T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÜVENLİ YAZILIM GELİŞTİRME SÜREÇ MODELİNİN SEÇİMİ

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Şeyda OCAK

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜH.

Tez Danışmanı : Prof. Dr. Nejat YUMUŞAK

Ocak 2012

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÜVENLİ YAZILIM GELİŞTİRME SÜREÇ MODELİNİN SEÇİMİ

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Şeyda OCAK

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜH.

Bu tez 19 / 01 /2012 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Prof. Dr. Nejat YUMUŞAK	Doç. Dr. Cemil ÖZ	Doç. Dr. Ayhan DEMİRİZ
Jüri Başkanı	Üye	Üye

TEŞEKKÜR

Bu çalışmada teşvik, yardım ve her türlü desteğini esirgemeyen danışman hocam Prof. Dr. Nejat YUMUŞAK'a ve her zaman her yerde maddi ve manevi olarak yardımlarını benden esirgemeyen aileme teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

SEI	: Software Engineering Institute (Yazılım Mühendisliği Enstitüsü)
CMM	: Software Capability Maturity Model (Yazılım Yetenek Olgunluk Modeli)
YM	: Yazılım mühendisliği
IEEE	: Elektrik Elektronik Mühendisleri Enstitüsü
ACM	: Association for Computing Machinery
ÇG	: Çalışma grubu
CS	: Computer Society
CERT	: Carnegie Mellon University's Computer Emergency Response Team
SNMP	: Simple Network Management Protocol
KG	: Kalite güvence
YGSM	: Yazılım geliştirme süreç modeli
YG	: Yazılım geliştirme
BDYM	: Bilgisayar destekli yazılım mühendisliği
SPSM	: Secured Process Selection Model
C	: Kriter
F	: Fonksiyon
SLCM	: Software Life Cycle Model
SSS	: Secured Software System
PSC	: Process Selection Criteria
SPM	: Software Process Model
CMMI	: Capability Maturity Model Integration
SDL	: Micosoft Güvenli Geliştirme Döngüsü
The RAD	: Rapid Application Development Model

ŞEKİLLER LİSTESİ

ÖZET

Anahtar kelimeler: Yazılım Yaşam Döngü Modeli, Güvenlik, Güvenli Yazılım Sistemi, Süreç Seçim Kriteri, Yazılım Süreç Modeli

Gelişen teknoloji ile birlikte günümüz şartlarında yazılım güvenliği ve güvenli yazılım süreç seçim kriterleri giderek önemli hale gelmeye başlamıştır. Uygulama geliştirme departmanları tarafından güvenlikten yoksun olarak yapılan projelerin dış kaynaklı saldırılardan korunması yazılımın güvenliği bakımından önem arz etmektedir.

Bu çalışmada altı yazılım süreç seçim kriteri çeşitli faktörler göz önüne alınarak incelendi. Yazılım süreç seçim kriterleri üzerinde çalışıldıktan sonra da, yazılım geliştirme yapan kurumların, sistem saldırılarından korunmak için yazılımlarını destekleyen güvenli bir model seçmelerinin önemi anlatıldı.

Altı fonksiyonel nokta değerli ve otuz beş kriterli bir set oluşturuldu. Seçilen altı yazılım geliştirme süreç modeli de bu kriterlere göre değerlendirildi. Kriter setine göre olması gereken güvenli bir yazılım geliştirme süreç modeli oluşturuldu ve altı yazılım geliştirme süreç modeli de istenilen güvenli modele göre karşılaştırılarak değerlendirildi. Bu değerlendirme sonucunda da içlerinden en güvenli ve de en güvensiz yazılım geliştirme süreç modeli belirlendi. Örnek olarak üç adet canlı yazılım projesine de aynı kriterler uygulanarak hangi yazılım geliştirme süreç modelinin uygulandığı belirlendi.

Çalışmanın sonucunda çok büyük projelerde uygulanabilirliği açısından, yazılım sürecinde kullanıcının projeyi her aşamada test edebiliyor olmasından ve de proje yöneticisinin projeyi her aşamada gözlemleyebiliyor olmasından dolayı en çok tercih edilen model ve en güvenilir yazılım geliştirme süreç modeli olarak Sarmal Model belirlendi. Aynı şekilde prototiplerin yavaş ve hantal programlar olması ve de müşterilerin geliştirilen prototipi gerçek program gibi algılaması ihtimali olduğundan Prototip modeli en az tercih edilen ve de en güvensiz yazılım geliştirme süreç modeli olarak belirlendi.

SELECTION OF SECURED SOFTWARE DEVELOPMENT PROCESS MODEL

SUMMARY

Key Words: Software Life Cycle Model, Security, Secured Software System, Process Selection Kriteria, Software Process Model

In today's terms with developing tecnology, software security and secured software process selection criteria are becoming increasingly important. Security is an important property of any software for many applications are outsourced where the application deveopment lacks strong integration of software security. These six software process selection criterias are considered by various factors. After working on these software process selection criterias, the importance of selecting a model is explained for protecting the companies which develops software by system attacks.

A set with thirty-five criteria and six functional point values was created. The selected six software process selection models were evaluated with this set. A most wanted secured software process selection model was created with this criteria set and the other six software process selection models were compared with this wanted secured model. As a result the most secured software process selection model and the least secured software pocess selection model were determined. As a sample the same set was applied to three living software projects to find which software process selection model is used.

As a result of the study, the applicability of many major projects , the user is able to test in every cycle of processing software and a project manager can observe at every stage of the project and at every stage of the test , the Spriral Model was determined as the most reliable and the most preferred model as a model of the software development process. In the same way Prototypes are slow and cumber and is also the possibility of customers' perception of the prototype model was developed as a prototype for at least the actual program was determined as the most preferred software development process model identified as unsafe.

BÖLÜM 1. GİRİŞ

Geleneksel yöntemler artık günümüz yazılım uygulamalarında etkili olamamaktadır. Bu konuda da şirketlerin en başarısız oldukları konuların başında yazılım güvenliği gelmektedir. Proje bolluğu, gelişmiş yazılım güvenliği, ürün çeşitliliği ve teknolojinin gelişmiş olmasına rağmen birçok uygulama günümüzde yazılım güvenliğinden yoksun şekilde geliştirilmektedir. Bu durum da şirketlerin dışarıya açılış kapılarında (ekstranet), intranet ve internet güvenliği konusunu daha da önemli hale getirmektedir.

Yazılım güvenliği konusu projeleri kötü niyetli saldırılara karşı korumak amaçlı ortaya çıkmıştır. Şuan ki organizasyon çevreleri yazılım sistemini riske atacak saldırılara karşı kritik güvenlik gereksinimlerinden yoksun şekilde oluşturulmaktadır. Yazılım güvenliği ölçümlerini dâhil etmek için, işletmeler var olan uygulama geliştirme yaşam metodolojilerini değiştirmek zorundadırlar. Geleneksel yöntemlerde, birçok yazılım geliştirme şirketleri yazılım yaşam döngüsünün en sonunda yazılım güvenliğine yer vermektedirler.

Günümüzde yazılım sistemleri birçok iş alanının temel bileşenleri arasında yer almaktadır. Örnek olarak bankacılık, telekomünikasyon gibi bazı sektörlerde gerekli yazılımlar olmadan organizasyonun yaşamını devam ettirmesi mümkün gözükmemektedir. Bu yüzden, artan rekabet, gelişen teknoloji ve yazılım kuruluşlarının artan kabiliyetlerinin de etkisiyle gelişmiş yazılım sistemlerine olan ihtiyaç her geçen gün artmaktadır. Bu seviyedeki yazılım sistemlerinin gerçekleştirilmesi karmaşık projelerin başarıyla tamamlanmasına bağlıdır. Yazılım geliştirme işlemlerini destekleyen süreçlerin yokluğunda ise, bu projelerin başarıya ulaşma ihtimali çok düşüktür.

Carnegie Mellon Üniversitesi bünyesindeki Yazılım Mühendisliği Enstitüsü (Software Engineering Institute - SEI) tarafından oluşturulan Yazılım Yetenek

Olgunluk Modeli (Software Capability Maturity Model - CMM) yazılım sürecini, "yazılımın ve yazılımla bütünleşik diğer ürünlerin (örnek olarak; proje planları, tasarım dokümanları, kod, test durumları, kullanıcı kılavuzları) geliştirilmesi ve bakımı için kullanılan aktivitelerin, metotların, uygulamaların ve dönüşümlerin oluşturduğu bütün" şeklinde tanımlamaktadır[1]. Son yirmi yıl içerisinde birçok yazılım süreci modeli geliştirilmiştir. Bunların başlıcaları arasında TickIT, ISO 9001, BOOTSTRAP, CMM, ISO/IEC 12207, ISO/IEC TR 15504 (SPICE), Unified Process (UP) sayılabilir. SEI"nin ortaya koyduğu CMM bunlar arasında önemli bir yere sahiptir. Bu, CMM"in en geniş şekilde uygulanmış ve kabul görmüş modellerden biri olmasından kaynaklanmaktadır [1].

CMM, yazılım geliştiren organizasyonlara süreçlerini iyileştirmeleri için, yazılım tedarikinde bulunan organizasyonlara ise yazılımı üreten müteahhit firmaların kalite düzeyini değerlendirmeleri için yardımcı olmayı amaçlamaktadır. Bu amaç doğrultusunda, CMM, yazılım firmalarına süreçlerinin mevcut olgunluk düzeyini belirleyip süreçleri iyileştirme stratejilerini seçmelerine ve bu iyileştirme çalışmaları sırasında kendilerini başarıya götürecek anahtar faktörleri tespit etmelerini sağlar.
Buradaki, temel varsayım, her süreç modelinde olduğu gibi, süreçler iyileştikçe, bu süreçlere göre üretilen ürünün kalitesinin de artacağıdır [2].

Yazılım geliştiren organizasyonların yürüttükleri projeleri düzgün bir şekilde, yani planlanan bütçe içerisinde kalarak, zamanında ve beklenen kalite seviyesini tutturarak, tamamlayabilmeleri için çeşitli yazılım araçlarına ihtiyaçları vardır. Yazılım projeleri yürütülürken karşılaşılan yönetimsel ve teknik zorluklar bu ihtiyacı daha da arttırmaktadır. Aslında, bir yazılımın ortaya çıkarılması için yapılması gereken birçok işin ilgili yazılım araçları olmadan tamamlanması pek mümkün görülmemektedir. Özellikle, yazılım geliştirme projelerinin yürütülmesine ve yönetilmesine yönelik araçların projede varlığı gittikçe daha önemli bir faktör haline gelmektedir. Bu yüzden, bu çalışmada "Yazılım Geliştirme Bilgi Sistemleri" olarak adlandırılabilecek bu tür araçlar açıklanmış ve bu araçların kullanımı bir bilgi sistemi modeli önerisi ile gösterilmiştir. Eğer ki servis organizasyonları bütünleşik güvenlik faktörleri ve güvenlik unsurlarının güvenli ölçüm modellerinden yoksun

sistemlere idrak ettirilmesi konusunda başarısız olurlarsa, bu müşteri memnuniyeti ve güvenini zedeleyen bir durum olacaktır [3].

