YBS Ansiklopedi

www.YBSAnsiklopedi.com

Cilt 2, Sayı 3, Eylül 2015

Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü

Sadi Evren SEKER

Istanbul Medeniyet University, Department of Business

Özet

Bu yazı kapsamında, yazılım geliştirme yaşam döngüsü veya sistem geliştirme yaşam döngüsü olarak bilinen SDLC kavramı ve bu kavrama bağlı olarak literatürde sıkça geçen ve temel olarak kabul edilebilecek bazı yazılım geliştirme metotları incelenmiştir. Bu yazılım geliştirme modelleri, şelale modeli (waterfall model), fıskiye modeli (fountain model), v-şekil modeli (v-shaped model), iteratif model (iterative model) ve spiral modeldir. Modellerin temel özelliklerinin tanıtılmasının yanında modellerin üstün ve zayıf oldukları yanların incelemesi ve sistem yaşam döngüsü içerisindeki konumları da incelenmiştir.
***Anahtar Kelimeler**:* Yönetim Bilişim sistemleri, Yazılım Mühendisliği, Proje Yönetimi, Yazılım Proje Yönetimi

Summary

This paper aims to cover the concept of software development life cycle or system development life cycle in the literature. Besides the concept, some well-known and relatively important software development methodologies are also covered in the paper. Some of these methods are waterfall model, fountain model, v-shape model, iterative model and spiral model. Besides the introductory sides of the models, their strengths and weaknesses are also covered together with the software development life cycle.
***Keywords:** Management Information Systems, Software Engineering, Project Management, Software Project Management*

1. Giriş ve Tanımı

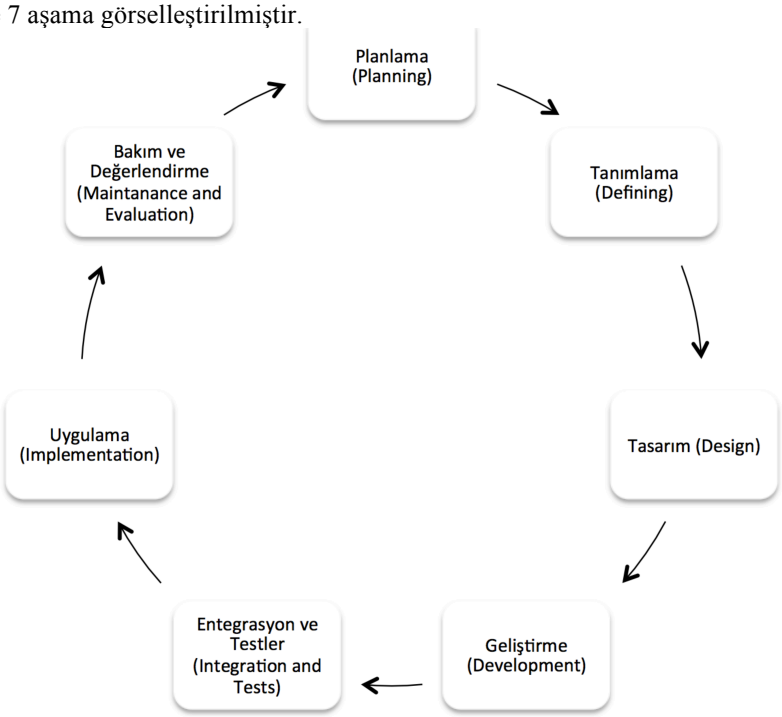
Yazılım Geliştirme Yaşam Döngüsü (SDLC) kavramı bir döngü yaklaşımdır (SEKER, 2014). Bir sarmal hareket halinde olan bir yaklaşımdır. SDLC aşamaları aşağıdaki şekilde sıralanabilir.

1. Planlama
2. Tanımlama
3. Tasarım
4. Geliştirme
5. Entegrasyon ve testler
6. Uygulama

Peer-review under responsibility of Academic World Education and Research Center.

7. Bakım

Yukarıdaki liste her ne kadar çoğu kaynakta ortak olan adımları içerse de, farklı kaynaklarda farklı adımlar eklenmiş veya farklı isimlerle sunulmuş olabilir. 10'a kadar çıkartan veya 4'e 5'e kadar indiren kaynaklardan bahsedilebilir. Şekil 1 de 7 aşama görselleştirilmiştir.



Şekil 1 SDLC Adımları

Kaynağa göre değişebilir ama anlatmak istenilen şey aynıdır. Hayat bir döngü içinde dönmektedir. Hepimiz hayatın içinde belli döngülerde yaşıyoruz. İnsanlar doğuyorlar, yaşıyorlar, ölüyorlar sonra toprak oluyor ve sonra o toprak başka hayatlara kaynak oluyor. Yağmur yağar, sular birikir, sonra buharlaşır, tekrar bulut olur ve tekrar yağmur yağar. Bir döngü her zaman vardır. Dolayısıyla Yazılımın da kendi doğal yaşam döngüsü olduğu bilmeliyiz. Bu döngünün doğru analiz edilmesi durumunda, doğru kontrollerin yapılması durumunda bizi başarıya götüreceğini bilmeliyiz. SDLC kavramı ilk başlarda yazılım yaşam döngüsü olarak kabul ediliyordu. Fakat sistem geliştirme yaşam döngüsü (system development life cycle) ismi daha çok kullanılmaya başlanıldı. Aslında her sistemin geliştirilmesi için kullanılan bir yapıya sahip. Dolayısıyla herhangi bir sistem geliştirirken de bu yaşam döngüsüne bakmakta fayda vardır. Yazının bu kısmında SDLC adımları teker teker anlatılacaktır, ardından avantaj ve dez avantajları tartışılıp literatürde sık geçen bazı yazılım geliştirme metotları ile karşılaştırmaları yapılacaktır. SDLC yaklaşımı, literatürdeki farklı ve tarihsel olarak sonradan çıkmış yazılım geliştirme metotları ile de çalışabilir. Örneğin saldırgan yazılım geliştirme (scrum software development) yaklaşımı ile birlikte uygulamaları da bulunmaktadır (SEKER, Saldırgan Yazılım Geliştirme (Scrum Software Development), 2015).

1.1. Planlama

Örneğin; bir şehir inşa edilecek, inşaatlar yapılacak onun alt yapıları yapılacak ve bu şehir yaşayacak bir organizma olacak. Şehrin içerisinde yeniler, eskiler, doğanlar ve ölenler şeklinde çok sayıda yaşam döngüsü unsurları bulunacak, işte bu döngünün nasıl kontrol edileceği sorusunun ilk adımı şehrin nasıl planlanacağıdır. Yani ilk adımımız ne istenildiğinin planlanması olarak düşünülmelidir. Bu plan aşaması aslında proje yönetimlerinde en önemli aşamasıdır. Diğer adımların da başlangıç aşamasıdır. İşin projelendirildiği, fikrin ortaya çıkarıldığı ve fikrin tartışıldığı aşamadır.

