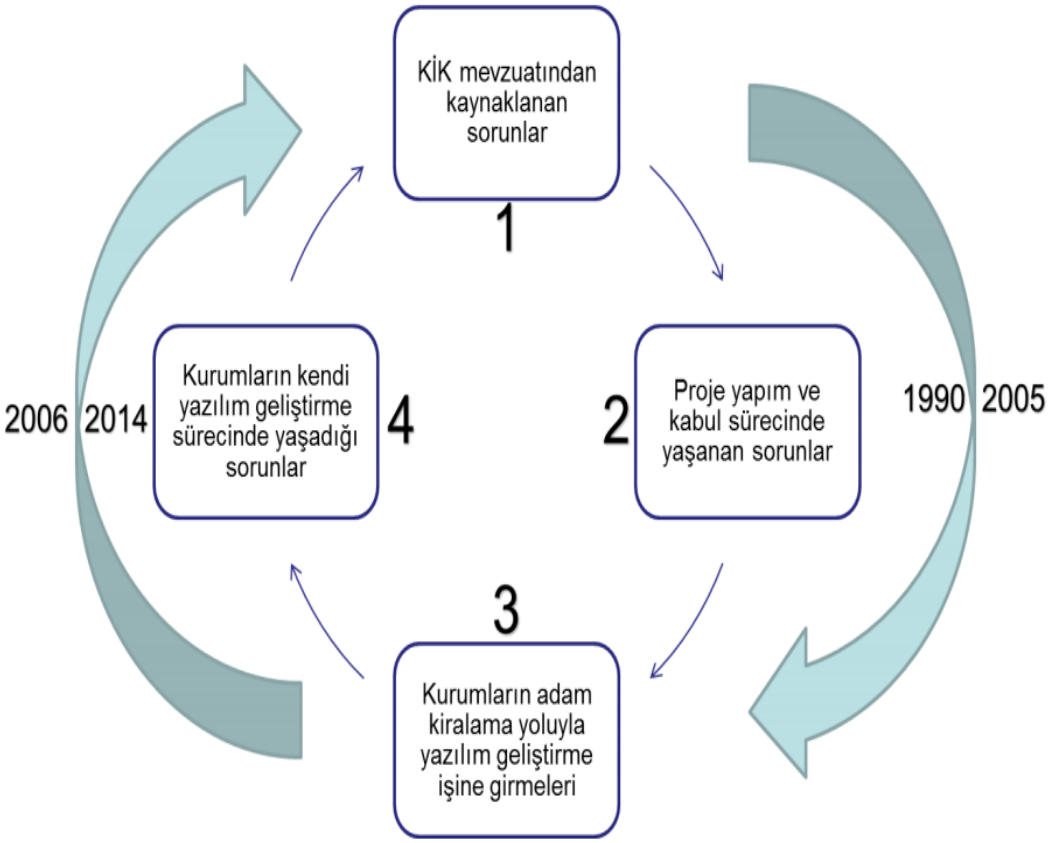
Şekil 19: Boyutlarına Göre Yazılım Sınıflandırmaları
Kaynak: (Sommervile, 2000, s. 51).

HESAPLAMA	MÜHENDİSLİK ÇÖZÜMLEME
VERİ İŞLEME	BANKACILIK
SÜREÇ TEMELLİ	GÖMÜLÜ SİSTEMLER
KURAL TEMELLİ	ROBOTİK, YAPAY ZEKA
CAD (COMPUTER AİDED DESİGN)	SİNYAL İŞLEME

Şekil 20: İşlevlerine Göre Yazılım Sınıflandırmaları
Kaynak: (Sommervile, 2000, s. 53).

2.5. Yazılım Geliştirme sürecinde karşılaşılan sorunlar

Yazılım geliştirilirken karşılaşılan sorunlar, sürekli değişen kullanıcıların istekleri sebebiyle yazılımın zamanında yetiştirilememesine neden olmaktadır. Yazılımcı personelin çok sık değişmesi, yetenekli (kaliteli) yazılımcı temininde güçlük çekilmesi önemli sorunlar arasında yer alır. Belgeleme konusunda zayıf kalınması, yazılımın uygun şartlarda ve yeterli kalitede test edilememesi, yazılımcı personelin çalıştığı parça üzerinde farklı teknikler kullanması, raporlama, test etme gibi işler için personel yetersizliği sorunlar arasında göze çarpmaktadır. En küçük rapor tasarımın da bile yazılımcı personel gereksinimi göze çarpmaktadır. Bakım aşamasına geçildiğinde ise; organizasyonda ki kullanıcı değişiklikleri, yeni gelen kullanıcıların eğitimleri ve yazılıma alışma süreleri sorunlar arasında gösterile bilir. Yazılımın kullanımının devamı için nitelikli personel bulunmaması, az sayıda olan nitelikli personelin görev ve sorumluluklarının fazla olması, işletme kültüründe bakım anlaşmalarının önemsenmemesi sebebiyle yazılımın kullanım ömrünün kısalması gibi sorunların oluştuğu da gözlemlenmektedir (Bank, 2014, s. 28-31).



Şekil 21: Yazılım Geliştirme Sorun Döngüsü
Kaynak: (Bank, 2014, s. 29

Yazılım sektörü içinde yer alan beş farklı firmadan toplanan bilgiler ışığında, yazılım geliştirme süreçlerinde yazılım projesinin tamamlanmasıyla birlikte müşteri kullanımına sunulduktan sonra problemlerin ortaya çıktığı tespit edilmiştir. Süreçler tetkik edildiğinde sorunların büyük çoğunluğunun geliştirme ihtiyaçlarının belirlenmesi safhasında ortaya çıkan aksaklıklardan meydana geldiği belirlenmiştir. Temel amaç oluşan problemi bulmak ve kısıtlar teorisi düşünce süreçlerini kullanarak ortadan kaldırmaktır (Akman ve Karakoç, 2005, s. 112) .

ÜÇÜNCÜ BÖLÜM

KISITLAR TEORİSİ YAKLAŞIMININ YAZILIM GELİŞTİRME SÜRECİNDE KULLANIMI

Bu bölümde kısıtlar teorisi yaklaşımının yazılım geliştirme sürecine etkisi incelenmektedir. Bu bölümde bir Büyükşehir belediyesi bütçesinin finansman kaynaklarının durumu incelenmekte ve tartışılmaktadır. Bu büyükşehir belediyesinin finansman kaynaklarının yatırımlara dönüştürülebilmesi için bir stratejinin belirlenmesi ve bu strateji yönetiminin bir yazılımla desteklenmesi gerektiği tespit edilerek yazılım geliştirme süreci başlatılmıştır. Aynı zamanda çeşitli firma ve işletmelerin benzer durumları da göz önüne alınarak kısıtlar teorisinin yazılım sürecine uygulanışıyla ilgili örnekler verilmektedir.

Kısıtlar teorisi düşünce süreçlerinin her bir aşaması birbirine sıkı sıkıya bağlıdır. Bir aşama tamamlanmadan diğer aşamaya geçilmesi kısıtların ortadan kaldırılmasını ya da iyileştirilmesini mümkün kılmaz. Her aşama titizlikle uygulanarak benimsenmeli ve yönetimin desteğiyle de tamamlanmalıdır. Aşamalar tamamlanarak uygulandığı takdirde kısıtların ortadan kaldırılması mümkün olacaktır. Kısıtlar teorisi düşünce süreçleri kısaca aşağıda belirtilmektedir.

Mevcut Gerçeklik Ağacı: Büyükşehir Belediyesinde ki yazılım geliştirme sürecinde ortaya çıkan temel problemler ortaya çıkarılır.

Buharlaşan Bulut: Ortaya çıkarılan problemler için Büyükşehir belediyesi bünyesinde çözüm yolları aranır**.**

Gelecekteki Gerçeklik Ağacı: Amaç buharlaşan buluttaki çözüm çalışmalarının, istenilen etkiye ulaşacağının doğrulanmasıdır.

Ön Gereksinim Ağacı: Geliştirilen çözümlere nasıl ulaşılacağının yolu aranır.

Geçiş Ağacı: Ayrıntılı şekilde planlanan amaçlara ulaşmak için gerekli uygulamaların tanımlanmasını hedefler. Bu nedenle iletişim, işbirliği ve yönetimin desteğini de alarak koordinasyon kurulmasıyla, hatasız bir yazılım üreterek Büyükşehir belediyesinin finansman kaynaklarının yönetilmesi mümkün olabilmektedir.

3.1. Mevcut Gerçeklik Ağacı' nın Kullanımı

Düşünce süreçlerinin başlangıcı, istenmeyen etkilerin tespit edilmesi ve Mevcut Gerçeklik Ağacı (MGA) kullanılarak şablonun oluşturulmasıdır. Mevcut Gerçeklik Ağacı sorunlar ve sonuçlar arasındaki neden sonuç ilişkisini kuran ilk adım diyagramıdır. Hedef ana problemin tespit edilmesini sağlayarak bu probleme odaklanılmasıdır. Problem tespit edildiğinde istenmeyen etkinin kaynağı da ortaya çıkmış olur, dolayısıyla problem ortadan kaldırıldığında istenmeyen etkinin de ortadan kalkacağı kesindir.