Bu tez çalışmasında, yaygın olarak kullanılan yazılım geliştirme süreç modelleri karşılaştırılarak, gelişen yazılım mühendisliği projelerinde uygun ve güvenli yazılım süreci için en iyi kriter modelinin seçilmesi ve tüm yazılım yaşam döngü sisteminin güvenli hale getirilmesi hedeflenmiştir. Bunun için de yazılım geliştirme süreç tekniklerinin yazılım geliştirme hayat döngüsü içerisindeki öneminden bahsedildi. Süreç modelleri arasından altı tanesi detaylı olarak anlatıldı. Bunlar: Çağlayan (Şelale) Modeli, Artırımsal Model, Yeniden Kullanılabilir Yazılım Modeli, Prototip Modeli, Helezonik (Sarmal) Model ve Evrimsel Modeldir. Bu modelleri karşılaştırmak için altı fonksiyonel değerli otuz beş parçalık bir kriter seti oluşturuldu. Yine bu sete göre olması gereken istenilen güvenli yazılım geliştirme modeli oluşturuldu ve bu altı yazılım geliştirme süreç modeli de kriter setine uyarlanarak olması gereken yazılım geliştirme süreç modeline göre karşılaştırıldı.

Karşılaştırmanın sonucunda da çok büyük projelerde uygulanabilirliği açısından, yazılım sürecinde kullanıcının projeyi her aşamada test edebiliyor ve de proje yöneticisinin projeyi her aşamada test edebiliyor olmasından dolayı en çok tercih edilen model ve en güvenilir yazılım geliştirme modeli olarak belirlendi. Prototiplerin yavaş ve hantal programlar olması ve de müşterilerin geliştirilen prototipi geçek program gibi algılaması ihtimali olduğundan prototip modeli en az tercih edilen ve de en güvensiz yazılım geliştirme süreç modeli olarak belirlendi.
Bu modellerin seçilmesinde Tablo 4.1'de belirtilen kriterlere göre birbirine bağlı dağıtılan sistemlerde en iyi güvenlik konusunu ele alacak olan kriter Yüksek-Orta (f4-f5-f6) toplamlarıdır çünkü bu çözüm için en iyi alan adreslerini sunar.

Bölüm 2'de yazılım mühendisliği, yazılım mühendisliği konuları, yazılım güvenliği konusu, yazılım geliştirme yaşam döngüsü ve güvenlik eksikliklerini vurgulayan yazılım süreç modelleri anlatılmıştır. Yazılım geliştiren organizasyonların önerilen bilgi sisteminin ortaya çıkmasına vesile olan ihtiyaçları ve bilgi sisteminin gerçekleştirilmesi sonucu ulaşılması planlanan amaçları listelenmekte ve yazılım geliştirme bilgi sisteminin temel işlevleri sunulmaktadır.

Bölüm 3'te yazılım geliştirmede güvenli süreç seçim modelleri olarakta aşağıda belirtilen bilinen yazılım süreç modelleri detaylı olarak anlatılmıştır: V Süreç Modeli, Gelişigüzel Model, Barok Modeli, Araştırma Tabanlı Model, Helezonik (Sarmal) Model, Prototip Modeli, Hızlı Uygulama Geliştirme Modeli, Çağlayan (Şelale) Modeli, Artırımsal Model, Evrimsel Model, Yeniden Kullanılabilir Yazılım Modeli ve Çevik Modelleme.

Bölüm 4'de yazılım geliştirme süreç modellerini karşılaştırmak için otuz beş kriterden oluşan ve altı fonksiyonel nokta değerli bir set hazırlanmış ve yazılım süreç modelleri arasından en çok kullanılan altı örnek model detaylı olarak bu set üzerinden incelenmiştir. Yine kriter setine göre istenilen güvenli yazılım süreç geliştirme modeli oluşturulmuş ve bu altı yazılım geliştirme süreç modeli de güvenli modele göre karşılaştırılmıştır. Yapılan incelemeler sonucunda da en güvenilir olan süreç modeli belirlenmiştir.

Bölüm 5'de karşılaştırılan modeller arasındaki sıralama incelenmiştir. Örnek olarak üç adet canlı yazılım projesine de hazırlanan set uygulanarak hangi yazılım geliştirme süreç modelinin kullanıldığı tespit edilmiştir.

Bölüm 6'da ise tartışma ve öneriler kısmında, tez çalışmasındaki sonuçların değerlendirmesi yapılmış ve güvenli yazılım geliştirme süreç modelinin bulunabilmesi için kullanılabilecek başka yöntemlerden bahsedilmiştir.

BÖLÜM 2. YAZILIM GELİŞTİRME YAŞAM DÖNGÜSÜ

Geliştirilen yazılımın, üretim aşaması ve kullanım süreci boyunca geçirdiği tüm aşamalar "Yazılım Geliştirme Yaşam Döngüsü" olarak tanımlanır. Yazılımın üretildiği aşamadan itibaren işlevsellik ile gereksinimleri sürekli değişecek ve bu değişiklikler de yazılımın genişlemesine neden olacaktır. Dolayısı ile bu aşamalar bir döngü olarak ele alınmak zorundadır. Bu döngü doğrusal ve tek bir yöne ilerleyen bir döngü değildir çünkü herhangi bir aşamada geriye dönmek, geliştirmeyi yapmak ve tekrar ilerlemek mümkündür.

.1. Yazılım Mühendisliği

“Yazılım (software)” ilk kez bir istatistikçi olan John Tukey tarafından 1952 yılında kullanılmıştır.

Yazılım Mühendisliği, 1968 yılında yazılım geliştirmedeki problemleri tartışmak ve gerekli çalışmaları tanımlayabilmek için NATO tarafından düzenlenen bir konferansta, güvenilir ve etkin olarak çalışan yazılımların ekonomik olarak elde edilebilmesi için mühendislik ilkelerinin uygulanması olarak tanımlanmıştır [4].

1972’de IEEE Bilgisayar Topluluğu Yazılım Mühendisliği konusunda ilk süreli yayını çıkartmış: Transactions on Software Engineering

1993 yılında hem IEEE’nin Bilgisayar Topluluğu hem de ACM, YM konusunda kurullar oluşturuyor, bu kurulların amacı

1. YM’de kullanılan terimleri belirlemek ve tanımlamak
2. YM’nin standartlarını tanımlamak
3. YM eğitimi ile ilgili olan konuları tanımlamak

1994 başında bu iki kurul birleşmiştir ve YM’leri hangi kriterlere ve normlara uygun olarak yetiştirilmesi ve sertifikasyonu nasıl yapılması konuları üzerinde yoğun olarak çalışmıştır. (eğitim kriterleri ve normları)

Aşağıdaki konularda çalışacak 3 Çalışma Grubu kurulmuştur:

ÇG-1: YM'nin edinmesi gerekli bilgileri saptama
ÇG-2: Meslek etiği ve standartlarını belirleme
ÇG-3: Eğitim programı tanımlama ve sürekli meslek içi eğitim olanakları

1968 yılında düzenlenen bu konferanstan bugüne kadar Yazılım Mühendisliği disiplini çok gelişme kaydetmiş ve ilerlemiş olmakla beraber, günümüzde hala diğer mühendislik dalları ile eş tutulmamaktadır [5].

Mayıs 2001'de yayınlanan bir raporla YM'nin içeriğinin sürekli değiştiği ve edinilmesi gereken bilginin durağan olmadığı vurgulanmıştır.

Bununla birlikte, yazılımın hayatımızdaki artan önemi ve ülke ekonomilerine etkileri, yazılım mühendisliğine olan ilgiyi hem akademik çevrelerde hem de endüstriyel platformlarda artırmıştır. Son yıllarda yazılım geliştirmeye önemli katkıları olan yazılım geliştirme metodolojileri, programlama paradigmaları, programlama dilleri ve araçlar geliştirilmiştir.

Günümüzde yazılım geliştirme alanında milyonlarca kişi çalışmakta, yazılım mühendisliğinde yer alan çeşitli konular üzerinde çeşitli konferanslar düzenlenmekte ve yazılım mühendisliği üzerine eğitim programları bulunmaktadır. Bütün bu gelişmelere paralel olarak yazılım mühendisliğinin geçmişe oranla daha iyi tanındığını ve diğer meslek dalları arasındaki yerinin belirginleştiğini söyleyebilir. Ancak yazılım mühendisliği çevrelerinin üzerinde birleştiği bir konu, yazılım mühendisliğinin halen "genç" bir disiplin olduğu ve yeterli olgunluğa ulaşması için birçok konuda fikir birliğinin sağlanması için zamana gereksinim olduğudur [6].

Yazılım mühendisliğinin olgunlaşma sürecini hızlandırmak için yakın geçmişte, uluslar arası meslek kuruluşları olan Association for Computing Machinery (ACM) ve IEEE Computer Society (IEEE CS), "Yazılım Mühendisliği Koordinasyon Komisyonunu" oluşturmuş ve yazılım mühendisliği alanında çeşitli projelere başlamıştır [7].

2.1.1. Yazılım mühendisliğinin konuları

Yazılımca Karşılanan Gereksinimler : Uygulama ortamlarında bir sorunun çözümü için yazılım tarafından yerine getirilmesi istenen her bir özellik yazılımca karşılaşılan gereksinim olarak tanımlanır. (Açık biçimde anlaşılır tanımı yapılmalıdır.)

Yazılımın Tasarımı: Tasarım, yazılım yaşam döngüsünün tümü ile ilgilidir. Tasarımla yazılıma öyle bir yapı kazandırılmalıdır ki gereksinimlerin ve ileride yapılması gerekecek değişikliklerin yazılım içine uygun olarak yansıtılması sağlanmalıdır.

Yazılımın Gerçekleştirimi (Kurulumu): Yazılım birimlerinin kodlanması, geçerlenmesi ve sınanmasıdır.

Yazılımı Sınama: Sınırlı bir sınama veri kümesi ile yazılımın kendinden beklenen davranışsal nitelikleri devingen olarak (çalışma zamanında) gösterdiğinin anlaşılması gerekir. Veriler uygulama alanının gerçek verileri arasından seçilir.

Yazılımın Bakımı (Ayakta Tutma) : Uygulama başladıktan sonra daha önce fark edilmemiş anormallikler ile karşılaşılabilir, uygulama başladıktan sonra işletim ortamında yeni donanımsal alt yapılar kullanıma girecek olabilir, ya da kullanıcının yeni gereksinimleri için istekleri söz konusu olabilir, bütün bu durumlara yazılımın uyarlanabilmesi için bakımı yapılmalıdır. Bakım aşamasına yazılımın teslimi ile geçileceği sanılabilir, ancak bakım etkinlikleri çok daha önceden başlar.