1.2. Tanımlama

Planlamadan sonra planlanan projenin tanımlarının yapıldığı aşamaya geçilir. Genelde istenilen fikrin ne olduğu ve temel tanımlar üzerinde konuşulacak kavramların tanımlandığı aşamadır. Çünkü aynı fikirden bahsedilmiyorsa farklı sonuçlara varılabilir. Bu tanımlar üzerine bir tasarım vardır. Bu analiz aşaması olarak da düşünülebilir. Problemin tanımlandığı veya yaşam döngüsünün tanımlandığı sistemin veyahut yazılımın tanımlandığı aşamadır. Tanım aşamasında projede nelerin istenildiği ile ilgili analiz çalışmaları da yapılabilir.

1.3. Tasarım

Yazılımımızın veya sistemimizin tasarımları yapılır. Projeleri çizilir. İnşaat projeleri gibi düşünülebilir. Planlama ve tanımlaya göre bir tasarım çizilir. Kararlar verilir, seçimler yapılır, örneğin yazılımın ekranları, ekranlarda neler bulunacağı, hangi ekranlara nasıl geçileceği, fonksiyonel olarak hangi adımların oluşturulacağı, hatta yazılımın bileşenleri, modülleri bu aşamada tasarlanır. Bir sonraki adım olan uygulma/geliştirmeye bütün kararlar verilmiş olarak geçilmesi beklenir. Geliştirme aşamasında herhangi bir soru veya karar bırakılmaz.

1.4. Geliştirme

Bu aşama, yazılım projeleri için kodlama olarak düşünülebilir veya sistemin yaşatılmaya başlandığı ilk örneklerinin çıkmaya başlandığı aşama olarak düşünülebilir. Daha önceden tasarım aşamasında karar verilen ortama uygun olarak, yine tasarım aşamasında verilen kararlar doğrultusunda projenin gerçekleştirilmesine başlanır. Bir anlamda daha önceden mimari projesi tamamlanmış inşaatın gerçekten yapılmaya başladığı aşamadır, yazılım projelerinin kodlandığı aşamadır.

1.5. Entegrasyon ve Testler

Sistemin artık gerçek hayata geçtiği aşamadır. Sistem veya yazılım hayat içinde bir yer edinir. Bir şirkette kullanılacak yazılımdan bahsediliyorsa şirketle ilgili birimlerin yazılımla ilgili eğitimlerin alınması, donanımlarının temin edilmesi, bağlantılı olduğu birimlerin buna entegre edilmesi, başka yazılımlarla bir entegrasyonu varsa bunun sağlanması ve veri tabanı bağlantıları gibi pek çok adım bu aşamada ele alınır.

1.6. Uygulama

Proje artık yaşayan bir yazılım haline gelir. Yazılım kullanılır ve problemler ortaya çıkar. Problemlere karşılık yeni fikirler ortaya çıkar, yeni ihtiyaçlar oraya çıkar ve bunlar değerlendirilmeye alınır. Bu değerlendirmeler sonrasında bakım aşamasına geçilir.

1.7. Bakım

Projenin bakımı yapılırken farklı aşamalardan geçirilir. Yeni tanımlar ortaya çıkar, yeni tasarımlar yapılır ve bu şeklinde devam eden döngü hiçbir zaman bitmez.

İşte yazılım veya kurulan herhangi bir sistem sürekli yaşayan bir üründür bir organizmadır. Tüketim ürünüdür. Belli bir yaşam süresi vardır. Belli bir süre sonra tükenir. Bunun yanında yazılım, bir mühendislik projesidir. Üretilir ve kalıcı olması beklenir. Uzun yıllar boyunca yaşaması beklenir. Belki hiç müdahale edilmeden çok ufak müdahaleler yapılması beklenen bir yaklaşım da vardır. Yazılım Geliştirme Yaşam Döngüsü (SDLC) bu iki yaklaşımın (tüketim ürünü ve mühendislik ürünü yaklaşımlarının) ortasındadır. Projenin bir planlaması ve proje yönetimi vardır. Uzun yıllar boyunca yazılımın yaşaması istenilir. Ama bu yazılımının hayatını devam ettirebilmesi için belli aralıklarla ufak dokunuşlar yapılması ve düzeltmeler yapılması da gerekir. Yazılım Geliştirme Yaşam Döngüsü işte tam da bu noktadadır. SDLC yapılacak düzenlemelerin nasıl yapılacağını organize eder. Proje yönetimi açısından bakıldığında yazılımcı belli aşamalardan geçtikçe biriktirdiği birikimleri bir sonraki aşamada kullanmaya başlar. Bir problem ortaya çıktığında SDLC yol göstericidir.

Geliştirme aşamasına girildikten sonra kodu geliştirip entegrasyona başladıktan sonra tanımlama ile ilgili bir hata ortaya çıktıysa iki ihtimal vardır. Projeyi tamamen iptal edilir tekrar planlamaya geri dönülür ve aşamalara en baştan başlanılır ya da entegrasyon yapılır projeye devam edilir.

Uygulama ve bakım bittikten sonra ikinci döngüde planlama aşamasında yeni bilgi alınıp devam edilir. Dolayısıyla Şekil 1'deki dairesel yapı aslında içinde doğrusal bir yapı da ifade eder. Bu dairedeki aşamalar arasında bir atlama beklenilmez. Ya döngü tamamlanır ya da bir problem varsa proje iptal edilip tekrar baştan başlanır. Genelde döngünün tamamlanması proje için daha iyidir. Çok büyük bir hata olmadığı sürece, büyük bir problem yoksa eğer döngü tamamlanmalıdır. Bir sonraki adımda küçük hatalar düzeltilir.

Tam bu noktada, yazılım kültürü denilen bir kültürden de bahsetmekte yarar vardır. Bir firmanın kendi yazılımını geliştirmesi için gerekli bir yazılım kültürü veya yazılım geliştirmeyip satın almak için gereken yazılım kültürü aslında aynıdır. Office programları gibi basit programlar satın alınıp kullanılabilir. Bu paket programların ne kadar derin kullanılacağı, kullanıcı ile ilgili bir durumdur. Programların kullanımında bazı problemler ortaya çıkabilir ve bu problemlerin de çözümleri için destek sistemleri bulunur. Ancak yine de kullanıcının yapabilecekleri, programın sundukları ile sınırlıdır ve kullanıcıya "al kullan ve fazlasını isteme" gibi bir limit koyar. Mesela özel bir problem olduğunda ya "bu problemin çözümü yazılımda yoktur" şeklinde bir algı veya bu problem için ilave yazılımlar mesela "plugins (eklentiler) alınması" veya özel yazılımlar yazdırılması gerekiyor. Böyle çok genel problemler için paket programlar yazılıp satılabiliyor ama özel problemler için hala işletmelerin kendi yazılımlarını yazdırma eğilimleri devam ediyor.