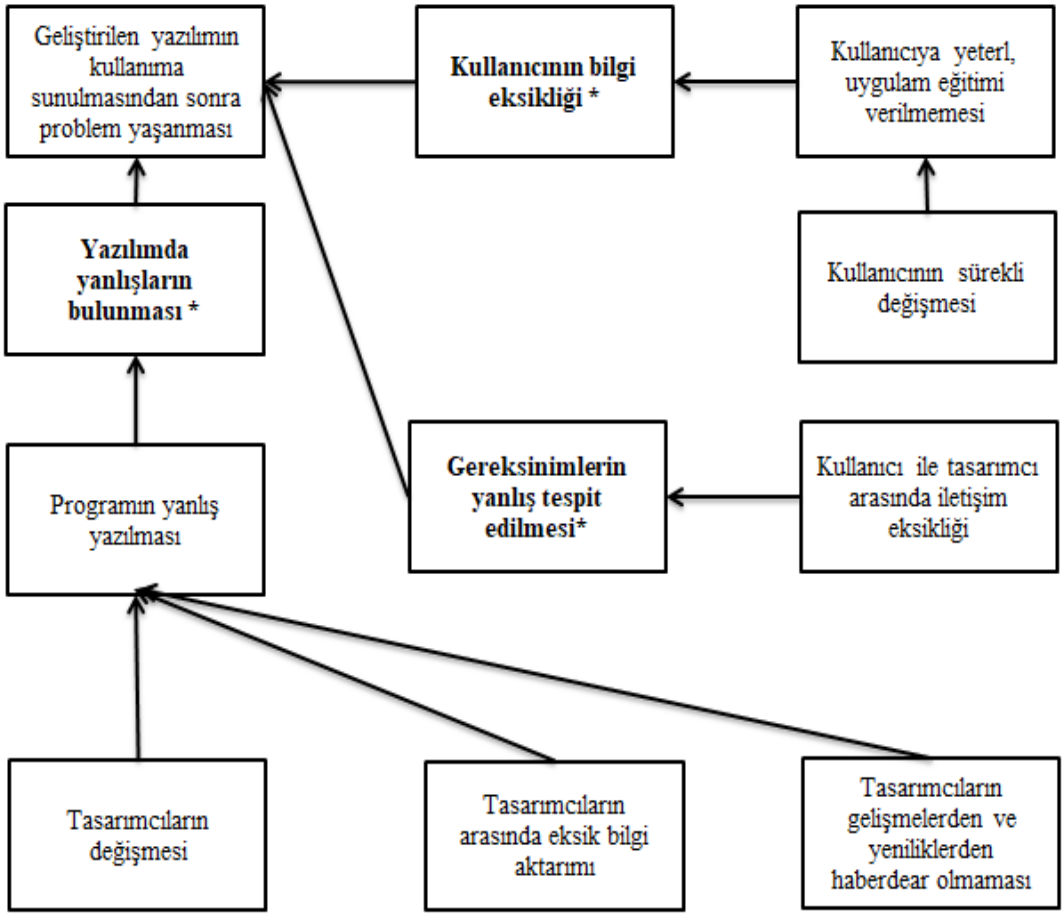
Örneğin büyükşehir belediyelerinde harcama bütçesinin oluşturulması sürecinde bakanlıktan gelen parasal payların yanı sıra belediyenin kendi elde ettiği gelirler göz önünde bulundurularak yatırımlar için kredi kullanım ihtiyacı tahmin edilmeye çalışılır. O an ki yatırımların ve büyükşehir belediyesinin mevcut borç durumu incelenir, ortaya çıkan senaryo değerlendirilir. Senaryonun sonunda ortaya çıkan takip ihtiyacı tespit edilerek yazılıma aktarılabilmesi için gereksinimler ortaya konulur. Takip ihtiyacının doğması ise planlanan durumun ne aşamada olduğunun tespit edilememesinden kaynaklanmaktadır. Bu da bir kısıt olarak göze çarpmaktadır.

Yazılım projelerinde ortaya çıkan teme problemler şu şekilde sıralanabilir (Akman ve Karakoç, 2005, s.113):

- Yazılımın kullanıcı ihtiyaçlarına tam anlamıyla cevap verememesi,
- İhtiyaçların tam anlamıyla tespit edilememesi,
- Kodlama kısmındaki hatalar,
- Yazlımın hazırlandığı çevre ile kullanıldığı çevrenin farklı olması,

- Yazılımı gerçekleştiren mühendislerin sık değişmesi
- Kullanıcıların bilgi eksikliği
- Kullanıcı eğitimlerinin yetersiz olması

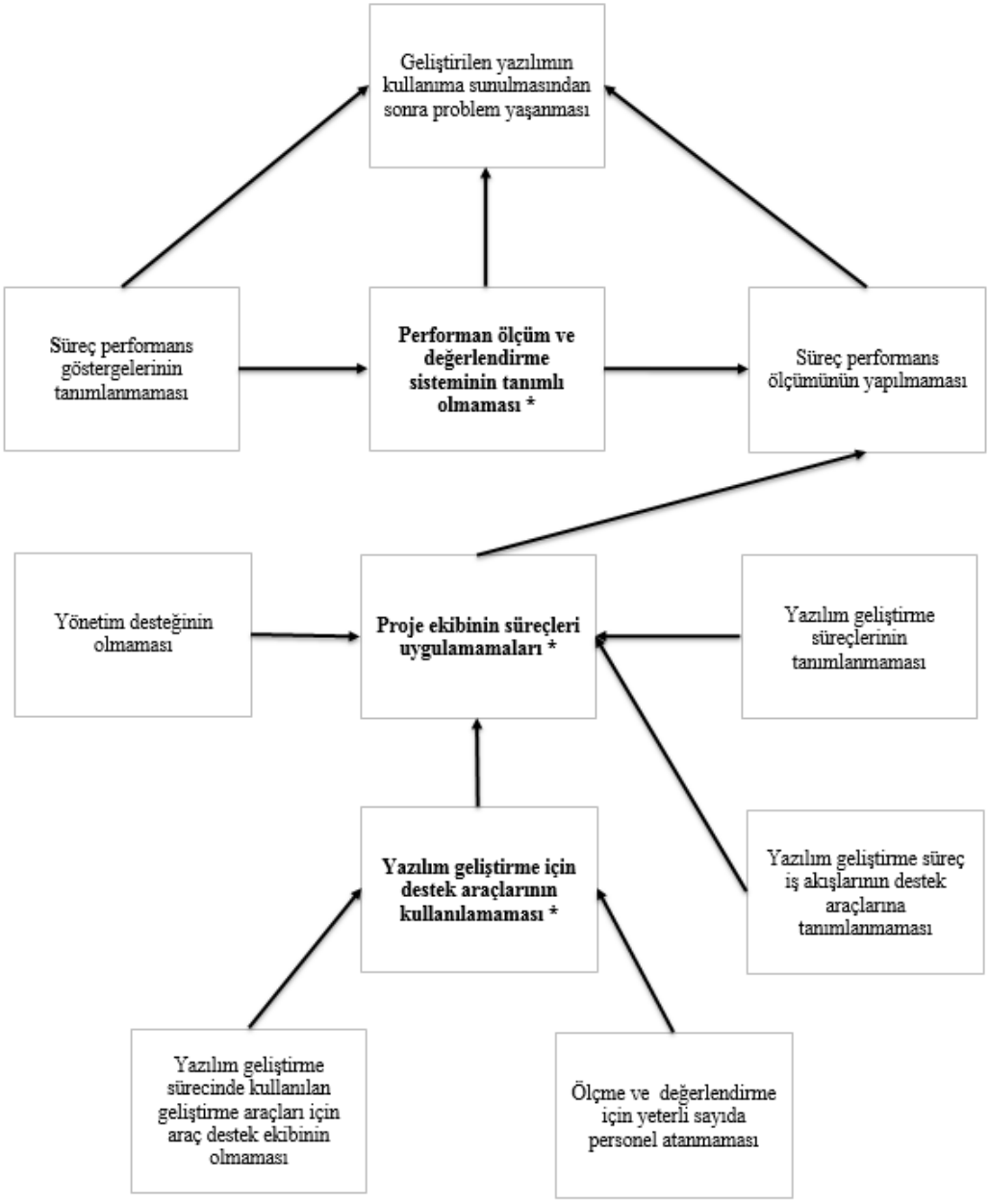
Şekil 22 ve Şekil 23'de oluşturulan Mevcut Gerçeklik Ağacı'nda yazlımınım kullanıma sunulmasından sonraki aşamada oluşan problemlerin temel nedenleri tespit edilmiştir. (*) işaretiyle gösterilmiş olan problemler ana kısıtı ortaya çıkarmış problemlerdir.



Şekil 22: Mevcut Gerçeklik Ağacı Uygulaması 1

Yazılım projesi süreçlerinin performans ölçümünde karşılaşılan diğer kısıtlar; proje geliştirme faaliyetlerinde destek araçlarının kullanılmaması, proje

çalışanlarının süreçleri uygulamamaları, performans ölçüm ve değerlendirme sürecinin tanımlı olmaması olarak sıralanabilir (Gün, 2015, s. 11).