Yazılımı Yapılandırma ve Yapılandırma Yönetimi: Yazılım yaşam döngüsü boyunca yazılımın bütünlüğünü, tutarlılığını, bakımını (günlenmesini) yapmak üzere kullanıldığı her yerde sistemli biçimde yapılanma özelliklerinin izlenmesidir.

Yazılım Mühendisliği Süreci ve Süreç Yönetimi: Sürecin tanımlanması, uygulama, ölçümler yapma, yönetme, süreç metodolojisinde değişiklikler ve geliştirmenin yapılmasıdır.

Yazılım Mühendisliği Araçları ve Yöntemleri: Yazılım geliştirme ortamları ve dayandıkları geliştirme yöntemine ilişkin bilgiler, bilgisayara dayalı araçlar, yazılım geliştirme sürecinin desteklenmesi, sistemli biçimde uygulanan etkinliklerin yararının belilenmesidir.

Yazılım Kalitesi : Bütün bir yaşam döngüsü sırasında etken olan bir konudur.
Kalite kavramı, YM etkinliklerinin hepsinde yer alması gereken, sürecin bütünü ile ilgilidir (bir şemsiye etkinlik) [34].

2.2. Yazılım Güvenliği

Hangi kaynaktan geldiği belirsiz kodların bilinçsizce çalıştırılması sonucu her gün binlerce sistem milyonlarca dolarlık hasara uğramaktadır.

CERT koordinasyon merkezinin verilerine göre 2003 yılında, 2002 yılındakine oranla %70 artış gösteren yazılım zayıflıkları ve açıkları raporlanmıştır. 1970 yıllarında orijinal bilgisayar güvenliği savunma stratejisi "nüfuz et ve yama." terimleriyle ifade ediliyordu.

O zamanlar savunma tamamen, ancak saldırı tespit edildikten, hasar oluştuktan sonra reaktif(tepkimeli)di. "Nüfuz et ve yama"'yı real-time saldırı tespit sistemleri, denetleme toolları gibi daha gelişmiş savunma teknikleri takip etti.

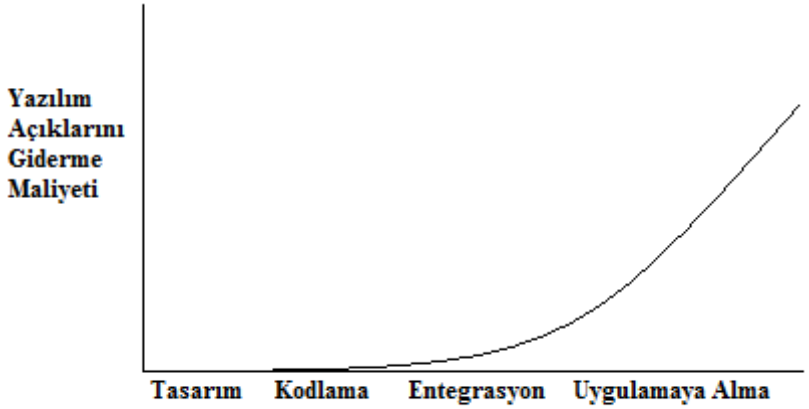
Yazılım güvenliği ve güvenli yazılım geliştirme sürecinde güvenlikle alakalı önlemlerin ne kadar erken alınmaya başlandığına bağlı bir gelişim göstermektedir.

2.2.1. Güvenli yazılım geliştirme

Yazılımların yaygın olarak kullanılmaya başlandığı ilk yıllarda kaliteli ve olgun yazılım üretmek, son yıllarda ise özellikle güvenli yazılım geliştirmek için çok sayıda model ve çerçeve üzerinde çalışılmıştır. Bu durumun en büyük tetikleyicisi son yıllarda güvenlik açıklıklarının artmasıdır. Güvenli bir sistem oluşturabilmek güçtür. Sistemlerdeki güvenlik açıkları yüzünden birçok sistem hasara uğramaktadır.

Bu açıklıkların kaynağı donanım ve yazılım olmak üzere iki temele dayanır. Günümüzde donanımsal hatalar en aza indirgenmiş durumdadır. Ortaya çıkan güvenlik açıkları büyük oranda yazılımsaldır.

Artan bu güvenlik tehditleri Şekil 1'de görüldüğü üzere hiç hesaba katılmayan sürpriz maliyetleri de beraberinde getirmektedir. Yazılım geliştirmede erken bir süreçte farkına varılan yazılım açıklıklarının düzeltilmesinin daha ileri süreçlerde farkına varılan açıklıklara göre daha az maliyetli olacağı yazılım endüstrisince yaygın olarak kabul edilen bir ilkedir [24]. Bu ilke yazılım geliştirme sürecinin güvenli olmasının maliyet açısından da ne denli önemli olduğunun göstergesidir.



Şekil 2.1. Yazılım süreçlerinde yazılım açıkları giderme maliyeti

Yazılım güvenliği kavramı ile ilgili yapılan en önemli yanlış güvenliği sadece kodun güvenliği ve ek olarak da yetkilendirme güvenliği ile sınırlandırmaktır. Hâlbuki yazılım güvenliği kavramını "güvenilir bilişim" (trusted computing) kavramı ile yakından ilişkilendirmek gerekmektedir. "Trusted Computing Group" tarafından konmuş olan güvenilir bilişim kavramı gizlilik, bütünlük, erişebilirlik ve kurtarılabilirlik olmak üzere dört temel kavram üzerinde durmaktadır.

Güvenli yazılım geliştirme süreçlerinde ayrıca değişiklik ve konfigürasyon yönetimi, geliştirme, test ve üretim ortamı ayrışımı, geliştirme ortamında gerçek verilerin kullanılmaması, üretim ortamına almadan önce kod incelemesi, güvenli programlama teknikleri kullanımı, uygulama güvenlik duvarı kullanımı ya da kaynak kod inceleme

hizmeti alınması gibi çalışmaların yapılması da güvenliğe ayrıca katkı sağlayacaktır [25].

Güvenli yazılım geliştirme sürecinde ele alınması gereken temel olarak dokuz ana güvenlik konusu vardır:

2.2.1.1. Girdi geçerleme (Input validation)

Günümüzde bilinen ve gelecekte de muhtemel tehditlerin çoğu kötü niyetli girdi ile başlamaktadır. Bununla birlikte; basit girdi geçerleme yöntemleri ile büyük güvenlik tehditlerinin önlenmesi mümkündür.

Girdi geçerleme yöntemlerini "beyaz kutu" ve "kara kutu" olmak üzere ikiye ayırmak mümkündür. Beyaz kutu yönteminde bilinen bir şablon girdi olarak kullanılmakta, bu şablonun dışındaki tüm girdiler kötü niyetli olarak kabul edilmektedir. Şablonun kontrolü çok kolay olduğundan bu yöntem oldukça etkili bir yöntemdir. Kara kutu yöntemi ise daha az etkili olmasına rağmen daha çok tercih edilen bir yöntemdir. Bu yöntemde kullanılan belirli bir şablon yoktur, sadece bilinen saldırıların bir listesi mevcuttur. Eğer girdi bilinen bir saldırıya benziyor ise o zaman girdi reddedilecek, onun dışındaki tüm girdiler ise kabul edilecektir. Bugün bile tüm atak çeşitlerini belirlemek zor iken gelecekteki atakları bilip filtrelemek daha da zor olacağından bu yöntemin etkinliğinin az olduğu açıktır. Dolayısıyla veri yapıları, mümkün olduğunca belli bir şablona uygun tasarlanarak geçerleme daha güçlü kılınmalıdır.

İstemci-sunucu uygulamalarında geçerleme hem istemci hem de sunucu tarafında yapılabilmektedir. Bununla birlikte; bir saldırgan istemci tarafındaki geçerleme kontrolünü kolay aşabileceğinden istemci tarafındaki geçerleme hiçbir zaman yeterli bir güvenlik önlemi olarak ele alınmamalıdır. Bunun yerine daha çok sunucu tarafında geçerleme kontrolü yapılarak güvenlik seviyesi arttırılmalıdır. Kısaca güvenilir olmayan bir kaynaktan (örneğin kullanıcıdan) gelen veri mutlaka onaylanmalıdır.

2.2.1.2. Kimlik doğrulama (Authentication)

Kimlik doğrulama, varlıkların (kullanıcı, cihaz veya bir uygulama) kimlik kontrolünden geçmesi işlemidir ve farklı kimlik doğrulama yöntemleri bulunmaktadır. Genellikle yazılımlar önceleri sadece kullanıcı adı ve şifre kullanması şeklinde zayıf doğrulama yöntemleri kullanılmakta idi. Eğer bir "domain" yapısı varsa, kullanıcılar "Active Directory" kullanılarak doğrulanmakta, "domain" dışında ise kimlik yönetimine ilişkin veritabanı uygulanmaktadır. Daha güçlü doğrulama yöntemleri olarak da biometrik metotlar veya akıllı kartlar kullanılmaktadır. Bir diğer doğrulama yöntemi ise üçüncü bir tarafın doğrulama işini yapması ve bu üçüncü tarafa güven duyulması şeklindedir.

2.2.1.3. Yetkilendirme (Authorization)

Kullanıcıların tanımlanması aşaması olan kimlik doğrulamadan sonra kullanıcının kimliği doğrultusunda erişim haklarının belirlendiği ve kontrolünün gerçekleştiği aşama yetkilendirmedir. Hangi yetkilerle işlem yapılacağını belirlemek için birçok yöntem bulunmaktadır.

2.2.1.4. Konfigürasyon yönetimi (Configuration management)

Konfigürasyon, uygulama ile ilgili hassas bilgileri içermektedir. Örnek vermek gerekirse veri tabanına erişim için gerekli bağlantı bilgilerini içeren dosyalar bu kapsamdadır. Konfigürasyona müdahale uygulamanın işleyişini değiştirebilir veya çalışmamasına sebep olabilir. Konfigürasyon dosyalarının sunucularda saklanıyor olması yeterli güvenlik önlemlerinin alındığı anlamına gelmemektedir. Konfigürasyon dosyaları hassas bilgi olarak nitelendirilmeli, şifrelenmiş bir şekilde tutulmalı ve bu dosyalara erişim kayıt altında tutulmalıdır.

2.2.1.5. Hassas bilgi (Sensitive information)

Hassas bilginin ne olduğunun belirlenebilmesi için uygulamanın ve işin bir arada ele alınması gerekir. Uygulama geliştirici işin niteliğini tam olarak bilemediğinden, diğer

yandan işin sahibi de uygulamanın teknik altyapısı hakkında sınırlı bilgiye sahip olacağından bu iki taraf tek başlarına hassas bilgi için yeterli tanımlama yapamayacaklardır. İki tarafın bir araya gelmesiyle hassas bilgileri içeren bir liste oluşturulmalı ve bu listeyi koruyacak bir politika oluşturulmalıdır.