SDLC bu dünyayı ifade eden bir yaklaşım ve özel yazılımlar için işletmenin yazılım kültürünün bir parçası olacağı anlaşılıyor ayrıca paket program için de kullanılabilir. Paket olarak satın alınan yazılımlar, mesela Office yazılımı gibi Windows yazılımı gibi işletim sistemi yazılımları gibi yazılımlar için de bir SDLC söz konusu olabilir. SDLC yapılan planlama binlerce kişinin belki milyonlarca kişinin kullanacağı yazılımlar için yapılır. Tanımlama, tasarımlar bunun için yapılır ve burada toplanan bilgiler, uygulamalar, testler, entegrasyonlar bir yazılım geliştirme yaşam döngüsüdür.

2. SDLC Yaklaşımına Farklı Bakış Açıları

Yazılım yaşam döngüsü kavramına farklı kaynaklarda farklı yaklaşımlar olduğundan bahsetmiştik. Aşağıda alternatif olarak getirilen iki farklı yaklaşım bulunmaktadır. İlk yaklaşım SDLC adımlarını 4'e indirger ve bunlar aşağıdaki şekilde sayılabilir (Blanchard, 2006):

Planlama: Bu aşama, planlamanın bütün gereklerini tamamlar. Zaman kaynak ayrımı, personel, kurumun mevcut bilişim seviyesi ve yeni sistemin kullanılabilirliği, uzun vadede sistemin bakımı ve sürdürülebilirliği gibi çok sayıda parametre bu aşamada incelenir. Bu aşamanın tamamlanmasının ardından bir uygunluk raporu (feasibility report) hazırlanarak kurumdaki irade sahiplerine (yönetim) sunulur ve bilişim sisteminin yapacağı olumlu katkıların maliyet analizine göre katma değeri olduğu kesinleştirilir.

Sistem Tahlili (Analiz) : Bu aşamada sistem analizi yapılır ve geliştirilmekte olan bilgi sisteminin etkilediği diğer birimler ve mevcut işleyiş tahlil edilir. Örneğin geliştirilmekte olan sistem ödeme emirleri ile ilgili olsun, bu durumda girilen emirlerin satış, defteri kebir, stok kayıtları gibi çok sayıdaki etkilediği diğer kayıt tahlil edilerek bu sistemlerle olan etkileşimi çalışılır.

Sistem Tasarımı (Design): Bu aşamada sistemin bütün bileşenleri ve genel yapısı tasarlanır. Çok çeşitli tasarım yöntemleri ve aşamaları olmakla beraber genelde mantıksal bir tasarım aşamasından geçilerek sistemin hedefleri ve etkileşimde olduğu alt bileşenleri arasındaki uyum gözetilir. Ardından bu mantıksal tasarımın fiziksel tasarıma çevrilmesi aşaması gelir. Fiziksel tasarıma çevrilim aşamasında, programlama dilinden kullanılacak olan teknolojilere ve sistemin bütün parçalarına kadar olan tasarımı tamamlanır. (veri tabanı tasarımı, kullanıcı ekranları, veri akış yolları gibi).

Uyarlama ve işletme aşaması (implementation and operation): Bu aşama, şimdiye kadar yapılan bütün kağıt üzerindeki işlemlerin gerçeğe dönüştürüldüğü ve yazılım halini aldığı aşamadır. Kodlama, kod kontrolleri, testleri, kurulumu, yönetimi gibi işlemlerin tamamı bu aşamada yapılır.

Yukarıdaki 4 aşama kapsamında SDLC adımlarının çıktıları aşağıdaki şekilde özetlenebilir:

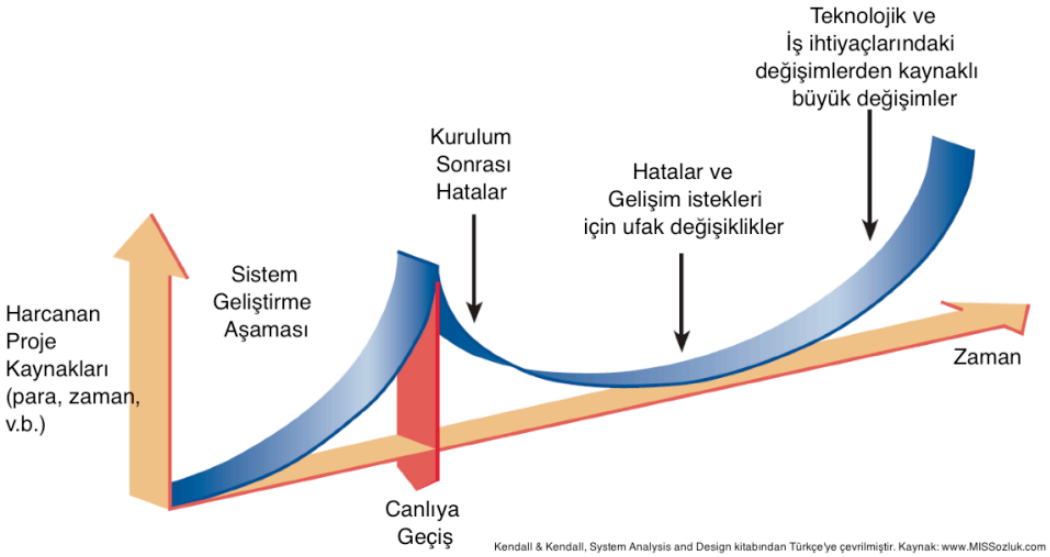
Tablo 1 SDLC Aşamaları ve Çıktıları

Aşama	Çıktı
Planlama ve seçim safhası	Sistem önceliklerіVeri, ağ, donanım ve bilgi sisteminin mimarisiSeçilmiş olan proje için detaylı çalışma planıSistemin geçerlilik alanı için özelliklerSistemin iş akışı açısından önemi
Sistem Tahlili	Mevcut sistemin tanımıMevcut sistemin düzeltilmesi, iyileştirilmesi, değiştirilmesi için önerilerMuhtelif sistemler için açıklamalar ve alternatiflerin değerlendirilmesiYeni teknoloji gereksinimlerinin belirlenmesi ve geçiş için öneriler
Sistem Tasarımı	Sistemin bütün alt bileşenleri için detaylı birer tasarım
Uyarlama ve işletme	KodDokümantasyonEğitim ve destek kaynaklarıYeni sürüm için kod, eğitim ve destek aşamalarındaki değişiklikler.

Yukarıdaki 4 adımlı yaklaşımın yanında Kendall and Kendall tarafından sunulan 7 adımlı yaklaşımdan söz etmek de mümkündür. Bu yaklaşım aşağıdaki adımlardan oluşur (Kendall, 1992):

1. Problem, fırsatlar ve hedeflerin belirlenmesi
2. İnsan seviyesi enformasyon ihtiyaçlarının belirlenmesi
3. Sistem ihtiyaçlarının (isterlerinin) belirlenmesi
4. Şimdiye kadar olan adımlar ışında elde edilen isteklere cevap veren bir sistemin tasarlanmsası
5. Yazılımın geliştirilmesi ve dokümantasyonu.
6. Sistemin testleri ve bakım adımlarının belirlenmesi.
7. Sistemin gerçeğe alınması ve değerlendirilmesi.