Şekil 23: Mevcut Gerçeklik Ağacı Uygulaması 2

3.2. Buharlaşan Bulut'un kullanılması

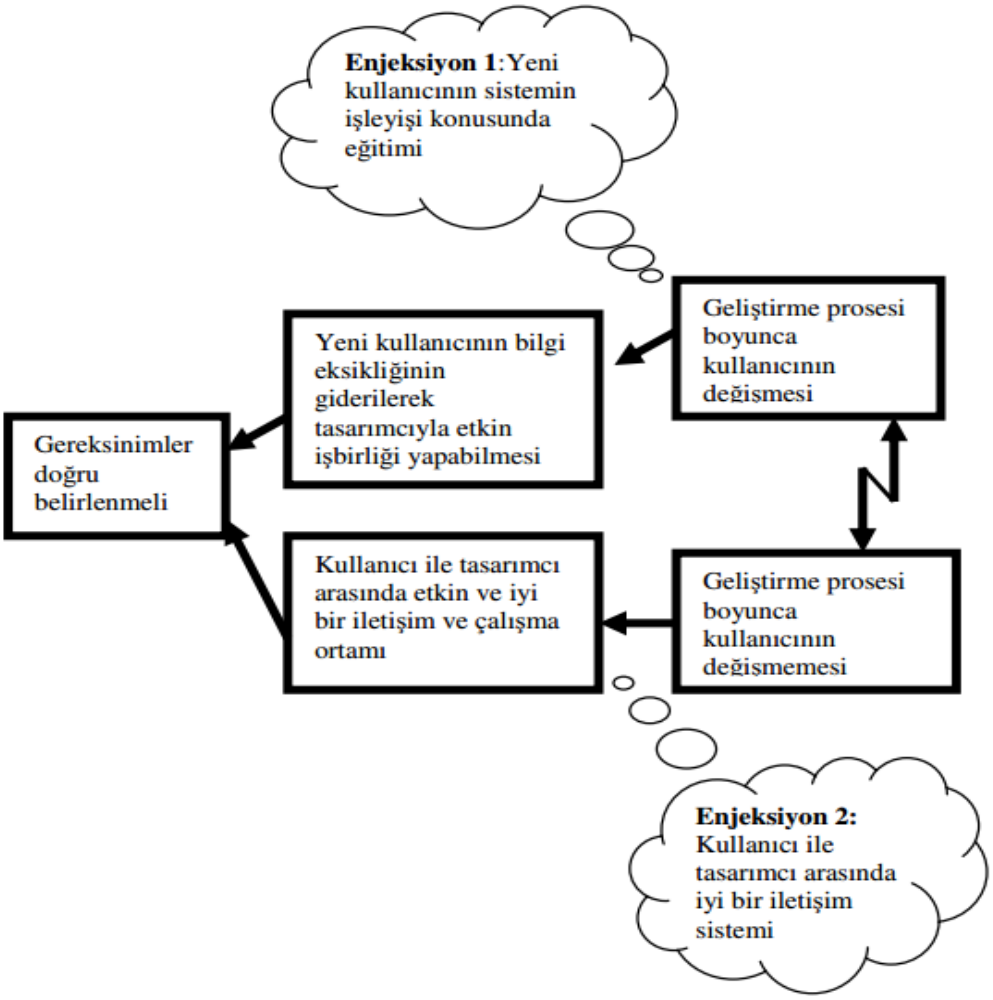
Kısıtlar teorisinin yaklaşımın ikinci adımı olan Buharlaşan Bulut tekniği, tespit edilen kısıtların ortaya çıkmasına neden olan sorunlar için çözümler üretmeye yöneliktir. Buharlaşan Bulut tekniğinde ortaya çıkan çatışmalar için bir takım enjeksiyonlar uygulanır. Şekil 24'te Buharlaşan Bulut yaklaşımıyla yazılım geliştirme projesinde ortaya çıkabilecek çatışmalardan bir tanesine uygulanabilecek enjeksiyonları ile çözüme ulaşılma yolu gösterilmektedir. Yaşanan çatışmanın içeriği yazılım geliştirme sürecince kullanıcıların sirkülasyonudur. Enjeksiyonların amacı etkili iletişim sistemi kurumak ve kullanıcıların yazılımı kullanırken eğitimlerine önem vermektir. Enjeksiyonların amacına ulaşması, bu kısıtın ortadan kalkmasını sağlamaktadır (Akman ve Karakoç, 2005, s. 114) .

Buharlaşan bulut tekniğiyle kısıtın ortadan kaldırılması için çareler üretilirken, en temel ve ön ihtiyaçlar tanımlanır. Ortaya atılan çözümler arasında ki çatışmaları belirleyerek, çatışmaların giderilmesi için enjeksiyonlar uygulanır (Özer, 2001, s. 7-29) .

Buharlaşan Bulut, problemlerin tek tek ele alınmasını, her problem için oluşan, çatışmalar ve hipotezler için çözümler üretilmesini içerir (Köksal ve Karşılıklı, 2000). Buharlaşan bulut sıkıntılı durumdan ideal duruma geçiş için kısıtın ortadan kaldırılmasına katkı sağlar ve diğer aşamalara geçişe bir bağ oluşturur. Temel kısıtlara etkili ve kesin çözümler üretilir. Bu yaklaşımda kaç tane sorun varsa o kadar buharlaşan bulut üretilmelidir. Buharlaşan bulut çıkan çatışmaları birleştirerek çatışmalarda ortaya atılan varsayımlardan, çatışan tarafların yanlışlarını fark ettirerek bu çatışmaların bir buhar gibi dağılmasını sağlar (Atay, 2009, s. 4).

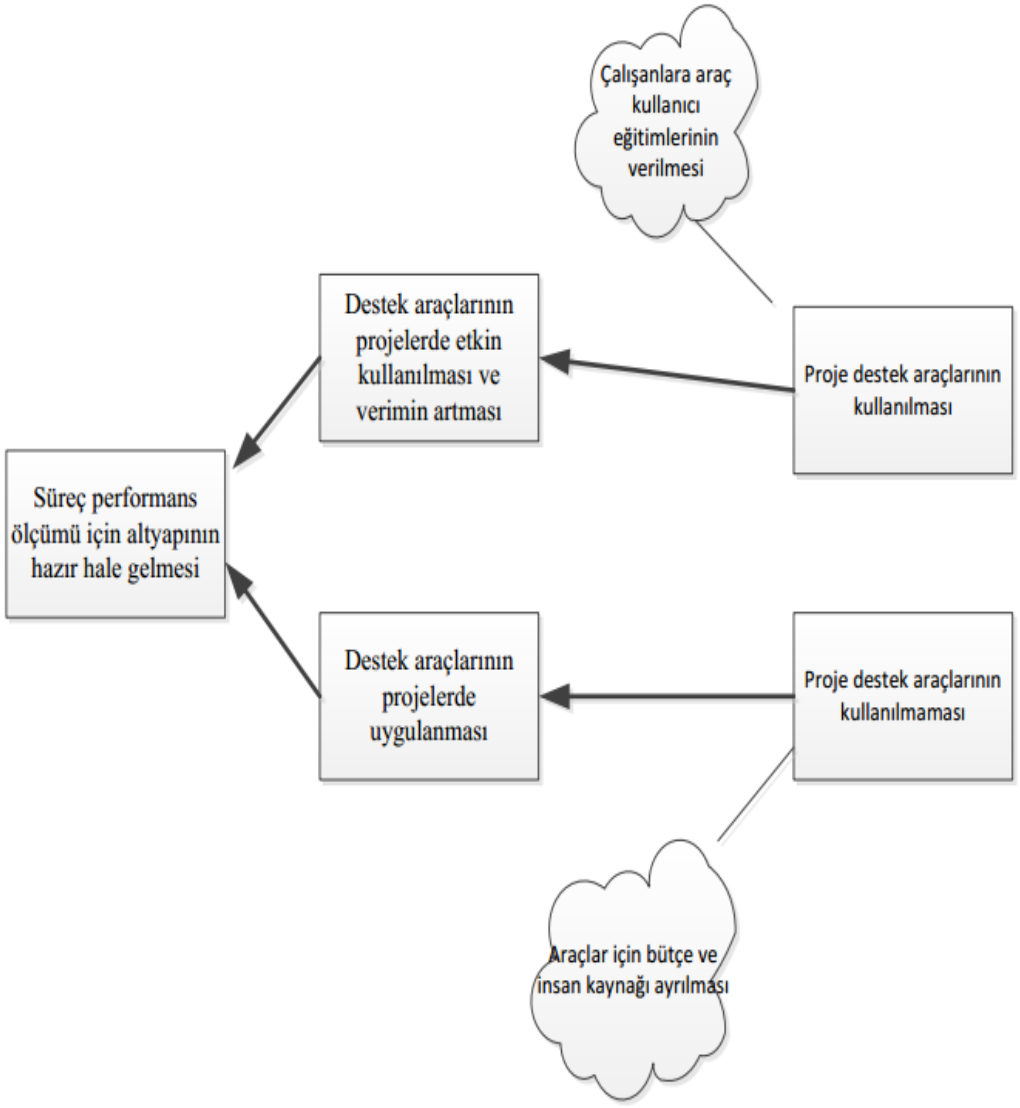
Büyükşehir belediyesi uygulamasında ise yazılımı kullanacak ve geliştirilecek personel için görevlendirme yapılması ve personelin değişiminin mümkün olduğunca engellenmesi çözümü ortaya atılır. Bununla birlikte atanacak personelin eğitim

planlamasının yapılması gerektiği tartışılarak çeşitli fikirler ortaya atılır. Yazılımı geliştirecek personel ile yazılımı kullanacak olan personel arasındaki iletişimin sağlanması gerektiği ile ilgili çözüm önerileri ortaya atılır.