2.2.1.6. Kriptografi (Cryptograhy)

Veriyi korumanın yollarından biri de şifrelemedir. Bugün şifreleme çalışmaları oldukça ilerlemiş, bilgisayarlar oldukça gelişmiştir. Fakat bu durum saldırganlar için de geçerlidir. Hassas bilgiler bilinen ve test edilmiş şifreleme yöntemleri ile saklanmalıdır. Ayrıca daha önce kırılması uzun zaman alan algoritmalar günümüzde daha kısa zamanda çözülebilmektedir. Dolayısıyla uygulama içindeki algoritmalar zamanla gözden geçirilmeli ve güncellenmelidir.

2.2.1.7. Parametre manipülasyonu (Parameter manipulations)

Dağıtık algoritmalar modüller arasında parametre gönderirler. Eğer bu parametreler arada değiştirilirse, saldırı gerçekleştirilmiş olur. 1 dolara satın alınan Ferrari bu duruma bir örnektir. Borcun belirlenmesi için web formu kullanan uygulama bu formdaki rakamın http Proxy kullanılarak manipüle edilmesi sonucu değer 1 dolara olarak değiştirilmiştir.

2.2.1.8. Hata yönetimi (Exception management)

Bazı teknolojiler hataları kullanarak hata yönetimi gerçekleştirmektedirler. Hatalar geliştiriciler ve sistem yöneticileri için uygulama ile ilgili birçok önemli bilgi ihtiva ettiği için çok önemlidirler. Bununla birlikte; geliştirici için bu derece önemli olan bilgi kullanıcı açısından problem oluşturabilmektedir. Her ne kadar kullanıcılar bu hataların ne demek olduğunu anlamasalar da saldırganlar için büyük ipuçları, yazılımla ilgili önemli bilgiler içermektedir. Bundan dolayı sadece genel bir hata mesajının dönmesi, hataların kayıt altında tutulması ve gerçek hataya sadece yöneticiler ulaşmasını sağlayacak sürecin oluşturulması gerekmektedir.

2.2.1.9. Kayıt tutma ve denetim (Logging and auditing)

Uygulama veya uygulamanın yöneticileri saldırı altında olduklarını anlamalıdır. Bu durum aslında neyin normal neyin anormal olduğunun belirlenmesi ile sağlanır. Bir uygulamaya ilişkin normal süreç ve şablon tanımlanmalı ve bunu dışında bir olay olduğunda saldırı ihtimali ele alınmalıdır. Örneğin, normal senaryoda bir uygulamaya dakikada ortalama beş kişinin erişmesi beklenirken bu sayı bine ulaşıyorsa muhtemelen bir "Servis Dışı" bırakma atağı söz konusudur.

Yukarıdaki ve bunlara benzer onlarca tehdit güvenilir uygulamalar geliştirmek için yazılım geliştirme sürecinin güvenliğinin yönetilmesinin büyük önem arz etmekte olduğunu gözler önüne sermektedir.

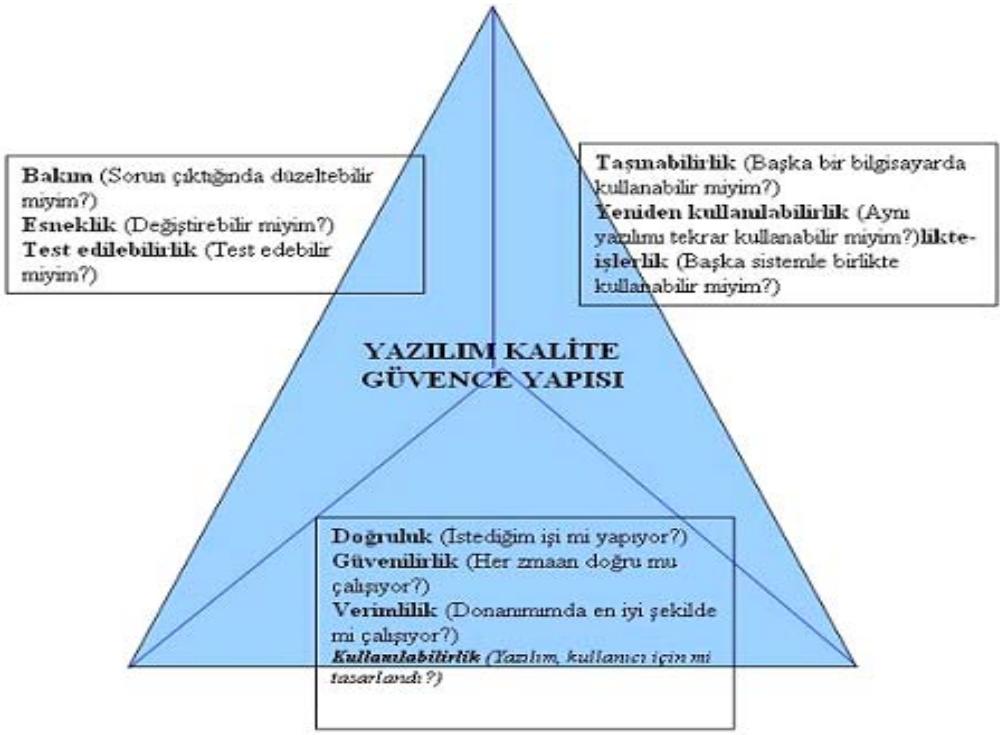
2.2.2. Türkiye'deki şirketlerde en sık rastlanan güvenlik açıkları

Hatalı Kablosuz Ağ Yapılandırması
Hatalı Yapılandırılmış Sanal Özel Ağ (VPN) Sunucuları
Web Uygulamalarında SQL Sorgularının Değiştirilebilmesi
Web Uygulamalarında Başka Siteden Kod Çalıştırma
Kolay Tahmin Edilebilir Şifrelere Sahip Kullanıcı Hesapları
SNMP Servisi Kullanımı
Güncellemeleri Yapılmamış Web Sunucusu
İşletim Sistemi ve Uygulamaların Standart Şekilde Kurulması
Hatalı Yapılandırılmış Saldırı Tespit Sistemleri
Güvenlik Duvarı Tarafından Korunmayan Sistemler

2.2.3. Yazılım kalite ve güvenlik aktiviteleri

YKG aktivitesi yazılım sürecinin başından sonuna kadar uygulanması gereken bir şemsiye aktivitesidir. Yazılım kalitesi kod ortaya çıktıktan sonra üzerinde düşünülmesi gereken bir olgu değildir. Şekil 2.2'de görüldüğü gibi YKG aktiviteleri kaliteye bağlı yönetim yaklaşımını desteklemek, etkin yazılım mühendisliği metodlarını öğretmek, formal teknik görüşmelerler düzenlemek, test stratejilerini

oluşturmak, yazılım dokümanları kontrol etmek, yazılım geliştirme standartlarına ne kadar uyulduğunu belirlemek ölçmek ve raporlamak amacıyla oluşturulmuştur.



Şekil 2.2. Yazılım kalite güvence yapısı [8]

Yazılım Yönetimi, planlama, kontrol yazılım projesini yönlendirme aktivitelerini kapsar. Yazılım Mühendisliği, gereksinimlerin analizi, tasarım geliştirme, kodlama, veritabanlarını yaratma aktivitelerini kapsar, yönetim ve mühendislik aktivitelerinin tüm gereksinimleri karşıladığını garanti eder [8].

2.2.3.1. Kalite güvence hedefleri

Yazılım geliştirme, diğer tüm karmaşık geliştirme aktiviteleri gibi risklerle doludur. Bu riskler, teknik veya program ile ilgili olabilir. Bu da, yazılımın beklenildiği gibi olmasını engelleyebilir.

KG'nin hedefi, bu riskleri azaltmak, CIA üçgeninin tamamlanmasını sağlamaktır. Örneğin, kodlama standartları yazılan kod kalitesini belirli bir seviyede tutmak için gereklidir. Eğer herhangi bir standart yoksa oluşan kodun, yeniden kullanılabilirlik

ile ilgili gereksinimlere uygun olmaması riski vardır ve bu da kodun üstünde tekrar çalışma yapmayı gerektirir.

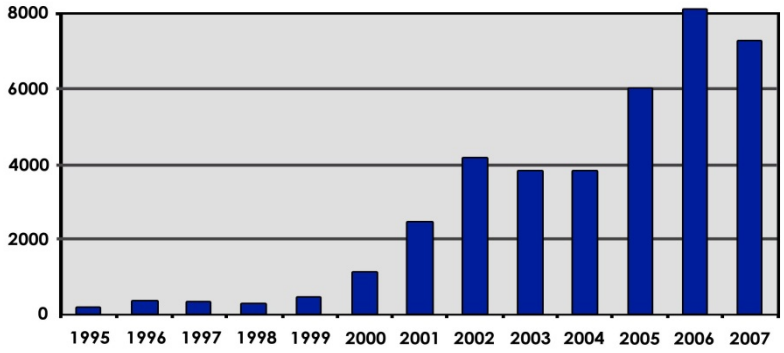
2.3. Yazılım Geliştirme Sürecinin Tarihçesi

Bilgisayarların ilk ortaya çıkmasıyla birlikte yazılım geliştirme süreci de başlamıştır. Bu süreç 1940'lı yıllara kadar gitmektedir. İlk yıllarda geliştirilen yazılımlarda görülen en büyük eksiklik yazılım projelerinin zamanında tamamlanamaması ve istenilen kalitede (dokümantasyon, fonksiyonellik, harcanan fazla iş gücü) olmamasıdır. Bunlara ek olarak teknolojinin hızla değişmesi, birçok yazılımın hız ve fonksiyonellik eksikliklerinden dolayı devre dışı kalmasına ve işin yeniden projelendirilmesine neden olmuştur. 1980'li yıllarda Fred P. Brooks "No Silver Bullet" [12] makalesi ile bu konuda tek parametre ile başarı sağlamanın mümkün olmadığın ifade etmiştir.

Bu problemlerin tespit edilmesinden sonra yeni diller (C++, Java vb.), yeni araçlar, yeni metodolojiler ve yeni disiplinler geliştirilmeye çalışılmıştır. İlk önceleri yazılım projelerinin zamanında, istenilen iş gücü ile belli bir disiplin içerisinde geliştirilmesi için çalışmalar yapılmıştır. İnternetin yaygın olarak kullanılmaya başlanması ile yazılımlar içerdikleri güvenlik zayıflıkları/açıklıkları nedeniyle kötü niyetli veya bilinçsiz kullanıcılar tarafından kötüye kullanılmaya başlanmış ve e-ticaret, e-devlet, e-sağlık gibi günlük hayatımızdaki işlerin yazılımlar aracılığı ile güvenli olarak sağlanması riskli olmaya başlamıştır.

Şekil 2.3'de CC CERT (Coordination Center Computer Emergency Response Team) tarafından yayınlanan yıllara göre açıklıkların sayısı görülmektedir. Şekil 2.3'den görüleceği gibi son yıllarda yazılımlarda görülen açıklıklar önemli oranda artmaktadır. Bu açıklıkların artma nedenlerinin başında internetin yaygın olarak kullanılması ve bilgisayarın iş uygulamaları da dâhil olmak üzere kullanım oranının çok artması gelmektedir Diğer önemli sebep yazılımların güvenliği dikkate alınmadan sadece fonksiyonellik göz önüne alınarak geliştirilmesidir. Bu durumda yazılımlar birçok açıklık içermektedir. Ayrıca yazılımın ortaya çıkmasından sonra tespit edilen açıklıkların kapatılması için çok fazla iş gücü ve zaman harcanmaktadır.