Yine Kendall ve Kendall kitaplarında bu geliştirme sürecindeki kaynak harcama miktarlarını aşağıdaki şekilde modellemektedir.



Şekil 2 SDLC Proje Maliyet Grafiği

Şekilden de anlaşılacağı üzere, sistemin geliştirilme aşamasında sürekli artan (üstel olarak artan) proje maliyeti varken, projenin canlıya geçmesinin ardından bu maliyet düşmeye başlar. Zamanla sistemdeki istek ve ihtiyaçların ölçeğinin artması ile proje maliyetleri de artmaya başlar. Hatta sistemin geliştirme aşamasındaki maliyetlerin bile üzerine çıkabilir. Bu yüzden projenin tasarımının ve analizinin ileride çıkabilecek ihtiyaçları en verimli şekilde karşılayacak yeterlilikte olması çok önemlidir.

3. SDLC Yaklaşımının Avantajları

Yazılım Geliştirme Yaşam Döngüsünde bir sonraki aşamada problemin düzeltilmesi mümkün olabilir dolayısıyla risk düşürücü bazı özellikleri vardır. İterasyon ile gerçek dünyaya biraz daha yaklaşılır. Gerçek dünyadaki problemlerin geliştirme süreçlerine dâhil edilmesi söz konusudur. Bilginin, enformasyonun sistematik olarak analizi yapılabilir. Plan aşamasında, analiz aşamasında veya tasarım aşamasında bir problem çıktığında nerede çıktığı, yeni adımda eklenecek şeylerin hangi adımda eklenmesi gerektiği ortaya çıkarılabilir. Modüler yaklaşım uygulanabilir. Bir modüldeki problemin diğer modüllere yansıması engellenebilir. Bir modülün tamamen iptal edilebilmesi söz konusu olabilir. Sisteme ve proje yönetimine yukarıdan bir bakış sağlar. Adımlar arasına kalite yönetimi aşamaları entegre edilebilir. Adımların çıktıları ölçülür ve bir sonraki aşamaya geçmeden önce o adımın belli kalite standartlarının sağlaması şartı konulabilir. Örneğin; tanımlama aşamasında tanımlama bittikten sonra tasarıma geçilecek olması, tanımlama aşamasındaki bir problem tasarıma da taşınacak demektir. Bu da dezavantajlarından birisidir. Bunun taşınmaması için tanımlama aşamasından sonra belli kalite standartları konulup kalite testlerini geçtikten sonra çıkan raporla ilgili tasarıma geçilmesi daha etkilidir. Dinamik ortam sürekli yaşayan, sürekli dönen bir ortamdır. Dolayısıyla dinamik ortamlar için uygundur. Örneğin; bir inşaat projesi yapılacak olsun ve genelde 100 yıl boyunca o inşaatın ayakta kalmasını beklenir. Bu çok dinamik bir proje değildir. Yazılım projeleri ise çok kısıtlı sürelerde çok büyük değişiklikler geçirir (SEKER, BT Projelerinde Yaşanan Problemler ve Çözüm Önerileri, 2014). Dolayısıyla SDLC burada doküman oluşturma açısından da bir avantaj sağlar. Proje yönetim yöntemlerinin sağladığı en önemli faydalardan birisi de bilgi yönetimi (knowledge management) açısından kritik öneme sahip doküman tabanını oluşturmasıdır (SEKER, Bilgi Yönetimi (Knowledge Management), 2014).

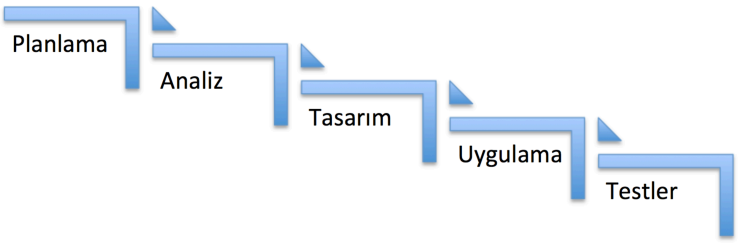
4. SDLC Yaklaşımının Dezavantajlar

Uzmanlık gerektirir. Yönetim ile ilgili belli tecrübelere ihtiyaç vardır. En az bir uzmanın projeye dâhil edilmesi hız kazandırır. Genelde, büyük sistemler için uygundur. Örneğin; ufak bir proje yazılacak. Hesap makinesi programı yazılacak. İyi bir programcı 2-3 saat içinde hesap makinesi programı yazabilir. Bu proje için SDLC uygulamak gerekli değildir. Hesap makinesi ile ilgili çok fazla bir şey istenmez. Bunun analizlerinin yapılması, tasarımlarının yapılması kalite standartlarının belirlenmesi mümkün. Fakat bu yaklaşım 3 saatlik projeyi 1-2 aya çıkabilir. 3 saatlik projenin maliyeti bir programcı için 100 lira civarıysa 1 aya çıkarttığında bu rakam 10bin lira civarına çıkabilmektedir. Genel anlamda belirtecek olursak büyük sistemler için daha uygun bir yaklaşımdır. Bir diğer dezavantajı ise adım testleridir. Genelde her adım sonrasında test yapılması mümkündür ama bu yaklaşımda genelde adım testleri atlanır. Adımlardan sonra bir tanımlama yapıldı. Bu tanımlamalar ile ilgili ön/son koşullar oluşturup test yapılması, maket uygulamalar yapılması müşteri ile bunların tartışılması, başarı kriterlerinin belirlenmesi mümkündür. Fakat SDLC da bu adımları göremeyiz. Biraz hızı arttırmak açısından bu şekilde uygulanmaktadır. SDLC bir yaşam döngüsü olduğu için bir sonraki dönüşte adımları test etme özelliği avantaj sağlamaktadır. Bunların adım adım test edilmesi mümkündür. Hatanın devamlılığı engellenemez. Tasarımda çıkan bir hata ancak bir döngüde ele alınıp çözülebilir. Tasarımdaki hata diğer aşamalara da taşınmak zorundadır.

Yazılım Geliştirme Yaşam Döngüsünün diğer yöntemlerle, SDLC içinde kullanılan modeller olarak düşünülebilir. SDLC'de bir yaşam döngüsü, geliştirme aşamaları ve entegrasyon aşamaları vardır. Bunlar ayrı birer proje metodu yöntemi olarak algılanabileceği gibi SDLC 'nin içinde kullanılan alt proje yönetim modülleri olarak da düşünülebilir. Bir yaşam döngüsü olarak düşünüldüğünde bu yaşam döngüsünün içinde çok farklı metotlarla geliştirilmiş, birden fazla projeden bahsedilebilir. Yazının devamında bahsi geçen bu farklı yazılım geliştirme metotlarına yer verilecektir.