Şekil 24: Buharlaşan Bulut Uygulaması 1
Kaynak: (Akman ve Karakoç, 2005, s. 114)

Şekil 25'teki buharlaşan bulut, yazılım projeleri geliştirme süreçlerinin performans ölçümlerinde karşılaşılabilecek çatışmalardan bir başkasını ve çözüm için olası enjeksiyonları göstermektedir. Burada yaşanan çatışma proje geliştirme aşamasında normalde kullanılması gereken destek araçlarının kullanılması ya da kullanılmamasıdır (Gün, 2015, s. 12).



Şekil 25: Buharlaşan Bulut Uygulaması 2
Kaynak: (Gün, 2015, S. 12)

3.3. Gelecekteki Gerçeklik Ağacının Kullanılması

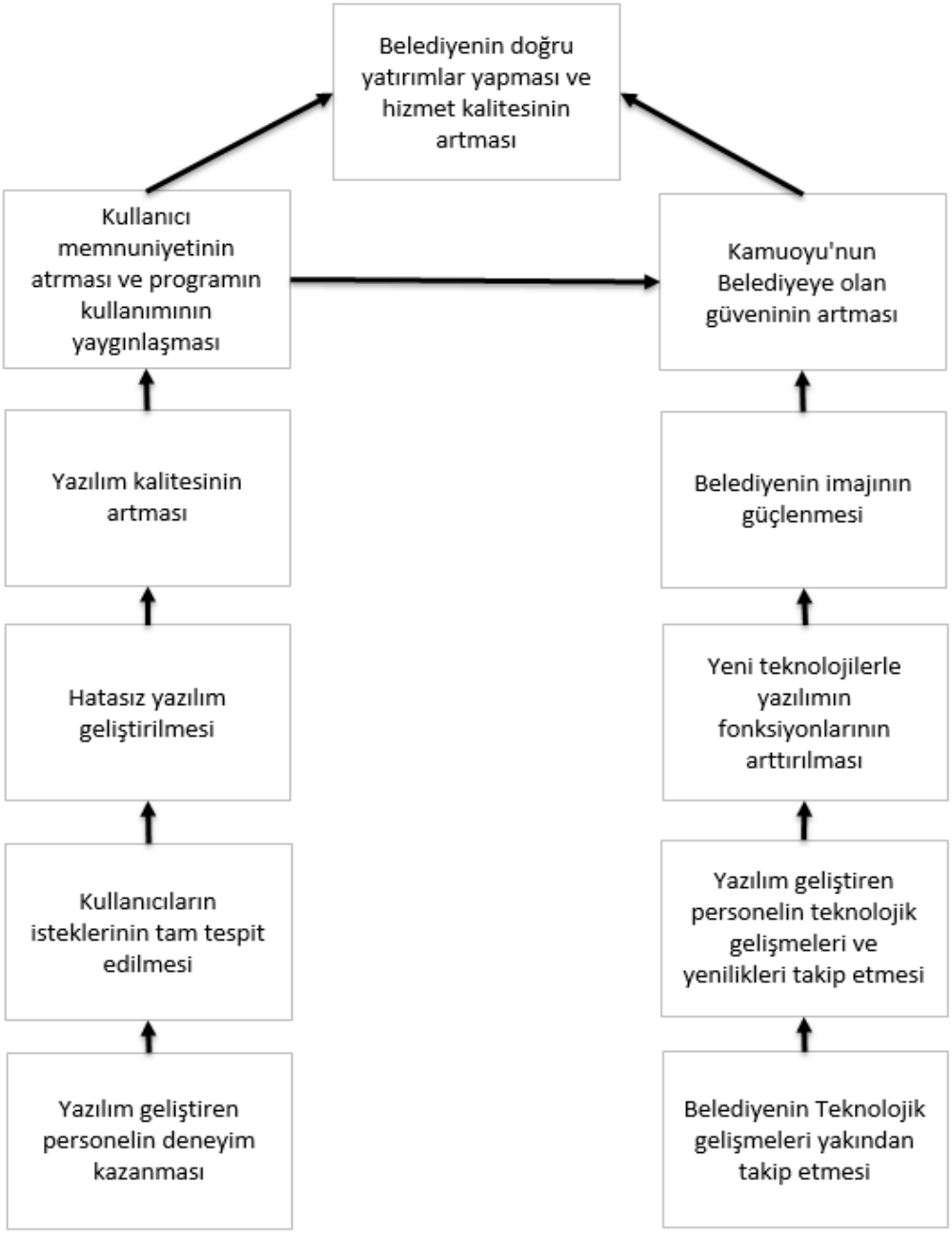
Firmaların veya kamu kurumlarının kalitesiz ve dokümantasyonsuz yazılım kullanmaları sonucunda bir takım sorunlar yaşadıkları tespit edilmektedir. Bunlar aşağıdaki gibi özetlenebilir (Akman ve Karakoç, 2005, s. 114):

- Rapor tasarımlarının yetersiz olması firmanın ve belediyelerin yatırım konusunda tespitlerini zorlaştırmaktadır.
- Kullanıcıların yazılımı anlayamaması yazılımın tam kapasiteyle kullanılamamasına sebep olur.
- Güncelleme gecikmeleri sebebiyle yeni teknolojik gelişmelere ayak uydurulamamaktadır.
- Yeni doğan ihtiyaçlara kısa sürede cevap verilememektedir.
- Yazlım geliştirilirken yapılan mantık ve kod hataları programın kapanmasına neden olmaktadır.
- Donanım ve yazılım uyuşmazlıkları performansın düşmesine neden olmaktadır.

Belediyeler ve firmaların karşılaştığı bu sorunlar ya yeni yazılım anlaşmalarını ya da mevcut yazılımın tekrardan geliştirilmesi ihtiyacını doğurmaktadır. Bu tür durumlar belediye ve firmaya hem fazladan maliyete neden olurken hem de üretim ve hizmet kalitesinde düşmelere neden olmaktadır. Yatırım fırsatlarının kaçırılması da ayrıca maliyete neden olmaktadır. Kaliteli yazılım kullanabilmek veya yazılımı kaliteli hale getirebilmek için Gelecekteki Gerçeklik Ağacı tekniğinden yararlanmak mümkündür.

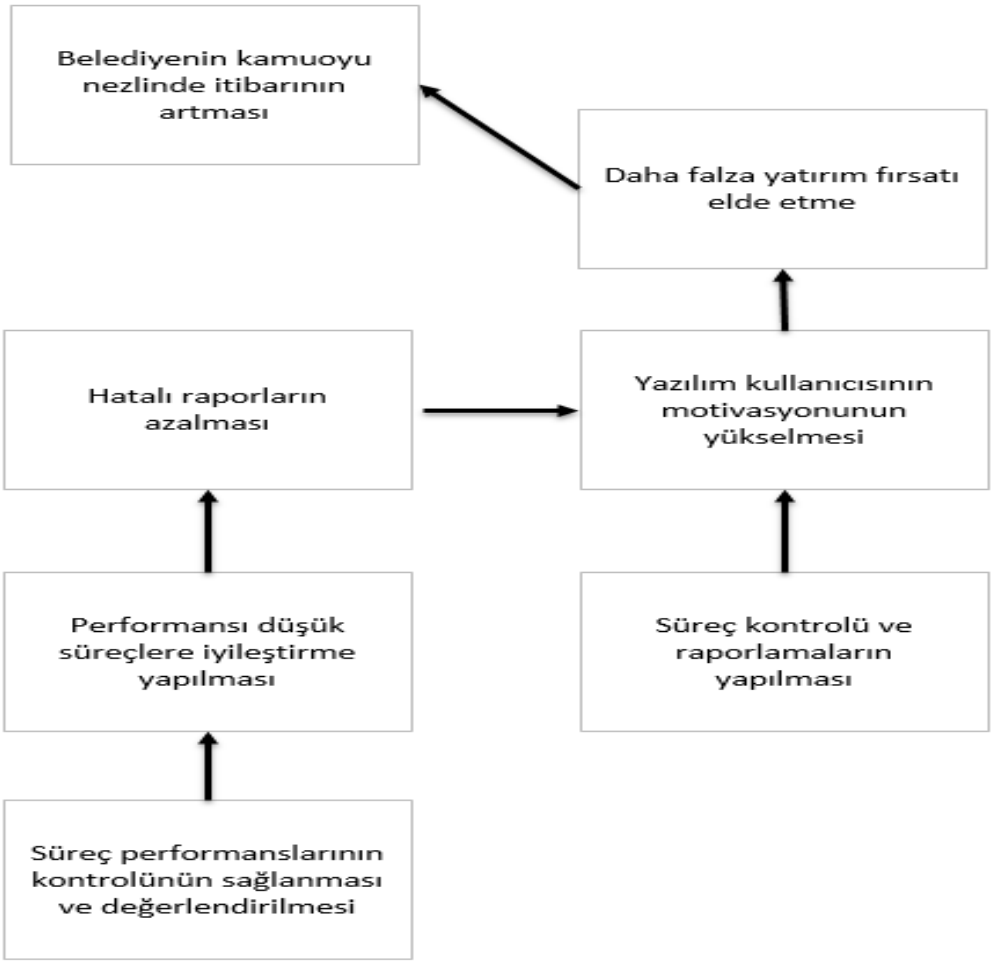
Şekil 26 ve Şekil 27'da Gelecekteki Gerçeklik Ağacı uygulaması belediye ve şirketin istenilen duruma ulaştığı anı göstermektedir.