Çünkü bir güvenlik açıklığı yazılım geliştirme evresinde ne kadar erken tespit edilebilirse, o açıklığın kapatılması maliyeti o kadar az olacaktır.



Şekil 2.3. Yıllara göre açıklıklar [12]

Yazılım geliştirme sürecinin yaygın olarak kullanılmaya başlandığı ilk yıllarda kaliteli ve olgun yazılım üretmek için, son yıllarda ise özellikle güvenli yazılım geliştirmek için çok sayıda model ve anaçatı üzerinde çalışılmıştır. Bu süreç CMM (Capability Maturity Model) ile başlamış daha sonrasında CMMI (Capability Maturity Model Integration), FAA–iCMM (Federal Aviation Administration integrated Capability Maturity Model), Trusted CMM, SSE-CMM (Security System Engineering CMM), Microsoft Security Development Lifecycle, OpenSAMM (Software Assurance Maturity Model) modelleri gibi modeller geliştirilmiştir.

Yazılım/Donanım ve sistem güvenliği değerlendirmesi konusundaki ilk çalışmalar, Orange Book olarak da bilinen TCSEC (Trusted Computer System Evaluation Criteria) standardının 1983 yılında Amerika Birleşik Devletleri Savunma Bakanlığı (USA Department of Defense) tarafından yayınlanması ile başlamıştır. 1980'li yıllarda Avrupa'da; İngiltere, Almanya, Fransa ve Hollanda kendi güvenlik test metodolojilerini oluşturmuşlardır. Daha sonra bu ülkeler değerlendirme standartları arasındaki farkları ortadan kaldırmak ve bir yerde yapılan değerlendirmenin her yerde geçerli olmasını sağlayabilmek için 1991 yılında ITSEC (Information Technology Security Evaluation Criteria) standardını oluşturmuşlardır. Kanada'da ITSEC ve TCSEC'den faydalanarak 1993 yılında milli değerlendirme standardı CTCPEC (Canadian Trusted Computer Product Evaluation Criteria)'i yayınlamıştır.

Avrupa, Kanada ve Amerika Birleşik Devletleri'nde üretilen yazılım/donanım ve sistemlerin farklı standartlara göre güvenlik değerlendirmelerinin gerçekleştirilmesi, uluslararası satılan ürünlere uygulanmış testlerin diğer ülkelerde anlaşılamamasına, yazılım/donanım ve sistem güvenliği konusundaki çalışmaların farklı ülkeler farklı şekilde geliştirilmeye çalışılması sorunlara sebep olmuştur. Bu sorunların önüne geçilebilmesi için Kanada, Fransa, Almanya, İngiltere, Avustralya, Yeni Zelanda ve Amerika Birleşik Devletleri 1996 yılında bir araya gelerek Ortak Kriterler (Common Criteria) standardı sürüm 1,0'ı yayınlamışlardır.

Ortak Kriterler Standardı ürün güvence değerlendirmesini EAL1 (Evaluation Assurance Level)'den EAL7'ye kadar 7 seviyede yapmaktadır. EAL1 seviyesinde fonksiyonellik, kullanıcı kılavuzları gibi üst seviye bilgiler kontrol edilmekle birlikte seviyeler arttıkça kontrol edilen bilgiler de (üst seviye tasarım, detaylı tasarım, kaynak kodu inceleme, açıklık analizleri, matematiksel olarak doğrulma vb.) artmaktadır. Hali hazırda standardın 3,1 sürümü kullanılmaktadır. Ortak Kriterler standardı 25 ülke tarafında kabul edilmiştir. Ortak Kriterler aynı zamanda ISO tarafından EN ISO/IEC 15408 standardı olarak yayınlanmıştır.

2.4. Yazılım Geliştirme Modelleri

2.4.1. Yetenek olgunluk modeli (CMM - Capability maturity model)

CMM, Amerikan Savunma Bakanlığı'nın isteği üzerine Carnegie Mellon Üniversitesi'ne bağlı Yazılım Mühendisliği Enstitüsü tarafından 1986 yılında geliştirilmeye başlanmıştır.

Yetenek Olgunluk Modeli (Capability Maturity Model) belirli mühendislik disiplinleri için referans olgunluk modeli sağlar. Bir organizasyon kendi pratik uygulamalarını muhtemel iyileştirmeleri sağlamak için model ile karşılaştırır. CMM özel süreçler (yazılım mühendisliği, sistem mühendisliği, güvenlik mühendisliği) için hedef aşamalar tanımlar. Fakat bu işlemler için rehber dokümanlar sağlamaz. Bu süreçler ne istendiğini tanımlar fakat nasıl olacağını tanımlamaz. "CMM tabanlı değerlendirmeler ürün değerlendirmesinin ya da sistem sertifikasyonunun yerini

alacağı anlamına gelmez. Daha çok organizasyon değerlendirmesi ile ilgilenir yani özel alanlardaki zayıflıkların iyileştirilmesine odaklanır" [13].

2.4.2. Tümleşik yetenek olgunluk modeli (CMMI - Capability maturity model integration)

CMMI (Capability Maturity Model Integration) uzun vadede organizasyonun iş performansının olgunluğunun artırmasına yardım etmektedir. CMMI, organizasyonların işlemlerinin yeteneğini ve olgunluğunu değerlendirmek için servis geliştirme, bakım, satın alma mekanizmaları için en iyi pratikleri sunmaktadır. Bu model tarafında içerilen geliştirme alanları, sistem mühendisliği, yazılım mühendisliği, tümleşik ürün ve süreç geliştirme, tedarikçi kaynakları ve satın alma alanlarıdır. CMMI, CMM'in üzerine geliştirilmiş olup sekiz yıldan beri kullanılmaktadır. CMMI geniş bir kullanım oranına sahiptir. Mart 2009'da Software Engineering Institute 3446 organizasyon ve 21141 projenin CMMI kullandığını açıklamıştır [14]. CMMI'da süreç iyileştirme ve değerlendirme dört kategoride incelenir. Bunlar Proje Yönetimi, İşletme Yönetimi, Mühendislik ve Destektir. CMMI süreçleri içerisinde doğrudan güvenlikle ilgili bir süreç yoktur.

2.4.3. Federal havacılık yönetim tümleşik yetenek olgunluk modeli (FAA-iCMM Federal aviation integrated maturity model)

FAA-iCMM federal havacılık yönetiminde yaygın olarak kullanılır. FAA-iCMM modeli, dış kaynak kullanımı ve kaynak yönetimini de içeren büyük yazılım sistemlerinde (enterprise) ilerlemeler yapmayı hedefleyen en iyi pratiklerden oluşan bir model sunar. Son sürümü tümleşik büyük yazılım yönetimi, bilgi yönetimi kullanım/geçiş/devre dışı bırakma ve işlem/destek süreçlerini içerir. FAA-iCMM ISO 9001:2000, EIA/IS 731, Malcom Baldridge National Quality Award ve President's Quality Award Criteria, CMMI-SE/SW/IPPD ve CMMI-A, ISO IEC TR 15504, ISO/IEC 12207 ve ISO/IEC CD 15288 standart ve modellerini bütünleştirir.
CMMI'da olduğu gibi FAA-iCMM genel en iyi pratikleri içerir herhangi bir alan için doğrudan güvenliği hedeflemez.

2.4.4. Güvenli CMM/Güvenli yazılım metodolojisi (Trusted CMM/Trusted software methodology (T-CMM, TSM))

90'lı yılların başında Strategic Defense Initiatives (SDI) "Güvenli Yazılım Geliştirme Metodolojisi" olarak adlandırılan bir model geliştirdi. Daha sonra bu model güvenli yazılım metodolojisi (Trusted Software Methodology (TSM)) olarak adlandırıldı. Bu model düşük seviyelerde bilmeden yapılan geliştirici hatalarına karşı yüksek seviyelerde ise bilerek yapılan zararlı yazılım ataklarına karşı direnç sağlayan seviyelerden oluşmaktadır. TSM daha sonra CMM ile birleştirilmiş (Harmonize) ve Trusted CMM ortaya çıkmıştır [23]. TCMM/TSM günümüzde yaygın olarak kullanılmamasına karşın ilerde yazılım geliştirme sürecinde bir kaynak olarak kullanılabilir.

2.4.5. Sistem güvenlik mühendisliği yetenek olgunluk modeli (SSE-CMM-Systems security engineering capability maturity model)

SSE-CMM bir organizasyonun güvenlik mühendisliği yeteneğinin değerlendirilmesi ve geliştirilmesi için kullanılabilecek bir süreç modelidir. SSE-CMM, güvenlik mühendislik pratiklerini genel olarak kabul edilmiş mühendislik prensiplerine göre değerlendirip kabul edilebilir bir çerçeve ortaya koyar. Böyle bir çerçeve güvenlik mühendislik prensiplerinin uygulamalarında performansı ölçme ve iyileştirmeyi sağlar. SSE-CMM ISO/IEC 21827 standardı olarak yayınlanmış ve hali hazırda sürüm 3 yayınlanmıştır.

Güvenlik mühendisliği bu konuda genel olarak kabul edilmiş birkaç prensibe sahip olmasına karşın güvenliği değerlendirmek için gerekli bütüncül bir çerçeveden yoksundur. SSE-CMM, prensipleri uygulamalarının performansını ölçmeyi ve iyileştirmeyi amaçlayan çerçeveyi oluşturmayı amaçlar.

Model, proje ve organizasyon süreçleri ve güvenlik mühendisliği olmak üzere iki geniş alana sahiptir. Güvenlik mühendisliği mühendislik süreçleri, güvenlik süreçleri ve risk süreçleri içinde yer alır. Üç organizasyon içinde 22 süreç bulunmaktadır.
SSE-CMM The International Systems Security Engineering Association (ISSEA) tarafından sürdürülmektedir [23].

2.4.6. Microsoft güvenli geliştirme döngüsü (SDL)

Microsoft SDL güvenli geliştirme döngüsü, Microsoft geliştiricilerinin daha güvenli yazılımlar geliştirebilme ihtiyaçları ve bu ihtiyaçlara yönelik arayışlarından ortaya çıkmış bir anaçatıdır. Temelleri Ocak 2002'de yayınlanan Trustworthy Computing (TwC) yönergesine dayanır.

Bu model yazılım geliştirmenin başlangıcından itibaren güvenli yazılım geliştirme ve yaşam döngüsünü hedefler. Bu hedeflere ulaşmak için, eğitim ve farkındalık, proje başlangıcı, en iyi pratikleri tanımla ve uygula, risk analizi yap, risk analizi aracı, risk analizi, yazılım dokümantasyonu araçları ve müşteriler için en iyi patrikler, güvenli kodlama politikası, güvenli test politikası, güvenlik ekleme, son güvenlik kontrolü, güvenlik müdahale planlaması, ürün çıkarma güvenlik cevapları ve işletme olmak üzere on üç adımı kullanır [23].