5. Şelale Modeli (Waterfall Model)

Proje yönetim modelleri genelde şelale modeli (waterfall model) ile başlar. Yazılım mühendisliğinde ilk anlatılan modellerden birisi budur. Şelale modeli çok daha statiktir. Örneğin bir köprü yapılacak. İstanbul'da boğaz içi köprüsü yapıldı ve 10'larca yıldır hiç değişmiyor, ne beklentide farklılıklar var, ne sistemde. Kimse, köprüye her hafta yeni bir kat çıkalım, şerit sayısını arttıralım, altından daha büyük gemiler geçeceği için köprüyü biraz daha yükseltelim demiyor ve hele hele bunu her hafta her gün talep etmek gibi bir dinamiklik yok. Ancak yazılım projelerinde bütün bunları görebilirsiniz. Bazan projenin yeniden oluşturulmasına kadar giden taleplerin arka arkaya aynı gün içerisinde geldiği bile olur. Oysaki boğaz köprüsü yıllardır hiç değişmeyen bir projedir. Şelale modeli bu projeler için uygun bir modeldir. Ama yazılım dünyası böyle değil. Yazılım projelerinde her 5-10 dakikada bir yeni fikir gelebiliyor. Şelale modeli yazılım projeleri için en kötü yaklaşımlardan birisidir denilebilir. Yine de ilk olması ve basit olması ve diğer yaklaşımlara temel teşkil etmesi açısından önemlidir.



Şekil 3 Şelale Modeli

5.1. Şelale Modelinin Aşamaları

Planlama ile başlar ve ardından analizler yapılır. Genelde analiz ikiye ayrılır. Durum analizi yani elimizdeki kaynaklar nelerdir?, Windows ortamında mı geliştirilecek yoksa cep telefonlarında mı geliştirilecek? gibi bazı durum analizleri yapılır. Bir de istek analizler vardır. Ne istiyoruz? Sorusu sorulur ve ilgili analizler yapılır. Tasarımlarım aşamasında ise analiz aşamasındaki çıktılara göre geçilir. Yazılımın ekranları tasarlanması, fonksiyonel özelliklerinin tasarlanması gibi tasarımlar yapılır. Tasarım aşaması biraz geniş bir aşamadır. Ayrıntılı bir aşamadır. Nesne yönelimli ortamda kullanılacak bir yazılımda use case'lerinin çizilmesi, class diagramları'nın hazırlanmasına kadar giden tasarım aşamaları vardır. Daha sonra uygulamaya aşamasına geçilir. Yazılım projelerinde, uygulamayı (implementation) kodlama olarak düşünebiliriz. Geliştirme (Development) aşamasıdır ve burada bir sürü alt aşamasından bahsedilebilir. Uygulama aşamasındaki testler, uygulama aşamasındaki tasarım yanlışları, analiz yanlışlarına geri dönülmesi. Sonra sistemin genel olarak test edilmesi ve müşteriye teslim edilmesi aşamaları görülebilir. Planlama ile başlayıp testlere ulaşana kadar sisteme çok dokunma şansımız yoktur. Ancak yeni bir şey yapılacaksa tekrar planlamaya gelinip yeni bir proje olarak ele alınabilir. Uzun süren projeler için çok ciddi bir tehdittir. Çünkü bir proje 1-2 yıl sürüyorsa şayet 1-2 yıl boyunca bu projenin hangi aşamasında ise o aşamasının bitmesini beklemek zorundayızdır. Müdahale etme şansımız yoktur. Bilişim projeleri için çok uygun bir yönetim sistemi değildir. Ama en basit, en etkili ve hepsinin temeli olan yöntem olduğu için bilmekte fayda vardır. Küçük projeler için uygulanabilir. Hangi aşamada hata var, kimden problem kaynaklanıyor veya nerede problemi çözebiliriz gibi bilgileri verir. Genelde burada geri dönüşler de söz konusudur. Tasarımda bir problem olduğunda analize, uygulamada bir problem olduğunda tasarıma dönülmesi gibi geri dönüşler yapılabilir. Hataların maliyeti de giderek artar. Analiz aşamasındaki bir hatanın düzeltilmesi çok düşük maliyeti olurken, bu problem test aşamasında yakalanırsa tekrar analize dönülüp bunun telafisi tekrar bütün süreçlerin sıfırdan ele alınmasını gerektirir ve maliyet artışını beraberinde getirir.

5.2. Tarihsel Gelişimi

Metot tarihsel olarak üretim ve inşaat sektörlerinde kullanılmakta olan ve isteklerin zaman içerisinde değişmesinin maliyetli olduğu ve isteğin ortaya konmasının ardından üretim tamamlanana kadar herhangi bir

değişimin olmadığı alanlarda uygulanmaktaydı. Genelde inşaat ve üretim alanında bir üretimin tasarımı yapılıp üretime geçildikten sonra talepler ya hiç karşılanamamakta ya da kısıtlı oranlarda karşılanabilmektedir. 1983 yılında literatüre ilk model olarak Benington tarafından kazandırılan şelale modeli de bu yapıyı benimsemiştir (Benington, 1983). Fikrin yayınlanmasından çok daha önce 1956 yılında fikir Amerikan Donanması tarafından düzenlenen bir matematik sempozyumunda yine Benington tarafından sunulmuştur (NMCAP, 1956).

Fikir boyutunda metodun geliştirilmesi incelendiğinde, yine "şelale (waterfall)" ismini kullanmadan metottan bahseden ve genelde literatürde en çok atıfta bulunulan çalışma Royce'a aittir (Royce, 1970).

Makalenin dikkat çekici yanı, yazılım dünyasında yaşanan geliştirme süreçlerini analiz ederken ve genel olarak yapılan hataları ve problemleri anlatırken şelale modeli yaklaşımını, ismini zikretmeden kullanmış olmasıdır.

Royce tarafından önerilen orjinal metot aşağıdaki şekildedir (Royce, 1970):

- İster analizi (Requirement Specifications), bu aşamanın sonunda ürün ister dokümanı çıkar
- Tasarım (design), bu aşamanın sonunda yazılım mimarisi çıkar
- İnşa (construction), kodlama veya uygulama (implementation) aşamasıdır ve sonunda yazılım çıkar
- Uyum (integration)
- Test ve hata ayıklama (test and debugging)
- Kurulum (installation)
- Bakım (maintenance)

Royce tarafından önerilen orijinal şelale modelinin yanında geliştirilmiş modellerden bahsetmek de mümkündür ancak orijinal modelde aşamalar arası geçişler net bir şekilde belirlenmiş ve geri dönüş olmayacak şekilde tasarlanmıştır. Modelin diğer çeşitlerinin en önemli farkı, adımlar arasında geri dönüşlerin daha başarılı tanımlanmış olmasıdır.

5.3. Şelale Modeline Eleştirel Bakış

Şelale modelinin en zayıf yönü olan ve geri dönüşe izin vermeyen yapı, çoğu zaman önceki adımlarda yaşanan problemler yüzünden geri dönülmeyi gerektirir. Özellikle yazılım gibi esnek bir yapıdaki üretimin çok defa bu şekilde ihtiyaçlarda ve dolayısıyla analiz, tasarım gibi ilerleyen adımlarda değişikliklere ihtiyaç duyması yeni modellerin çıkmasına sebep olmuştur.