Gelecekteki Gerçeklik Ağacının amacı buharlaşan bulut çalışmalarının istenilen etkiye ulaşacağının doğrulanmasıdır. Bir önceki aşama olan Buharlaşan Bulut'ta ortaya konan, kullanıcılar ve yazılımcılar arasında oluşturulabilecek işbirlikleri ile istenilen programın özellikleri ve karşılayacağı ihtiyaçlar tam anlamıyla tespit edilebilir, kaliteli ve hatasız yazılımlar geliştirilmesinin önü açılabilir. Dolayısıyla bu olayın sonucunda kaliteli yazılımlar doğar, yazılımın ihtiyaçları karşılama oranı artar ve kullanıcıların memnuniyeti artar. Bunların beraberinde getireceği etkiler ise yazılımı kullanan belediye ve firmanın değerinin maksime edilmesi, uzun vadede imajının güçlenmesi, güvenirliliğinin artması, performansının artması gibi etkilerdir (Akman ve Karakoç, 2005, s. 114).



Şekil 26: Gelecekteki Gerçeklik Ağacı Uygulaması 1

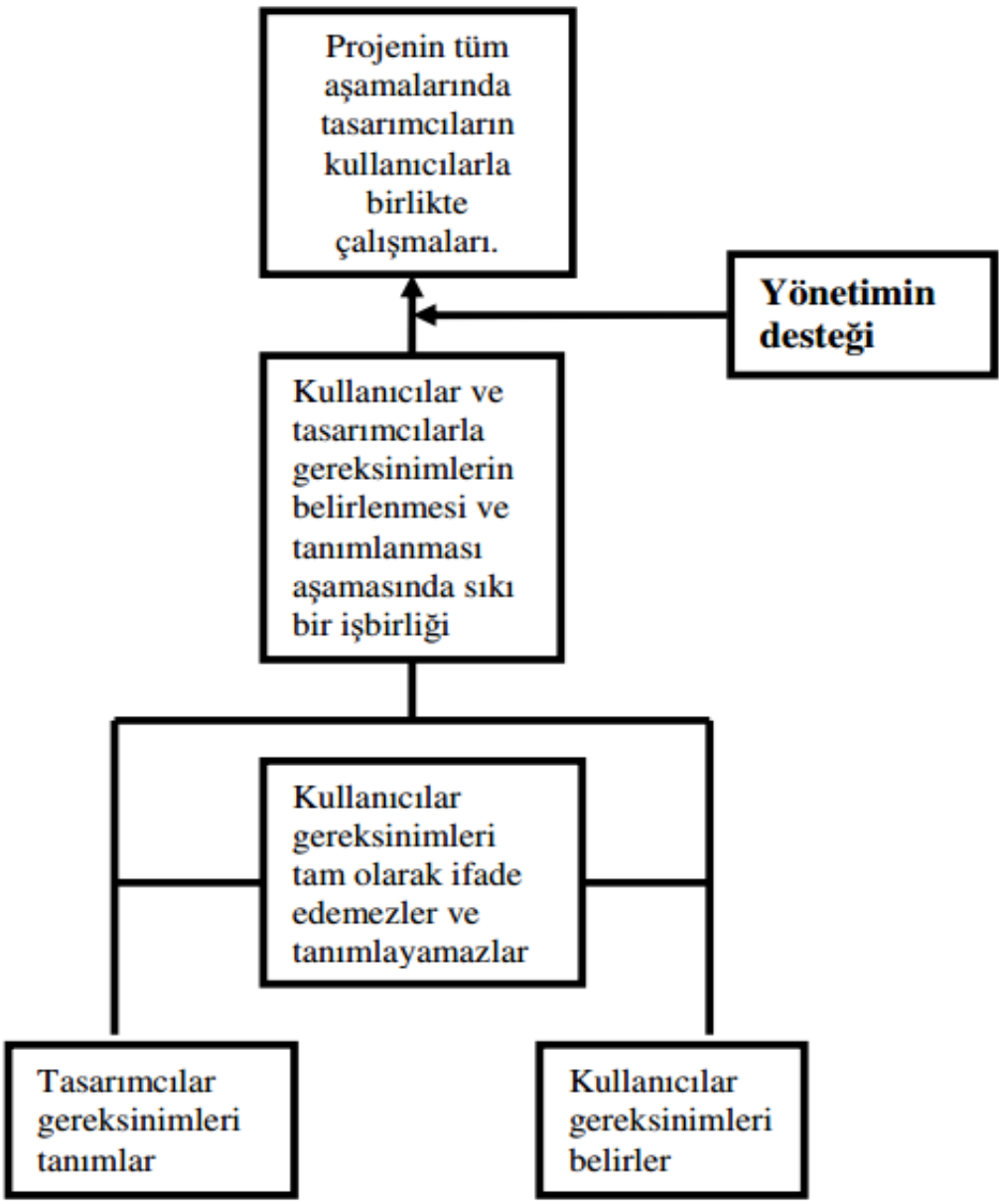
Şekil 25'teki Buharlaşan Bulut'ta bahsedilen proje destek araçlarının projelerde uygulamaya alınması ile üretkenlik (kod, dokümantasyon geliştirme süresi vb.) artar. Projede üretilen bilgilerin kişiye bağımlılığı ortadan kalkar. Performansı düşük olan süreç/alt süreçlerde iyileştirmeler yapılarak darboğazlar aşılır (Gün, 2015, s. 13).



Şekil 27: Gelecekteki Gerçeklik Ağacı Uygulaması 2

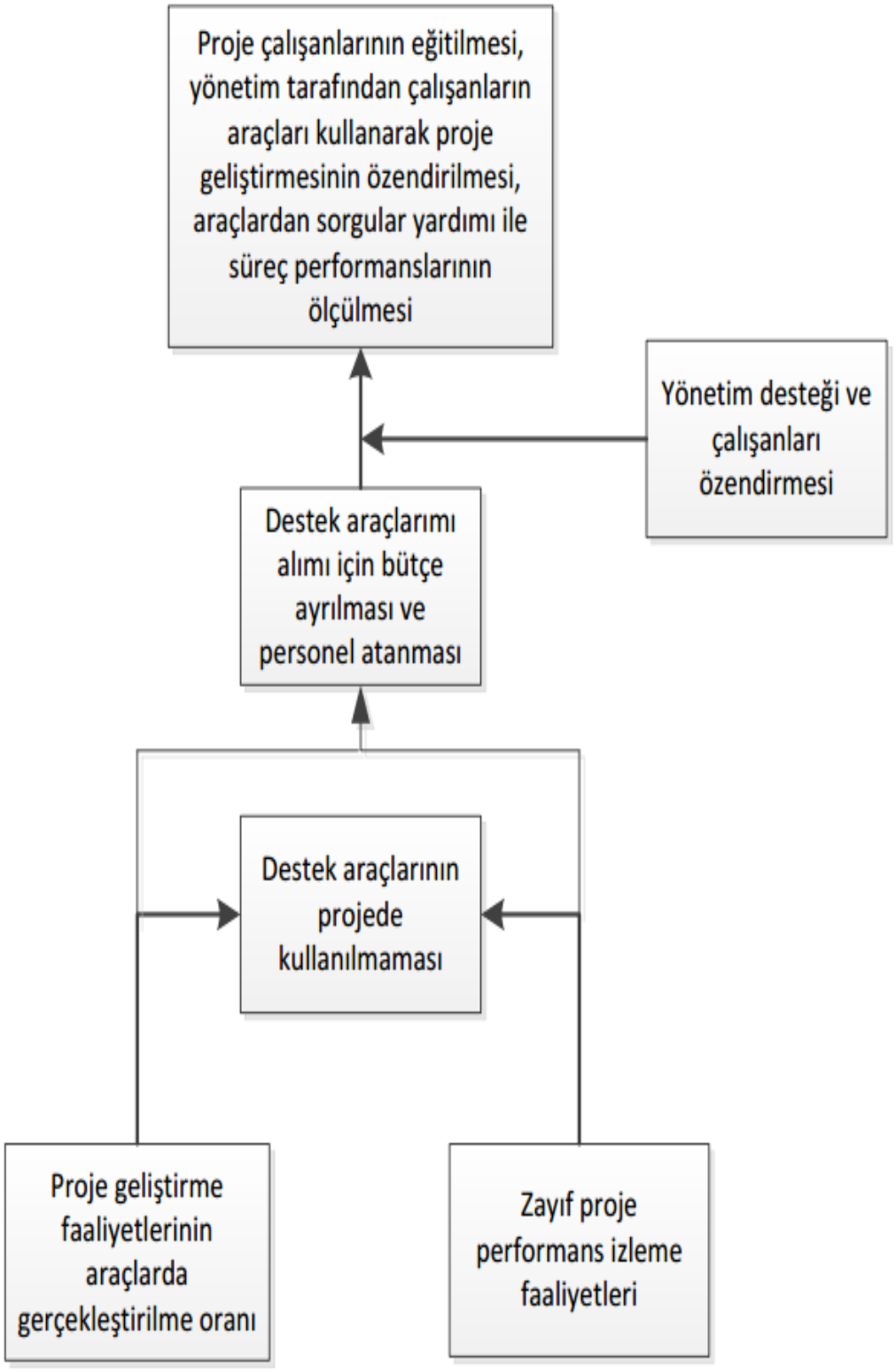
3.4. Ön Gereksinimler Ağacının Kullanılması

İstenilen duruma ulaşılabilmesi için değişimlerin uygulamaya konması gerekmektedir. Çözüm araçları ve guruplar Ön Gereksinim Ağacı'nda oluşturularak hayata geçirilirler. Bu aşamada geliştirilen çözümlere nasıl ulaşılacağının yolu aranır. Öncelikle çözümlerin önündeki engeller tanımlanır. Amaç kullanıcıların ihtiyaçlarını tam ve doğru şekilde tanımlayabilmektir. Bu kısıtın yok edilebilmesi için kullanıcı ve yazılımcılar sıkı bir işbirliği ile üst yönetimin desteğini de alarak, tüm analizleri doğru şekilde yaparlar ve gereksinimleri tam tespit ederler (Akman ve Karakoç, 2005, s. 114).