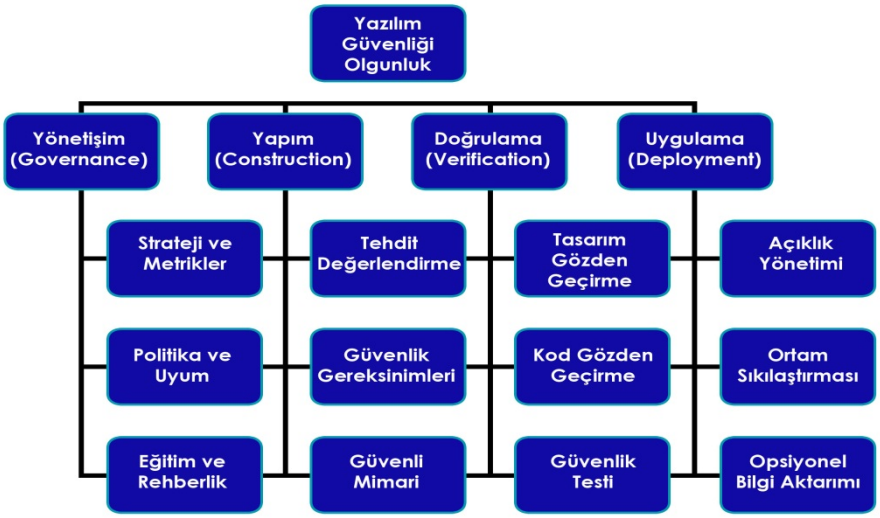
Bu modelin temel dezavantajı doğrudan proje yöneticisinin liderliğine bağlı olmasıdır.

2.4.7. OpenSAMM modeli

OWASP (Open Web Application Security Platform) tarafından desteklenen OpenSAMM projesi kapsamında yazılım garanti olgunluk modeli (software assurance maturiy model) ortaya konmuştur [8]. Bu çalışmada güvenli yazılım geliştirme amacıyla bir ana çatı oluşturulmaya çalışılmıştır. Ana çatı, büyüklükten bağımsız bir şekilde her organizasyonun kendine adapte edebileceği, normal yazılım geliştirme döngülerine uyarlayabileceği ve bir organizasyondaki yazılım güvenliği alanında gelişmeyi yönlendirebilecek bir yapıda oluşturulmuştur.

Anaçtı dört ana başlığa ayrılmıştır. Bu başlıklar, yönetişim (governance), yapım (construction), doğrulama (verification) ve uygulama (deployment) olarak belirlenmiştir. Bu ana başlıklar aslında normal yazılım geliştirme döngüsünün temel adımlarına karşılık gelmektedir. Her bir ana başlık altında üçer güvenlik eylemi yer almaktadır. Güvenlik eylemleri Şekil 2.3'de verilmiştir. Bu yapıda, güvenli bir yazılım geliştirmek için her bir temel yazılım geliştirme adımına karşılık yapılması

gereken güvenlik çalışmaları güvenlik eylemi olarak değerlendirilmiştir. Her bir eylem altında da organizasyonun o eyleme konu alan alandaki olgunluk seviyesine göre hedefler (objectives) belirlenmiştir.



Şekil 2.4. OpenSAMM güvenlik eylemleri

Yönetişim başlığı altında, organizasyonda uygulanacak yazılım güvenliği programının stratejik yönünün belirlenmesi, organizasyona ait güvenlik çalışmalarının performansını ölçme yöntemlerinin ortaya konması, belirlenmiş kurum içi standartlara ve kurum dışı düzenlemelere uyum sağlanması ve çalışanların yazılım güvenliği konusunda eğitilmesi ile söz konusu çalışanlara rehberlik yapılması konuları ele alınmaktadır.

Yapım başlığı, güvenli yazılım oluşturmak için yazılım gereksinimi ve tasarımı aşamalarında gerçekleştirilmesi gereken güvenlik eylemlerine değinmektedir. Yazılımın karşılaşacağı tehditlerin değerlendirilmesi, güvenlik ihtiyaçlarının belirlenmesi ve güvenli mimarinin oluşturulması "Yapım" başlığının altındaki güvenlik eylemleridir.

Doğrulama başlığı, tasarım, yazılım kodlama ve yazılım testleri aşamasında gerçekleştirilmesi gereken güvenlik gözden geçirmelerini ve güvenlik testlerini kapsamaktadır. Tasarım gözden geçirmesi, kod analizi ve güvenlik testleri bu kapsamdaki güvenlik eylemleridir.

Uygulama başlığı ise yazılımın canlı sisteme kurulması ve desteğinin verilmesi aşamalarını kapsamaktadır. İlgili güvenlik eylemleri, açıklık yönetimi, ortam sıkılaştırması ve operasyonel bilgi aktarımı olarak ele alınmıştır.

2.5. Yazılım Geliştirme Süreç Modeli

YGSM, herhangi bir yazılımın, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar yazılım geliştirme süreç modeli olarak tanımlanır [8].

Yazılım işlevleri ve ihtiyaçlar sürekli olarak değiştiği ve geliştiği için bir döngü biçiminde düşünülür. Yazılım yaşam döngüleri tek yönlü ve doğrusal olarak düşünülmemelidir. Döngü içerisinde herhangi bir aşamada geriye dönmek ve tekrar ilerlemek söz konusudur [28].

2.5.1. Temel adımlar

Analiz: Bir problemin çözümü olarak nitelediğimiz yazılımların ne yapacağını ve nasıl yapacağını belirlediğimiz, personel ve donanım ihtiyaçlarının çıkarıldığı, olurluk aşamasının yapıldığı, proje planının oluşturulduğu yani problemi tanımladığımız aşama planlama aşamasıdır. Yazdığınız kod ancak isteneni doğru bir biçimde yerine getiriyorsa başarılı bir yazılımdır. Bu nedenle öncelikle yazılımdan ne istendiğinin doğru bir biçimde tanımlanması gerekir. Analiz aşaması personel, donanım ve sistem gereksinimlerinin belirlenmesi, sistemin fizibilite çalışmasının yapılması, kullanıcıların gereksinimlerinin analizi, sistemin ne yapıp ne yapmayacağının kısıtlamalar göz önüne alınarak belirlenmesi, bu bilginin kullanıcılar tarafından doğrulanması ve proje planı oluşturulması adımlarından oluşur.

Çözümleme: Yazılım işlev ve ihtiyaçlarının ayrıntılı olarak çıkarıldığı aşamadır. Mevcut sistemde var olan işler incelenir, temel sorunlar ortaya çıkarılır, yazılımın çözümleyebilecekleri vurgulanır.

Tasarım: Çözümleme aşaması sonucunda belirlenen gereksinimlere yanıt verecek yazılımın temel yapısının oluşturulduğu aşamadır. Yazılım tasarımı, bir bileşen veya

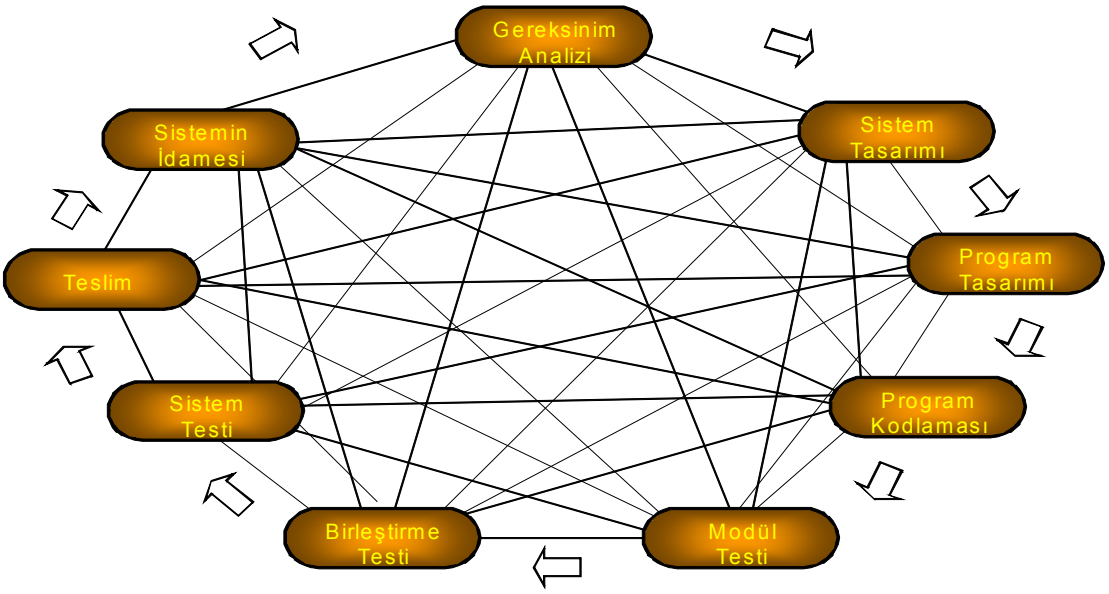
sistemin nasıl gerçekleştirileceğini belirlemek için kullanılan teknikler, stratejiler, gösterimler ve desenlerle ilgilidir. Bu aşama yazılım bileşenleri arasındaki içsel ara yüzler, mimari tasarım, veri tasarımı, kullanıcı ara yüzü tasarımı, tasarım araçları ve tasarımın değerlendirilmesi alt süreçlerini de kapsamaktadır. Tasarım aşaması, yazılımın hem kullanıcı ara yüzünü hem de programın omurgasını ortaya koymaktadır. Yapılacak tasarım, yazılımın işlevsel gereksinimlere uygun olmasının yanı sıra kaynaklar, performans ve güvenlik gibi kavramları da göz önüne alınarak gerçekleştirilmelidir.

Mantıksal Tasarım: Önerilen sistemin yapısı anlatılır. (akış şemaları)
Fiziksel Tasarım: Yazılımı içeren bileşenler ve bunların ayrıntıları. (ekran tasarımları)

Gerçekleştirim: Kodlama, sınama ve kurma aşamalarının yapıldığı aşamadır. Kodlama aşaması, tasarım sürecinde ortaya konan veriler doğrultusunda yazılımın gerçekleştirilmesi aşamasıdır. Bu süreç programlama çalışmalarının yanı sıra yazılımın geliştirilmesi ve kullanıcıya ulaştırılması sürecindeki bütün çalışmaları kapsar. Tasarım sonucu üretilen süreç ve veri tabanının fiziksel yapısını içeren fiziksel modelin bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışması olarak da nitelendirilebilir [5]. Yazılım geliştirme ortamı, programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları seçimi kodlama aşamasında gerçekleştirilir.