Bu konuda David Parnas tarafından "Mantıklı bir sistemi aldatmak" başlıklı yazısında yapılan kritik aşağıdaki şekilde tercüme edilebilir (Parnas, 1986):

"Çoğu sistemin detayları ancak sistem üzerinde çalışmaya başladıktan sonra anlaşılmaktadır. Yapılan işlemlerin bazıları, hata yaptığımızı göstermekte ve geri dönmemizi gerektirmektedir"

Bu karşı yaklaşımı destekleyen unsurlar aşağıdaki şekilde sıralanabilir:

1. Bilişim teknolojileri konusunda tecrübesi olmayan paydaşların teknolojik olarak neler yapılabileceğinden haberdar olmaması yüzünden isteklerinin değişime açık olması
2. Proje ihtiyaçlarının zaman içerisinde değişime açık olması
3. Proje paydaşlarının isteklerini doğru şekilde aktaramaması
4. Proje ekibinin acemiliği veya kodlama öncesi aşamalardaki kaynak eksiklikleri (nitelik ve nicelik anlamındaki eksikler)
5. Projenin ilerideki şeklinin önceden kestirilememesi

Yukarıdaki kritik noktalarına karşılık şelale modelini destekleyen unsurlar aşağıdaki şekilde sıralanabilir:

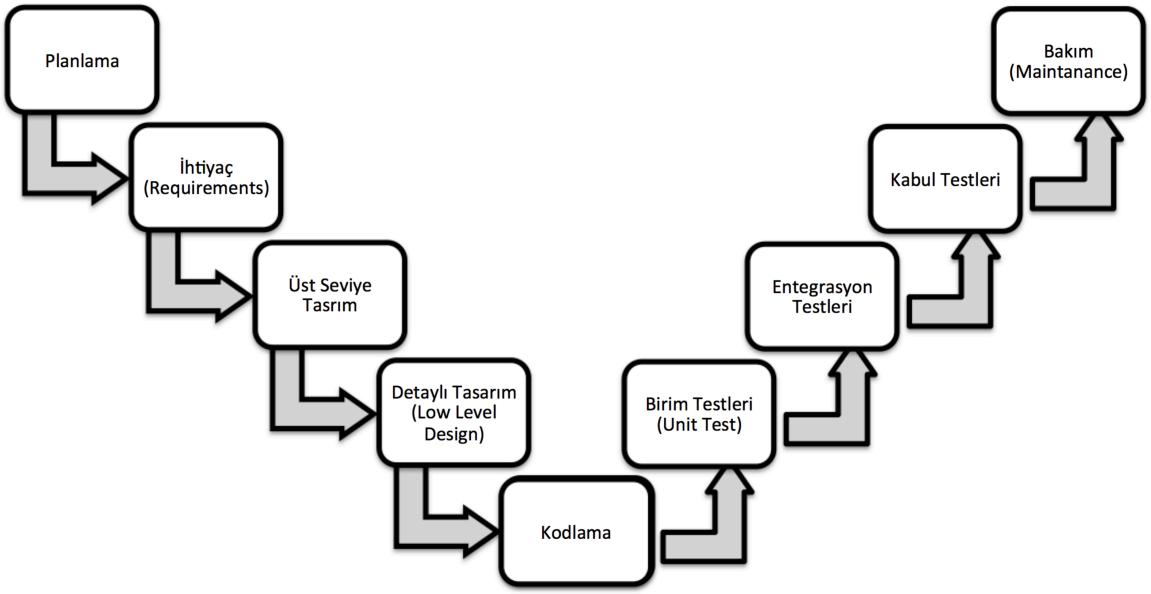
1. iki kere ölçüp bir kere biçmek yaklaşımının kurumda uygulanmasını sağlaması
2. Proje dokümantasyonu hazırlanmaya zorlaması
3. Hataların hangi aşamalarda ve kimin tarafından kaynaklandığının bulunmasını kolaylaştırması
4. Her sonraki aşamaya geçmeden önce, önceki aşamanın tamamlanması, bu sayede önceki aşamada ayrılan kaynakların serbest kalması ve farklı projelere aktarılabilmesi

Yukarıdaki destekleyen unsurlara ilave olarak bilinmesinde fayda olan bir özellik ise, çoğu yazılım mühendisliği derslerinde şelale modelinin bir zorunluluk olarak okutulduğudur. Bunun sebebi çoğu modelin aslında şelale modeline dayanan (üzerine devam şeklinde veya karşı görüş olarak) yapısıdır.

6. Fıskiye Modeli

Fıskiye modeli şelale modelinden öykünerek oluşturulmuş bir modeldir. Henderson-Sellers and Edwards tarafından geliştirilmiş bir modeldir. Şelale modelinden farkı, döngüleri vardır. Döngüler şeklinde her bir aşamada belli döngüler elde edilir. Tasarım aşamasında tekrar koddan tasarıma dönülmesi, tasarımdan istek analizlerine geri dönülmesi ve operasyona geçtikten sonra testlere geri dönülmesi gibi döngülere sahiptir. Maintenance (bakım) ve evolation (kriterler) belirleniyor. Bu yaklaşımda şelale modelindeki adımları görülebilir.

7. V-Şeklinde Model (V-Shaped Model)

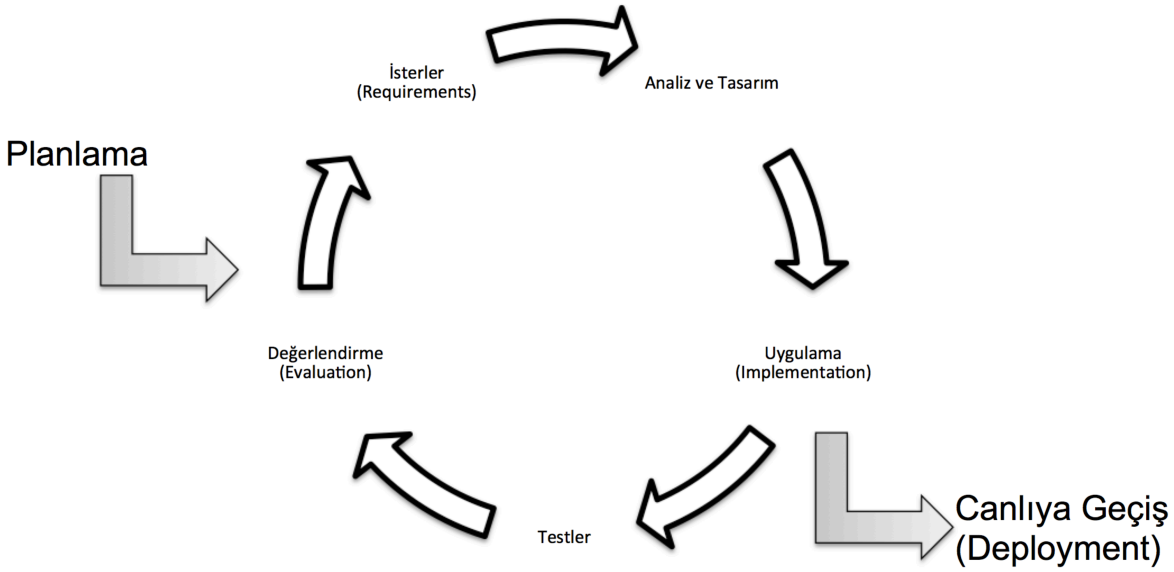


Şekil 4 V-Şekli Model (V-Shaped Model)