Şekil 28: Ön Gereksinimler Ağacı Uygulaması 1
Kaynak: (Akman ve Karakoç, 2005, s.114)

Başka bir uygulamada çözüme ulaşmamızı engelleyen durum, Şekil 29'daki ön gereksinim ağacında görüldüğü gibi proje destek araçlarının projelerde uygulamaya alınmaması ve kullanılmamasıdır. Bu engeli ortadan kaldırmak için proje destek araçlarının satın alınması için bütçe ayrılır, bu araçların kurulumu, yapılandırması, idamesi ve çalışanlara eğitim verilmesi için personel ataması yapılır (Gün, 2015, s. 14).

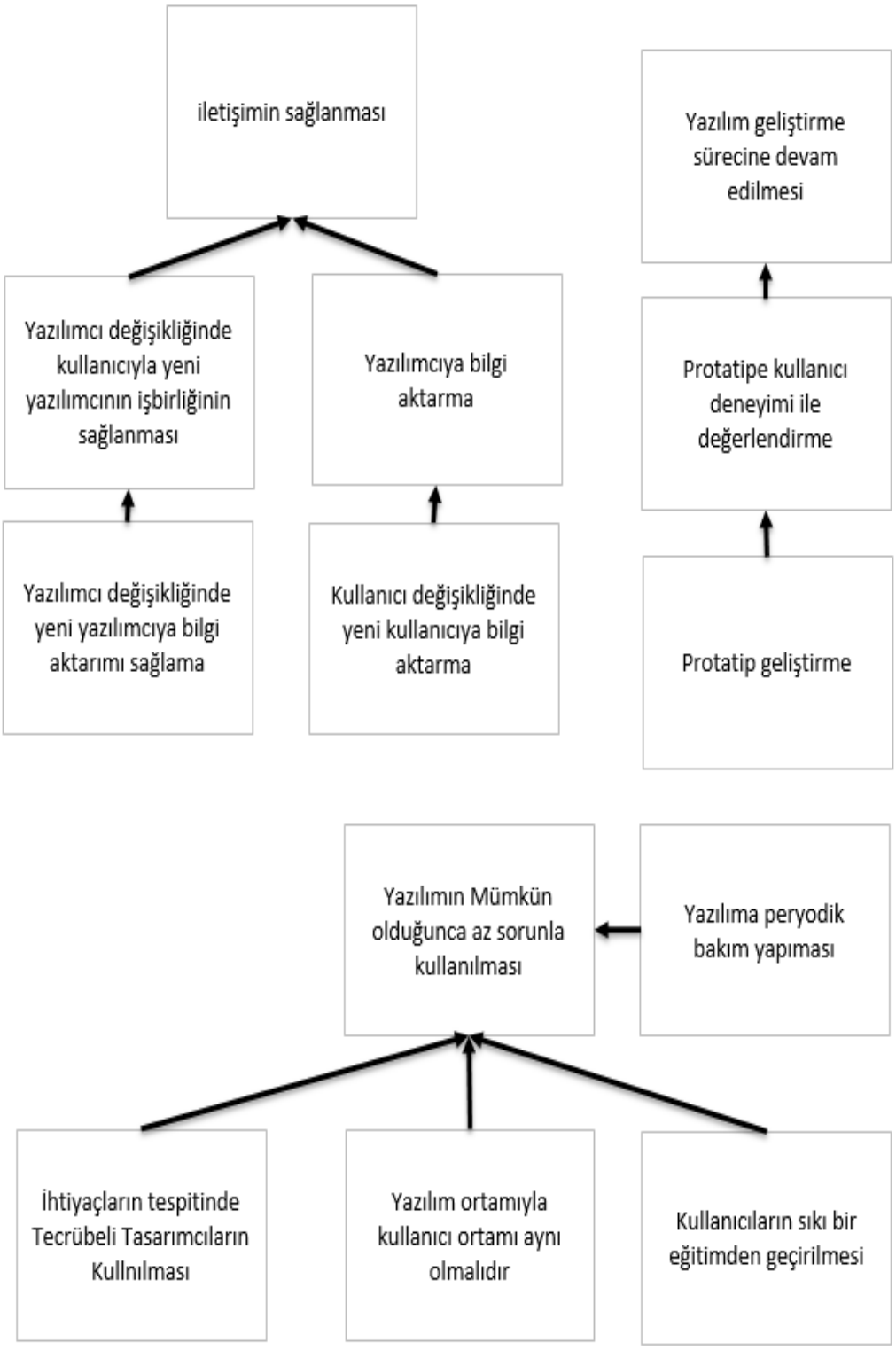


Şekil 29: Ön gereksinimler Ağacı Uygulaması 2
Kaynak: (Gün, 2015, s.14)