Test: Test aşaması, yazılım kodlanması sürecinin ardından gerçekleştirilen sınama ve doğrulama aşamasıdır. Elde edilen uygulama yazılımının hem belirlenen gereksinimleri sağlayıp sağlamadığı hem de gerçekleştirimin beklentilere uygun olup olmadığını kontrol etmek için statik ve dinamik sınama tekniklerinden yararlanır. Statik teknikler, yazılımın tüm yaşam döngüsü boyunca elde edilen gösterimlerin analizi ve kontrolüyle ilgilenirken, dinamik teknikler sadece gerçekleştirilmiş sistemi içerir. Yazılım üretiminde ilk testler genelde geliştirme sürecinde programcı tarafından yapılır. Bununla birlikte, asıl hata ayıklama ve geribildirim hizmeti test ekipleri tarafından yapılır. Testler ve geribildirim müşteri yazılımı kullandığı sürece devam eder. Test sürecinde en faydalı geribildirimler son kullanıcı test gruplarından gelir.

Bakım: Yazılımın tesliminden sonra hata giderme ve yeni eklentiler yapma aşamasıdır. Yazılımın kullanıma başlanmasından sonra yazılımın desteklenmesi sürecini kapsar. Yazılımın eksiklerinin giderilmesi, iyileştirilmesi gibi alt aşamaları içeren aşamadır. Yazılım var olduğu sürece sürer [3].



Şekil 2.5. Gerçek hayatta program geliştirme

2.5.2. YG sürecinin genel özellikleri ve içinde yer alan etkinlikler

Genelde dayanılan model ne olursa olsun başarılı olabilmek için sürecin çatısını oluşturacak etkinlikler içinde aşağıdaki anahtar roldeki konuların nasıl karşılanacağının yanıtları bulunmalıdır:

- Yazılım geliştirme sürecinin denetimini sağlayacak proje yönetimi
- teknik olarak yöntemlerin uygulanış biçimi
- Her bir etkinliğin ara adımları
- Her bir etkinliğin ara ürünleri olarak (her aşamada yapılan tanımları sağlayan modellemeler, belgelemeler, veri tanımları, raporlar, formlar vb.) nelerin elde edileceği
- Kalitenin nasıl sağlanacağının yolu, yordamı

- Değişiklik isteklerinin nasıl derleneceğinin, nasıl ele alınacağının, nasıl yapılacağının, nasıl uygulama ortamlarında kullanılan yazılımlara aktarılacağının yöntemi [21].

En genelde Yazılım Geliştirme Sürecinin 3 temel aşamada düşünmek gerekir:

Yazılımı Tanımlama Aşaması: Ne istendiğinin belirlenmesi üstünde odaklanılır.

Yazılım Proje Yönetimi ve Gereksinim Çözümleme
Hangi veriler işlenmeli
Hangi bilgiler elde edilmeli
Hangi işlemler nasıl yapılmalı
Hangi işlevler görülmeli
Nasıl bir davranış göstermeli
Hangi ara yüzler kurulmalı
Tasarımı kısıtlayıcı özellikler nelerdir
Başarım hangi etkenlere bağlı olarak anlaşılacaktır

Yazılımı Geliştirme Aşaması: İstenenlerin nasıl sağlanabileceği üstünde odaklanılır.

Yazılım Tasarımı, Kod Üretimi ve Yazılım Sınama
Veriler nasıl düzenlenecek
İşlemsel ayrıntılar nasıl yapılacak
İşlevler hangi yazılım mimarisi içinde nasıl karşılanacak
Kullanıcı ve mimari yapı içindeki kesimler arasındaki arayüzler nasıl olacak
Tasarım hangi programlama dili/dilleri ile gerçekleştirilecek
Her bir düzeyde sınamalar nasıl yapılacak

Yazılıma Bakım Desteğinin Verilmesi Aşaması: Teslim sonrası istenecek değişiklikler üstünde odaklanılır.

Yazılım Bakımı; düzeltici, uyarlayıcı, iyileştirici, önleyici

Düzeltici Bakım: En iyi yazılım kalite güvencesi sağlayıcı etkinlikler gösterilse de yazılımda belirlenememiş kusurlar uygulama başladığında ortaya çıkar. Düzeltici

bakım işletime yeni girmiş yazılımda geç fark edilmiş kusurların giderilmesi için yapılır.

Uyarlayıcı Bakım: Zaman geçtikçe kullanılan bilgisayarların merkezi işlem birimi, işletim sistemi, iş görme kuralları, girdi / çıktı ortamları değişebilir. Yazılımların bu tür değişikliklere göre uyarlanması, üzerinde uygun değişikliklerin yapılması gerekir.

İyileştirici / Yetkinleştirici Bakım: Kullanıcı yazılımı kullandıkça yeni bir takım işlevler kazandırılmasını ya da bir işlevin farklı biçimde yerine getirilmesini isteyebilir. Geliştirici bakım ilk geliştirme sırasında olmayan gereksinimleri yazılıma kazandırmak için yapılır.

Önleyici Bakım: Yazılım üzerinde değişiklikler yapıldıkça tasarımla kazandırılmış yapısal niteliklerini yitirmeye başlar ve giderek kötüleşir. Bu yüzden yeniden mühendislik denilen bir çaba ile yapısının iyileştirilmesi gerekir. Bu yönde yapılan bakımla yazılımın tekrar kolayca değiştirilebilir, uyarlanabilir ve iyileştirilebilir bir biçime getirilmiş olur [15].

2.5.3. YG sürecindeki şemsiye etkinlikler

Aşağıdaki etkinlikler Yazılım Geliştirmede önemli rolleri olan şemsiye etkinlikler olarak bilinir:

Yazılım Projesi İzleme ve Denetim
Biçimsel Gözden Geçirmeler
Yazılım Kalite Güvencesi
Yazılım Yapılandırma Yönetimi
Yazılımın Belgelenmesi
Yeniden Kullanılabilirlik Yönetimi
Ölçümleme
Risk Yönetimi

Şemsiye etkinlikler yazılım geliştirme sürecinin başından sonuna tümünde uygulanır.

2.5.4. Yazılım geliştirmede izlenen süreç modellerinin yetkinlik ve olgunluğunu ölçümleme

Carnegie Mellon Üniversitesi Yazılım Mühendisliği Enstitüsü (Software Engineering Institute - SEI) tarafından yazılım geliştirmede izlenen süreçlerin olgunluğunu (uygunluğunu) belirlemede kullanılan bir model geliştirilmiştir. Bu model (CMMI – Capability Maturity Model Integration) yazılım geliştirme sürecine bakarak işi yapan firmanın yeteneğini ölçmeye yöneliktir.

5 farklı etkinlik düzeyi belirlenmiştir:

Düzey 1: Başlangıç Düzeyi – Kullanılan tanımlı bir yazılım süreci yoktur, geliştirme kişisel çabalara dayanır.

Düzey 2: Yönetilen Düzey – Maliyet kestirimi, iş-zaman çizelgesi gibi konularda temel sayılabilecek düzeyde proje yönetiminin yapıldığı düzey, önceden yapılmış benzer projelerde kazanılmış deneyimin yeni bir projeye aktarılmasını sağlayan bir yeterlik aranır.

Düzey 3: Tanımlı Düzey – Firmanın tamamında belgeleme, standartların kullanımı ve süreç aşamalarını bütünleştirme konusunda yönetsel bir yaklaşım ve yazılım mühendisliği etkinliklerinin tanımlı olması, yazılım geliştirme ve desteklenmesinde belli bir süreç modeline dayanma. Bu düzey için bir önceki düzeyin karşılanmış olması gerekir. (ISO 9001 belgesine sahip olmakla eşdeğerde görülebilir.)

Düzey 4: Nicel Olarak Yönetilen Düzey – Yazılım süreci ve ürün kalitesi ile ilgili olarak ayrıntılı ölçümlerin yapılmasını sağlayan bir olgunluk düzeyi, hem yazılım geliştirme süreci hem de ortaya çıkan ürünlerin niteliğinin anlaşılması ve denetlenmesi ayrıntılı ölçümlerle yerine getiriliyor olmalıdır. Bu düzey için bir önceki düzeyin karşılanmış olması gerekir.

Düzey 5: En İyileştirilen Düzey – İzlenen sürecin sürekli olarak geribildirimler ile iyileştiriliyor olması gerekir, ayrıca sınama yaklaşımlarının ve kullanılan teknolojik araçların gelişiminin yapılıyor olması beklenir. Bu düzey için bir önceki düzeyin karşılanmış olması gerekir.

Bir yazılım evinin yazılım geliştirme yeteneğinin hangi olgunluk düzeyinde olduğunun belirlenebilmesi için Yazılım Mühendisliği Enstitüsünün (Software Engineering Institute – SEI) geliştirdiği bir sorgulama anketinde derlenen yanıtlar değerlendirilir. Sorgulama anketinde her düzey için belirlenmiş geliştirme süreci içinde yer alan anahtar etkinlik alanları bulunur.
Her bir düzey içinde yerine getirilmesi aranan anahtar etkinlik alanları aşağıda gösterilmiştir:

Düzey 2

- Gereksinim yönetimi
- Yazılım projesi planlama
- Yazılım sürecini izleme - denetleme
- Alt yüklenici sözleşme yönetimi
- Ölçümleme ve çözümleme
- Süreç ve yazılım kalite güvencesi
- Yazılım yapılandırma yönetimi

Düzey 3

- Gereksinim geliştirme
- Teknik çözümler
- Yazılım bütünleştirme
- Doğrulama
- Geçerleme
- Sürece odaklanma
- Süreç tanımı
- Eğitim programları
- Bütünleştirilmiş yazılım proje yönetimi
- Risk yönetimi
- Karar çözümleme

Düzey 4

- Süreçsel performans
- Niceliksel proje yönetimi

Düzey 5

- Örgütsel buluş ve değişim
- Nedensellik çözümleme

Bu anahtar etkinlik alanları aşağıdaki açılardan değerlendirilir:

- Amaç: neden yapıldığına ilişkin tanım
- Öngörüler: amacın nasıl sağlandığı ve amacın sağlanmasının nelere bağlı olduğu
- Yetenekler: öngörülerin sağlanması için örgütsel ve teknik olarak sahip olunan olanaklar
- İç etkinlikler (ara adımlar): anahtar etkinlik alanı içinde yerine getirilmesi gereken işler
- Gerçekleştirimi izleme yöntemi: iç etkinliklerin nasıl planlandığı ve yerine getirilirken yapılanların nasıl izlendiği
- Yapılanları geçerleme yöntemi: anahtar etkinlik alanında yapılanların uygun biçimde yerine getirildiğini anlamanın yolu/ yordamı [29].

2.6. Süreç Olarak Yazılım

Kalitenin uygulanabilir olduğu yazılım geliştirme süreçleri hakkında, değişik görüşler vardır.