Adımlar V şeklini oluşturduğu için V-Şeklinde Model denilmiştir. Aslında şelale modeline benzer fakat şöyle bir farkı vardır. Yine planlama, ihtiyaç belirleme, üst seviye ve alt seviye tasarımlar vardır. Üst seviye de daha genel bakışa göre tasarım yapılır. Üst seviye tasarımların daha sonra daha detaylı alt seviye tasarımlarının yapıldığı görülür. Daha alt seviyelerinin açıldığı, bunların girdilerini-çıktılarını ve beklentilerinin yazıldığını söylenebilir. Sonrasında kodlamaya geçilir. Buraya kadar şelale modeli ile benzerlikler söz konusudur. Aşağı doğru akan bir proje yönetimi söz konusudur. Buradan sonra yukarı doğru çıkılmaya başlanır. Birim testleri yapılır. Her üretilen modülün testi yapılır. Daha sonra bu modüllerin/alt seviye tasarım ürünlerinin birbiri ile çalışma durumu, entegrasyon testleri yapılır ve kabul testleri aşamasında müşterinin kabul edip etmemesi ile ilgili müşteriye gidilir. Müşteri uygulamayı test eder. Sonra bakım başlar.

Bu yaklaşımın diğer bir özelliği de, şekilden görüleceği üzere aynı hizada olan aşamaların birbirini karşılamasıdır. Detaylı tasarım yapıldı. Bunun hemen karşısında kodlama bittikten sonra birim testleri yapılır. Yapmış olduğumuz tasarıma ait testler yapılır. Üst seviye tasarıma ait entegrasyon testleri yapılır. Her bir alt modülün birbiri ile uyumlu çalışıp çalışmadığı test edilir. İhtiyaç analizleri müşteriden istenir ve nelere ihtiyaç olunduğu çıkarılır. İhtiyaçların ise kabul testleri yapılır. "Evet o ihtiyaçlar sağlandı ve tamamdır, uygundur" şeklinde onunla ilgili test senaryoları hazırlanır. Planlamanın da bakım aşamasında karşılığını görüyoruz. Yaptığımız planlamaya uygun mu? Bakım aşamasından gelen bilgiler ile bu sorunun cevapları aranır. V-Şeklinde model aslında biraz daha alta inip yukarı çıkan ve her aşamasının karşılığının olduğu bir modeldir. "Bir şey yapıldı ama bunun karşılığı doğru mudur?" sorgulamasının yapıldığı, en alta kadar inip kodlamaya kadar inip sonra da yukarıya doğru sistemi taşıyan bir yapıya sahiptir.

8. Arttırımlı Model (Incremental Model)



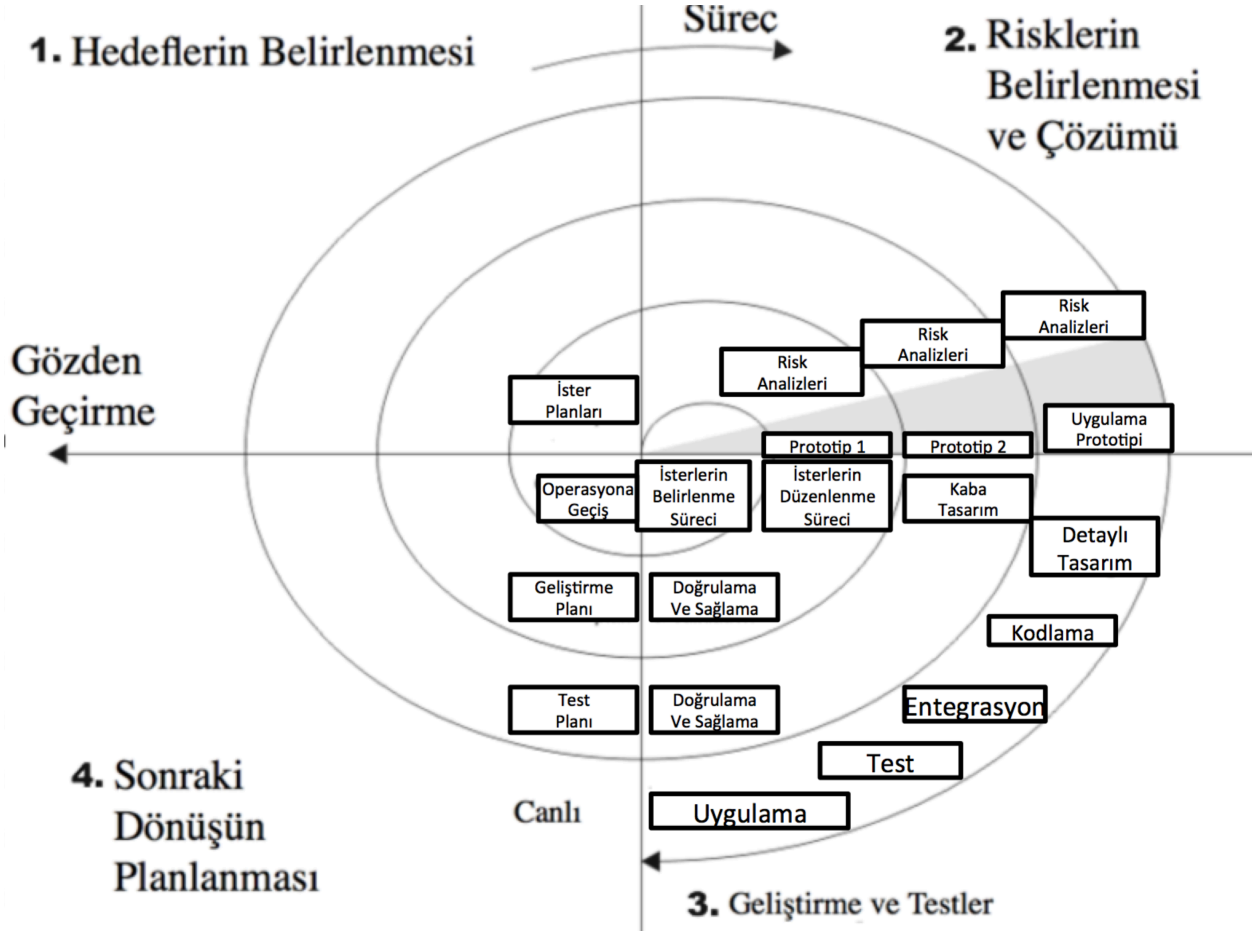
Şekil 5 Arttırımlı Model (Incremental Model)

Modelde bir döngü söz konusudur. Planlama aşaması ile başlar. Planlamalardan geçip bu planlamanın ardından bir döngüye girilir. İstek analizleri, analizler, tasarımlar yapılır ve uygulamaya geçilir. Uygulamadan sonra iki durum söz konusudur. Birincisi canlıya geçiş durumudur. Deployment edilir ve tabi problemler her aşamada olduğu gibi bu aşamada da yaşanır. Ama bu problemler testlerin birer parçası olarak görülebilir. Bir yandan da testlerimiz devam eder. Bu problemler sistemi besler. Testlerden ve canlıdan gelen bilgilere göre tekrar değerlendirilmeye tabii tutulur. Sonra tekrar istek analiz, analizler, canlıya bir bilgi verilmesi ve tekrar testler gibi süren bir döngü vardır. Bu sırada planlamada değişiklikler olabilir (Basili, 2003).