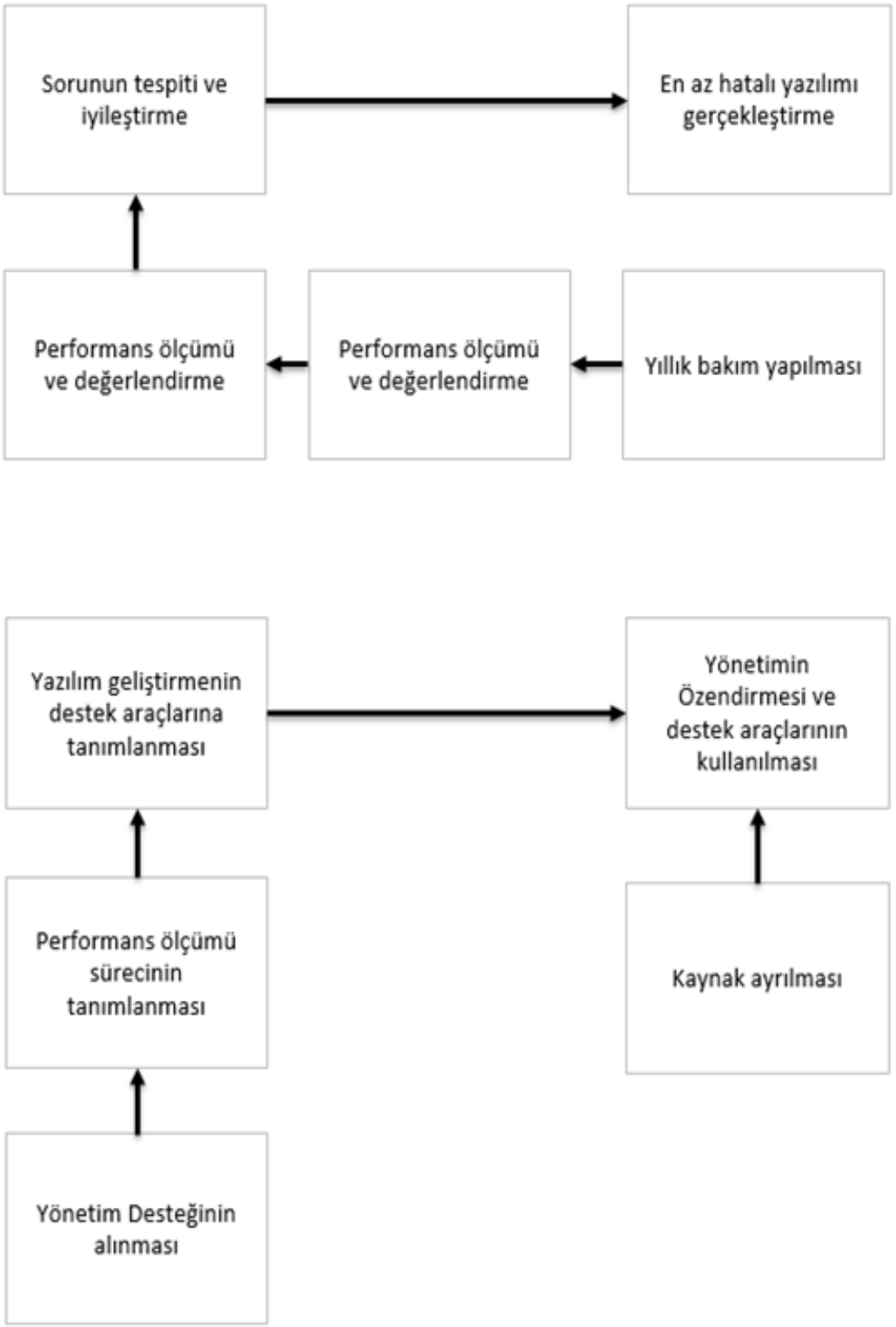
3.5. Geçiş Ağacının Kullanılması

Geçiş Ağacı ayrıntılı şekilde planlanan amaçlara ulaşmak için gerekli uygulamaların tanımlanmasını hedefler. Bu nedenle iletişim ve işbirliği ile koordinasyon kurularak hatasız bir yazılım üretmek mümkün olabilmektedir. Şekil 30'da ki Geçiş Ağacı programın kullanıcı gereksinimlerini tam olarak karşılayabilmesini sağlamak için tecrübeli yazılımcılarla çalışılması gerektiğini ortaya koymaktadır. Aynı zamanda kullanıcı değişikliklerinde yazılımcıya bilgi verilmeli ve yeni kullanıcıya bilgi aktarımının etkin bir koordinasyonla sağlanması gerekmektedir. Ön sürümler geliştirilip test aşaması güçlendirilmeli ve müşteri onayı alınmalıdır. Yazılımcı ve kullanıcı çevresinin aynı olması, kullanıcıların iyi bir ön sürüm eğitiminden geçirilmesi ve yazlım bitirilip kullanıma sunulduktan sonra bakım ve idame kısmının sürekliliğin sağlanması yazılımın kullanım ömrünün uzun ve sorunsuz olmasını sağlamaktadır (Akman ve Karakoç, 2005, s. 118).

Yazılım geliştirme sürecinin etkin bir şekilde yönetilebilmesi ve darboğazların giderilebilmesi için yönetim tarafından bir politika belirlenmelidir. Bu politika gereği performans ölçme ve değerlendirme süreçleri tanımlanmalı, ihtiyaç duyulan araçlar satın alınmalı, bu araçları yönetecek ve eğitim verecek personel görevlendirilmelidir. Proje çalışanları özendirilerek proje faaliyetlerinin araçlar kullanılarak yapılması sağlanmalı ve böylece kurumsal hafızanın oluşması sağlanarak kişiye bağımlılığın ortadan kaldırılarak standart iş yapma tarzı kuruma yerleşmelidir (Gün, 2015, s. 15). Şekil'31 de bir uygulama örneği sunulmuştur.



Şekil 30: Geçiş Ağacı Uygulaması 1



Şekil 31: Geçiş Ağacı Uygulaması 2

SONUÇ VE ÖNERİLER

Belediyeler kar amacı gütmeyen kuruluşlardır. Ellerindeki parasal kaynaklarını vatandaşa en iyi hizmet kalitesini sunabilmeleri için kullanırlar. Hizmet kalitelerini artıra bilmelerinin yolu ise sıkı bit takip sistemi kurmalarına bağlıdır. Yatırımların takibi, kültürel faaliyet takibi, yatırım için kullanılan kredilerin takibi v.b. takipler bunlardan bir kaçıdır. Bu takipleri hatasız yapabilmeleri için kaliteli yazılımlara ihtiyaç duyarlar. Günümüz belediyelerinin bünyesinde çalışan yazılım personellerinin kaliteli yazılım geliştirebilmeleri için kısıtlar teorisi düşünce süreçlerinin yazılım geliştirme süreçlerinde kullanılmasının sorunların daha yazılım geliştirilirken tespit etmelerine olanak sağlamaktadır. Dolayısıyla belediyelerin parasal kaynaklarını etkin kullanmalarına olanak sağlayacaktır. Belediye başkanlarının hizmet dönemleri kanun gereği 5 yıl ile sınırlandırıldığı için zamanı etkin kullanmalarına kaliteli yazılımların etkisi pozitif yönde olmaktadır.

Günümüzde yazılım firmaları yoğun bir rekabet ortamı içerisindedir. Bu nedenle firmaların rekabette üstünlük sağlayabilmeleri için yazılımlarının müşteri beklentilerini ve ihtiyaçlarını yeterince karşılaması gerekmektedir. Dolayısıyla yazılımların mümkün olduğunca az hatta sıfır hatayla çalışması bir zorunluluktur. Genellikle yazılım sektöründe, yazılımın müşteriye tesliminden sonraki aşamalarda da sorunlar ortaya çıkabilmektedir. Sorunların temel nedenleri planlama, analiz, tasarım ve gerçekleştirim kısımlarındaki eksikliklerdir. Bu adımlar sırasında karşılaşılan ve önemsenmeyen küçük sorunların ilerleyen aşamalarda ciddi kısıtlara dönüştüğü görülmektedir. Bu gibi durumlarda hem yazılım geliştiren firmanın emeği ve zamanı boşa gitmekte hem de müşterinin parasal kaynakları verimsiz olarak tüketilmektedir. Yazılım ürün kalitesi, sadece bitmiş ürünün test edilmesi ve hataların giderilmesi ile sağlanmaya çalışılırsa bu hem çok maliyetli olacaktır hem de projenin gecikmesine neden olacaktır. Önerilen çözüm, projenin erken aşamalarından itibaren süreç performanslarının sürekli izlenmesi ve sapmalar durumunda düzeltici önleyici faaliyetlerin başlatılmasıdır.

Projelerin başlamasından itibaren uygun proje yaşam döngüsü modeli seçilmelidir. Yazılım organizasyonu, yazılım geliştirme faaliyetleri için süreçleri kendi kurumunun özelliklerini ve iş yapış tarzını dikkate alarak özelleştirmeli ve projelerde uygulanmasını sağlamalıdır. Gereksinim yönetim aracı, yapılandırma ve değişiklik yönetim aracı, iş yönetim aracı, test yönetim araçları, risk yönetim araçları bir yazılım geliştirme organizasyonunun kullanacağı tipik araçlardır. Bu araçların organizasyonda kullanımı güçlü bir yönetim politikasına, bu politikanın gereği olarak süreçlerin tanımlanması, destek araçları için kaynakların ayrılmasına bağlıdır.

Kısıtlar Teorisi felsefesinin amacı en zayıf halkayı bulmak ve güçlendirmektir. Yazılım firmalarında, kısıtlar teorisi düşünce süreçlerini kullanarak, yazılımın müşteriye tesliminden önce, sorunlar tespit edilerek kısıta dönüşmesi engellenebilir. Bu sayede yazılım firmasının başarısını engelleyen faktörleri düzelterek firmanın rekabet gücünü büyük ölçüde artırmaktadır. Kısıtlar teorisi düşünce süreçleri mevcut gerçeklik ağacı, buharlaşan bulutlar, gelecek gerçeklik ağacı, ön gereksinimler ağacı ve geçiş ağacı yazılım geliştirme sürecinin yönetiminde etkin rol oynayarak süreçleri doğru şekilde yönetir ve sorunlara önceden önlem almaya imkân sağlar. Yazılım geliştirme süreçlerinin en temel sorunlarından birisi olan müşteri isteklerinin tam olarak anlaşılamamasıdır. Bu çalışmada bu sorunun oluşmasına neden olan faktörlerin tespit edilerek kısıtlar teorisi düşünce süreçlerini uygulamanın yararlarına vurgu yapılmıştır. Getirilen önerileri kısıtlar teorisinin son aşaması olan geçiş ağacında izlemek mümkündür. Geliştirme aşamasında oluşan en önemli kısıtın politik kısıtlar olduğu tespit edilmiştir. Literatür incelemelerinde de görüldüğü üzere kısıtlar teorisi süreçlerinin özellikle politik kısıtları gidermede çok etkili olduğu görülmektedir. Bu tez çalınmasında kısıtlar teorisi düşünce süreçlerinin kullanım alanının genişletilmesine katkıda bulunacağı ve gelecek çalışmalara yol gösterebileceği düşünülmektedir.

Kaynakça

Kitaplar

Büyükmirza, K. (2003). "*Maliyet ve Yönetim Muhasebesi"*. 9. Baskı, Ankara: Gazi Kitabevi.

Cox, J. (2004). *"The Goal: A Process of Ongoing Improvement",* North River Pres, 3th. Ed., 81.

Goldratt, E. M., ve Cox, J. F. (1984). "*The Race"*, First Edition, USA. North River Press, Inc., 346.

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

Scheinkopf, L. (1999). "*Thinking for a Change-Putting the TOC Thinking Processes to Use*". St. Lucie Press, USA.

Stein, R. E. (1996). *"Theory of Constraints Applications in Quality and Manufacturing"*. Marcel Dekker 2nd.ed, 2.

Umbela, M., ve Srikanth, M. L. (1995). "*Synchronous Manufacturing: Principles for World-Class Excellence",* First Edition, The Spectrum Publishing Company, Inc, USA. 50.

Yılmaz, G. (2007). "*Yazılım Geliştirme Yaşam Döngüsü"*. Yazılım Mühendisliği, 8-26.

Dergiler

Akman, G., ve Karakoç, Ç. (2005). "*Yazılım Geliştirme Prosesinde Kısıtlar Teorisinin Düşünce Süreçlerinin Kullanılması"*. İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi (7), 103-121.

Akman, G., ve Karakoç, Ç. (2005). *"Yazılım Geliştirme Prosesinde Kısıtlar Yazılım Geliştirme Prosesinde Kısıtlar"*. İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi: 4 (7) 111.

Atwater, B., ve Gange, M. L. (1997). *"The Theory of Constraints Versus Contribution Margin Analysis for Product Mix Decisions"* Journal of Cost Management, 11 (1), 14-22.

Balderstone, S., ve Keef, S. (1999). *"Throughput Accounting: Exploding an Urban Myth"* Management Accounting, 77, 26-28.

Blackstone, J. H. (2001). "*Theory of Constraints-A Status Report",* International Journal of Production Research, 9 (6), 1053-1080.

Boutquin, P., Poremsky, D., ve Slovak, K. (2000). *"Beginning Visual Basic 6 Application "*. Development, UK, Wrox Press., 35-23.

Büyüközkan, G., ve Feyzioğlu, O. (2003). *"Fuzzy-logic-based Decision Making"*. International Journal of Production Economics, 90 (1), 27-45.

Büyükyılmaz, O., ve Gürkan, S. (2009). *"Süreçlerde En Zayıf Halkanın Bulunması: Kısıtlar Teorisi"*. ZKÜ Sosyal Bilimler Dergisi 5 (9), 177-195.

Erol, M. (2008). "*Kısıtlar Teorisi (Yaklaşımı) ve Teorisinin Stratejik Maliyet Yönteminde Kullanımı"*. Muhasebe ve Finansman Dergisi stanbul Iss. 39, 39.

Filiz, A. (2008). "*İnsan İlişkileri Yönetimi, Kazı Hikâyesi"*. Elektrik Dergisi 12 (235).

Igel, B., ve Islam, N. (2001). "*Strategies for Service and Market Development of Entrepreneurial Software Designing Firms*." Teknovasyon. 21 (3), 157-166.

Kaygusuz, S. Y. (2005). *"Kısıtlar Teorisi: Varsayımlar, Süreç ve Bir Uygulama"*. Ankara Üniversitesi SBF Dergisi, 133-156.

Karagün, V., ve Sözen, M. (2017). "*Kısıtlar Teorisinde Kapasite Kısıtı Ve Bir Uygulama"*. Ekonomi ve Yönetim Araştırmaları Dergisi 2(6).

Küçükyavaş, N., Tanış, V. N., ve Ünal, E. N. (2006). *"Kısıtlar Teorisi ve Değişken Maliyet Sistemi"*. Analiz: Marmara Üniversitesi Muhasebe-Finansman Araştırma ve Uygulama Merkezi, 6 (15), 17-28.

Louderback, J., ve Patterson, W. J. (1996). *"Theory of Constraints Versus Traditional Management Accounting"*, Accounting Education, 1 (2) 189-196.

MacArthur, J. B. (1993). *"Theory Of Constraints And Activity Based Costing:Friends Or Foes?"*. Journal of Cost Management, 7,(2) 50-56.

Nambisan, S. (2002). *"Software Firm Evolution and Innovation-orientation"*. Journal of Engineering and Technology Management, 19 (2), 141-165

Özer, G. (2001). "*Dünya Sınıfı Bir Sistem ve Yönetim Yaklaşımı . Kısıtlar Teorisi ve Katkı Muhasebesi",* Verimlilik Dergisi, (2), 7-29.

Rahman, S.-U. (2002). *"The Theory of Constraints' Thinking Process Approach to Developing Strategies in Supply Chains"*. International Journal of Physical Distribution & Logistics Management, 32 (10), 809-828.

Rand, G. K. (2000). *Critical Chain:. The Theory of Constraints Applied to Project Management"*, International Journal of Project Management, 18 (3), 173-187.

Ronen, B. (2005). *"Special Issue on The Theory of Constraints – Practice and Research"*, Human Systems Management, 24. 1-2.

Rulh, J. M. (1997). *“Managing Constraints”, CPA Journal,"; “The Theory of Constraints within a Cost Management Framework”,* Journal of Cost Management,.11 ,(67)

Sadıç, Ş., Özdemir, D., ve Gözlü, S. (2006). *"Kısıtlar Kuramı Yaklaşımı İle Petrol İthalat Ve Ulusallaştırma Sürecinin İyileştirilmesi".* İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi 10, 99-118.

Scoggin, J. M., & Segelhorst, R. J. (2003*). “Applying the TOC thinking process in manufacturing: a case study”.* International Journal of Production Research, 41 (4), 767-797.

Seker, S. E. (2015). ”*Yazılım Geliştirme Modelleri ve Yazılım/Sistem Yaşam Döngüsü”*. YBS Ansiklopedi 2 (3).

Seker, S. E. (2014). *“Yazılım Geliştirme Hayat Döngüsü”.* YBS Ansiklopesidi, 11 (14).

Taşçı, S. (2005). “*Hemşirelikte Problem Çözme Süreci”*. Sağlık, 73-74.

Taylor, L. J., ve Ortega, R. D. (2003). *“The Application Goldratt’s Thinking Process To Problem Solving,”* . Academy of Strategic Management. 2 (2) , 9-14.

Tezler

Atay, G. (2009). *"Kısıtlar Teorisi ve Sap Projesinde Kısıtlar Teorisi Düşünce Süreçlerinin Uygulanması".* Basılmamış Yüksek Lisans Tezi, İstanbul Marmara Üniversitesi, Sosyal Bilimler Enstitüsü.

Soylu, S. (2017). *“Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği ve Çevre ve Şehircilik Bakanlığı için Öneriler.”* Uzmanlık Tezi Çevre ve Şehircilik Bakanlığı, 6-29.

Karamaraş, B. E. (2002). *“Kısıtlar Teorisi ve Muhasebe Uygulaması”*. Basılmamış Doktora Tezi, Dokuz Eylül Üniversitesi Sosyal Bilimler Enstitüsü, 23.

Kartal, Z. (2006). *“Kısıtlar Teorisi İle Senkronize Üretim Sistemi ve Bir Uygulama”*. Basılmamış Yüksek Lisans Tezi, Osmangazi Üniversitesi Fen Bilimleri Enstitüsü, 30.

Koç, G. (2017). “*Yazılım Geliştirme Modellerinin Güvenlik Açısından Analizi Ve Bir Güvenli Yazılım Geliştirme Modeli Önerisi.”* Hacettepe Üniversitesi Yüksek Lisans Tezi, 13.

Utku, B. D. (2007*). “Kısıtlar Teorisine Dayalı Süreç Katkı Muhasebesinin Muhasebe Yöntemleri ile Karşılaştırılarak Değerlendirilmesi: Bir Örnek Olay Çalışması”*. Basılmamış Doktora Tezi, Akdeniz Üniversitesi Sosyal Bilimler Enstitüsü, 1634.

Ünal, E. N. (2000). *"Kısıtlar Teorisi ve Yönetim Muhasebesi Açısından Değerlendirilmesi: Bir Sanayi işletmesinde Uygulama"*. Basılmamış Yüksek Lisans Tezi, Çukurova Üniversitesi Sosyal Bilimler Enstitüsü, 8.

Diğerleri

Atasoy, N. A. (2018). *"BLM 426 Yazılım Mühendisliği."* https://www.erseltarhan.com.tr/wp-content/uploads/2017/07/ymg_hafta_2.pdf, 29.11.2018 erişim tarihi

Bank, E. (2014). *"CBS Yazilim Gelistirme Sorun ve Çözum Önerileri"*. 14-17 Ekim 5. Uzaktan Algılama – Cbs Sempozyomu (Uzal-Cbs) 28-31.

Boutquin, P., Poremsky, D., ve Slovak, K. (2000). *"Beginning Visual Basic 6 Application"* . Development, UK, Wrox Press., 35-23.

Dettmer, H. W. (1997). *"Godratt's Theory of Constraints-A Systems Approach to Continous Improvement"*. ASQC Quality Pres, Milwaukee, 23.

Gün, Ö. (2015). *"Yazılım Proje Geliştirme Süreçlerinin Performansının Ölçümünde Kısıtlar Teorisi Düşünme Süreçlerinin Kullanımı."* Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Endüstri Mühendisliği Doktora Programı, 11.

Gürgen, O., ve Gençyılmaz, G. (2008). "*Kısıtlar Teorisi Yaklaşımı İle Ürün Karmasının Belirlenmesi"*. 24-25 Ekim 8. Ulusal Üretim Araştırmaları Sempozyumu.

Karadağ, L. (2010). "*Proje yönetimi, BT proje yönetimi ve başarısızlık nedenleri, Bilgi Yönetimi."* ERP Akademi , 2.

Köksal, G., ve Karşılıklı, U. K. (2000). "*Kısıtlar Teorisi ve Toplam Kalite Yönetimi Yoluyla Etkin Performans Yönetimi"*. 9. Ulusal Kalite Kongresi.

Natarajan, K. V. (2004). "*Efficient software development, Proceedings of*" Mid-Atlantic Student Workshop on Programming Languages and Systems, Seton Hall University, 3., 20-30.

Pauca, V. P. (2003). "*Software life cycle*." Wake Forest University, CSC, 331-361

Schwalbe, K. (2000). *"Information Technology Project Management"*. Course Technology Thomson Learning, Canada., 50-62.

Seker, S., ve Diri, B. (2010). *"TimeML and Turkish Temporal Logic"*. International Conference on Artificial Intelligence, ICAI'10, 10, 881-887.

Steinmuller, W. (1995). "*The U.S. Software Industry: An Analysis and Interpretive"*. Univesity Of Berkeley.

Sommervile, I. (2000)."*Software Engineering."* New York: Addison-Wesley, Harlow England, 45-63.