Örneğin; tekstil firmasına yazılım yazılıyor olsun. Tekstil firması iç süreçler değişebilir. Web sayfası üzerinden satış yapılıyor olsun. Web sayfası ile ilgili yeni ihtiyaçlar gelmeye başlar. Bir yandan yazılımın çalışan hali ile ilgili bir problemler ortaya çıkar. Bir yandan da "Orada şu modüle şu ekranı da ekleyelim" gibi fikirler, yen istekler çıkmaya başlar. İşte bütün bunlar arttırımlı modelde her seferinde arttırıla arttırıla bir iterasyon şeklinde devam eder ve her yenilik bir sonraki iterasyonda sürece dâhil edilir.

9. Spiral Model

Spiral model, diğer modellerden farklı olarak süreci oluşturan aşamalardan tekrar tekrar geçilmesini ve her geçişte projenin ilerleme kat etmesini hedeflemektedir. Örneğin projenin her döngüde yeni bir prototip çıkararak toplam 3 prototip ve dolayısıyla 3 döngüde ilerlemesi isteniyorsa, içeriden dışarıya doğru giden bir spiral şeklinde proje aşamalarında ilerlenerek dönülmelidir. Prototip model literatüre 1980'li yıllarda kazandırılmıştır (Boehm, 1986).



Şekil 6 Spiral Model Yaklaşımı

Bölgeleri 4 e böldüğümüzü düşünelim. Spiral şeklinde dönerek aşamaları ilerliyoruz. Bu dönme sırasında prototipler çıkarılıyor. Prototipler, maket uygulamadır. Ufak bir uygulama müşteriye gösterilir. Onun üzerinde müşterinin, yazılımı kullanacak kişinin fikir sahibi olması daha mümkündür. "Şurası olmamış. Böyle istiyorduk ama istediğimiz gibi olmamış." Deme şansı daha da yüksektir. Dolayısıyla bu prototip çıkartma işlemi şöyle ele alınıyor. Bir defa istekler belirleniyor. Operasyon ile ilgili fikir sahibi olunuyor. İhtiyaçlar belirleniyor. Daha sonra risk analizleri yapılıp prototip çıkartılıyor. Bu prototipten sonra tekrar bunun ihtiyaçlarla ilgili kontrolü yapılıp doğrulama ve onaylama aşamasından geçirilip bir daha yazılım sürecinden geçiriliyor. Yine bir risk analizi yapılıp prototip 2 çıkarılıyor. Sonra bunun test planının yapılıp ve risk analizinin yapılıp bir prototip daha çıkarılıyor. Bu prototip uygulamaya geçmeden önceki son prototip. Bu adımların sayısı arttırılabilir. Burada 2-3 prototip var 4-5 prototipe gidilebilir. Veya tek bir prototip de yeterli görülebilir projenin büyüklüğüne göre. Uygulama prototipinden sonra detaylı tasarım, kodlama, entegrasyon, testler ve uygulama süreci başlar. Bu son aşama şelale modeline benzer bir yapıda işler. Spiral model buraya kadar bize değişik döngülerden geçen risk analizlerinin yapıldığı ve sonunda bir prototipin ortaya çıktığı ve hem yazılım geliştirme ekibinin ve proje yönetim ekibinin, hem de müşterinin üzerinde mutabık olduğu bir prototip ortaya çıkartır ve daha sonra bu prototipi gerçekleştirip canlıya çevirebiliriz. Bu spirali de 4'e böldüğümüzde 4 aşaması vardır.

1. Hedeflerin belirlenmesi
2. Risklerin belirlenmesi ve çözümü
3. Geliştirme ve testler
4. Sonraki dönüşün planlanması

Böyle bir döngü içerisinde sürekli dönülür. Yukarı doğru gittikçe maliyetin büyümesi söz konusu ve sola doğru gittikçe de gözden geçirme süreçlerinin arttırılması söz konusudur.

10. Sonuç

Bu çalışma kapsamında, yazılım proje yönetimi açısından önemli bir yere sahip olan yazılım yaşam döngüsü veya sistem yaşam döngüsü olarak geçen kavram incelenmiştir. Kavramın, yazılım proje yönetim modelleri ile olan ilişkisi incelenmiş ve bu sayede bazı yazılım proje yönetim modellerine de giriş yapılmıştır.

Kaynaklar

Basili, C. L. (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer* .

Benington, H. D. (1983). Production of Large Computer Programs. *". IEEE Annals of the History of Computing (IEEE Educational Activities Department) , 5* (4), 350-361.

Blanchard, B. S. (2006). *Systems engineering and analysis (4th ed.)* . New Jersey:: Prentice Hall.

Boehm, B. (1986). A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes , 11* (4).

Kendall, K. E. (1992). *Systems analysis and design.* (Vol. 2). Prentice-Hall,.

NMCAP. (1956). United States. Navy Mathematical Computing Advisory Panel. (29 June 1956), Symposium on advanced programming methods for digital computers, [Washington, D.C.]: Office of Naval Research, Dept. of the Navy, OCLC 10794738.

Parnas, D. (1986). "A Rational Design Process: How and Why to Fake It". *IEEE Transactions on Software Engineering , 12* (2), 251-257.

Royce, W. (1970). Managing the Development of Large Software Systems . *Proceedings of IEEE WESCON 26 (August): 1–9* .

SEKER, S. E. (2014). Bilgi Yönetimi (Knowledge Management). *YBSAnsiklopedi , 1* (2), 8-14.

SEKER, S. E. (2014). BT Projelerinde Yaşanan Problemler ve Çözüm Önerileri. *YBSAnsiklopedi , 1* (3), 18-21.

SEKER, S. E. (2015). Saldırgan Yazılım Geliştirme (Scrum Software Development). *YBSAnsiklopedi , 2* (1), 12-15.

SEKER, S. E. (2014). Yazılım Geliştirme Hayat Döngüsü (Software Development Life Cycle). *YBSAnsiklopedi , 1* (2), 2-5.

United States. Navy Mathematical Computing Advisory Panel. (29 June 1956), Symposium on advanced programming methods for digital computers, [Washington, D.C.]: Office of Naval Research, Dept. of the Navy, OCLC 10794738. (n.d.).