Bir görüşe göre iki aşamalı bir süreç tanımı yeterlidir:

Geliştirme (analiz/tasarım/program/test) ile geçiş ve sonrası (geçiş/işletim/bakım) süreçleri. Öte yandan daha detaylandırılmış süreçler de söz konusu olabilir: İhtiyaç analizi, sistem tasarımı, kodlama, test ve bakım süreçleri gibi. Ya da hedeflenen ürünün tanımı, yöntem geliştirme, uygun teknoloji araştırması ve seçimi, analiz, görsel tasarım, teknik tasarım, geliştirme, iç testler, dış testler ve pilot proje süreçleri sayılabilir. Proje yönetimi açısından ele alındığında bunlara maliyet tahmini, risk analizi, kaynak ayırma ve koordinasyon, süreç planlaması ve zamanlama süreçleri de eklenebilir.

Yazılımı, süreç olma özelliği içerisinde etkileyen en önemli kriterlerden birisi, onun entelektüel bir ürün olmasından ileri gelir. Onun bu özelliği, kullanıcıya yansıyan üst yapı (görsel ara yüz) ile performans ve programcıya yansıyan alt yapıyı etkileyebilecek güçtedir. Dolayısı ile yazılım geliştiren şirketlerde insan kaynakları politikasının sürekli eğitim, iş tatmini vb. unsurlarla beraber dikkatle planlanması ve yürütülmesi gerekmektedir.

Resmi ve gayri resmi standartlara (modeller) girmeden önce yazılım süreçlerinde kalitenin nasıl sağlanması gerektiği hakkındaki genel düşünceleri toparlarsak;
Kalite güvence, organizasyona bir fonksiyon olarak yerleştirilir. (kalite güvence grubu belirlenir, kaliteyle ilgili politika ve prosedürler saptanır, kalite güvence yönetimi seçilir, kalite kontrol sonuçları toplanır ve değerlendirilir, kalite kontrol sonuçlarına göre iyileştirme çalışmaları yapılır, iyileştirme sonuçlarına göre standartlar revize edilir)

Yazılım çevrim süresi azaltılır. (müşteri gereksinimleri önceden ve somut bir şekilde tespit edilir, tekrar kullanım artırılır, değişiklik yapmak azaltılır, süreçler iyileştirilir-basitleştirilir, çalışma ortamı ve ergonomide iyileştirmeler sağlanır, iş için en iyi personel kullanılır, sorunlara proaktif bir şekilde yaklaşılır, standartlar
kullanılır, kod ve algoritma karmaşıklığı engellenir)

Yazılımlarda sık kullanılan ve standart (popüler) fonksiyonlar kullanılır, bilgi, uyarı ve hata iletişimi, güvenlik (yetki), performans, diğer platformlarla ilişki kurabilme ve uyarlanabilme yeteneği (customizing) için uygun alt yapı oluşturulur [33].

BÖLÜM 3. YAZILIM GELİŞTİRME SÜREÇ MODELLERİ

Yazılım ile ilgilenen ve küçük-büyük projelerde yer alan-alacak olan herkesin bilmesi gereken modellerdir. Geçmişten günümüze kullanılmış olan modellerden kısaca bahsederek bunları bilmemizin bizlere ne gibi katkılar sağlayacağını, ne gibi getirileri olacağını daha iyi anlayabiliriz. Yazılım geliştirmek için kullanacağımız bu modeller planlı çalışmamızı ve geliştireceğimiz yazılımları en iyi biçimde geliştirmemizi sağlayacaktır [29].

Günümüz 'Yazılım Geliştirme' sürecinde hedeflenen amaçlar:

1) Üretim döngüsünde yazılım maliyetlerini azaltmak,
2) Üretilen yazılımın kalitesini arttırmak,
3) Üretici ve kullanıcı arasındaki iletişimi arttırarak istenilen seviyede ürün elde edilmesi

Şeklinde sıralanabilir.

Bu amaçlara ulaşılabilmesi için gerekli koşullar aşağıdaki gibidir:

1) Prosedür:

- Ne yapılması gerekiyor?

Burada yazılım geliştirme süreciyle ilgili nelerin yapılması gerektiği, bunların sonucunda sonuçların neler olacağı ve bu sonuçların detaylı analizinden sonra hangilerinin olmazsa olmaz (proje açısından kritik) olduğunun karar verilmesi aşamasıdır.

2) Metotlar:

- Amaca nasıl ulaşabiliriz?

Burada belirlenmesi gereken ana hususlar, ilk seviye aktiviteleri, bunların sonuçları ve iş sahiplerine yapılacak sunuş yöntemleri şeklindedir.

3) Araç Gereksinimleri

- Amaca ulaştıracak araçlar nelerdir?

Burada değişik karakteristik özelliklerine göre hangi araçların yazılım geliştirme sürecinde kullanacağı ve bunların maliyet/performans analizlerinin tamamlanması aşamasıdır.

Bütün bu yukarıdaki aşamalar seviyesinde ana aktivite alanları ise:

A. Yazılım Geliştirme
B. Kalite Kontrol
C. Konfigürasyon Yönetimi
D. Proje Yönetimi

Şeklindedir.

Yazılım geliştirilirken sürecin aşamaları ve adımlarından nasıl bir yaklaşım ile geçileceğini gösteren şimdiye kadar kullanılmış üreysel nitelikteki süreç modelleri şunlardır: V Süreç Modeli, Gelişigüzel Model, Barok Modeli, Araştırma Tabanlı Model, Helezonik (Sarmal) Model, Prototip Modeli, Hızlı Uygulama Geliştirme Modeli, Çağlayan (Şelale) Modeli, Artırımsal Model, Evrimsel Model ve Yeniden Kullanılabilir Yazılım Modeli [29].

3.1. V süreç modeli (V process model)

Yazılım geliştirme süreç modelleri açısından uygulanabilirliği en kolay olan ve gerektirdiği maliyetler açısından en uygun olan model V-Modeli'dir. V-Modeli yazılım geliştirme sürecini koordine eden bütünleşik ve birleştirici methodların toplamıdır. Ana aktivite alanlarının birbirleri arasında koordinasyonunu sağlayan, tüm aktiviteleri içerik bakımından tanımlayan, proje kapsamında üretilecek tüm yazılım hakkında dokümantasyon bilgileri içeren ve bu dokümantasyonlar hakkında detaylı açıklama getiren ve son olarak da her seviye arasında problemin tanımlandığı andan ürün sürümüne kadar geçen tüm aşamalarda içsel kontrollü bir metodolojiler toplamıdır.

Bu modelde adından da anlaşılacağı gibi "V" yapısında bir yol izlenir ve adımlar bu şekilde gerçekleştirilir. Bu yol üzerinde sol taraf üretimi sağ taraf ise test işlemini ifade eder.

Bu modelde yer alan çıktıları "Kullanıcı Modeli", "Mimari Model" ve "Gerçekleştirim Modeli" adı altında toplayabiliriz.

Kullanıcı modelinde geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.

Mimari modelde sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevler ele alınmaktadır.

Gerçekleştirim modelinde de yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar ele alınmaktadır. Bu model belirsizliklerin az iş tanımlarının belirgin olduğu bilişim teknolojileri projeleri için uygun bir modeldir. Ayrıca model kullanıcının projeye katkısını artırmaktadır [16].

3.1.1. V-Modelin avantajları

1) Sonuç ürünün iş sahipleri tarafından proje başında arzu edilen ürün olması
2) İçerdiği kaliteli teknik dokümantasyon sayesinde kişilerden bağımsız olması. (Bu sistem ile yazılım geliştirici mühendislerin herhangi bir sebepten dolayı gerek proje üretim aşamasında gerekse sürüm sonrası ayrılmaları durumunda mevcut dokümantasyon ile yeni başlayan mühendislerin hızlı adaptasyonu ile sağlanan kişilerden bağımsızlık)
3) Paralel yürüyebilecek işlerin aynı zamanlamalarla yapılması sayesinde işin kolektif metotlara göre çok daha hızlı sonuca ulaşması.
4) Proje sürecinin herhangi bir yerinde iş sahibi tarafından yeni gereksinim istenmesi durumunda bunların projeye kolayca entegrasyonunun sağlanması.
5) Geliştirme sırasında ortaya çıkabilecek sorun ve hususların çok daha önceden belirlenebilmesi.

6) Standartlara bağlı geliştirme süreci sayesinde programın uygulanabilirliğinin yüksek ve modüler yapıda olması.

7) Sorunsuz bilgi aktarımı sayesinde kolayca uygulanabilen Risk Yönetimi.

8) İlerideki düşünülebilecek yeni modüllerin mevcut tasarıma uygun ve kolay adapte edilebilir şekilde entegrasyonuna izin vermesi.

9) Halen yürürlükte olan EURO-METHOD ile birebir uygunluk arz ettiğinden Avrupa Birliği standartlarına uygunluk.

10) Avrupa Birliği yüksek güvenlik seviyeli yazılımlar standartlarına uygunluk.

11) Bakım maliyetlerinin en az seviyeye çekilmesi [16].

3.1.2. Yazılım uygulama geliştirme projelerine adaptasyon

V-Model uygulamalarının en önemli özelliklerinden biri de herhangi büyüklükte ve ölçekteki bir uygulama yazılım projesine adapte edilebilir olma özelliğidir.

Bu süreçte takip edilmesi gereken aşamalar ise:

1) V-Modelinin belirlenmesi,

2) İş sahipleriyle çalışılarak model hakkındaki tavsiyelerin ve çıkarımların belirlenmesi,

3) Proje biçimlendirme çalışmaları sırasında:

a. Gerekli aktivite ve ürünlerin seçilmesi,

b. Proje için gereken diğer ayarlamaların yapılması,

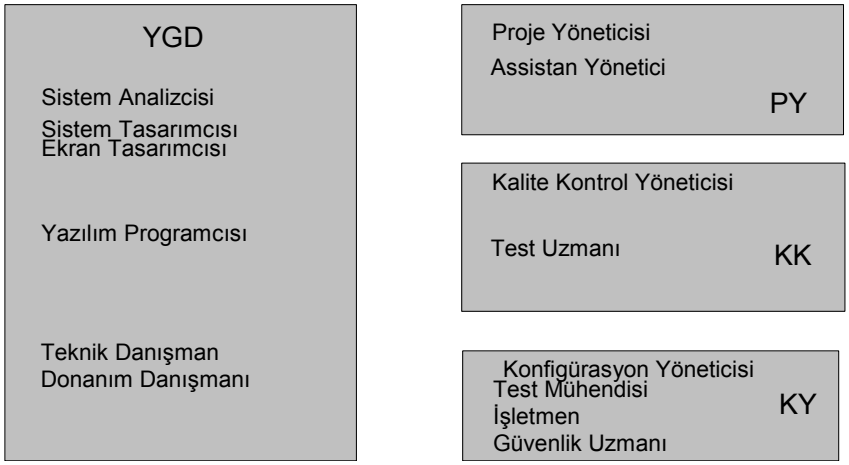
c. Proje rehberinin oluşturulması,

d. Teknik çalışmalar sonucu değerlendirme ve maliyet analizi yapılması,

4) Proje Planının hazırlanması

Şeklindedir [16].

3.1.3. V-Model dâhilindeki roller ve tanımlar:



Şekil 3.1. V-Model dâhilindeki roller ve tanımları

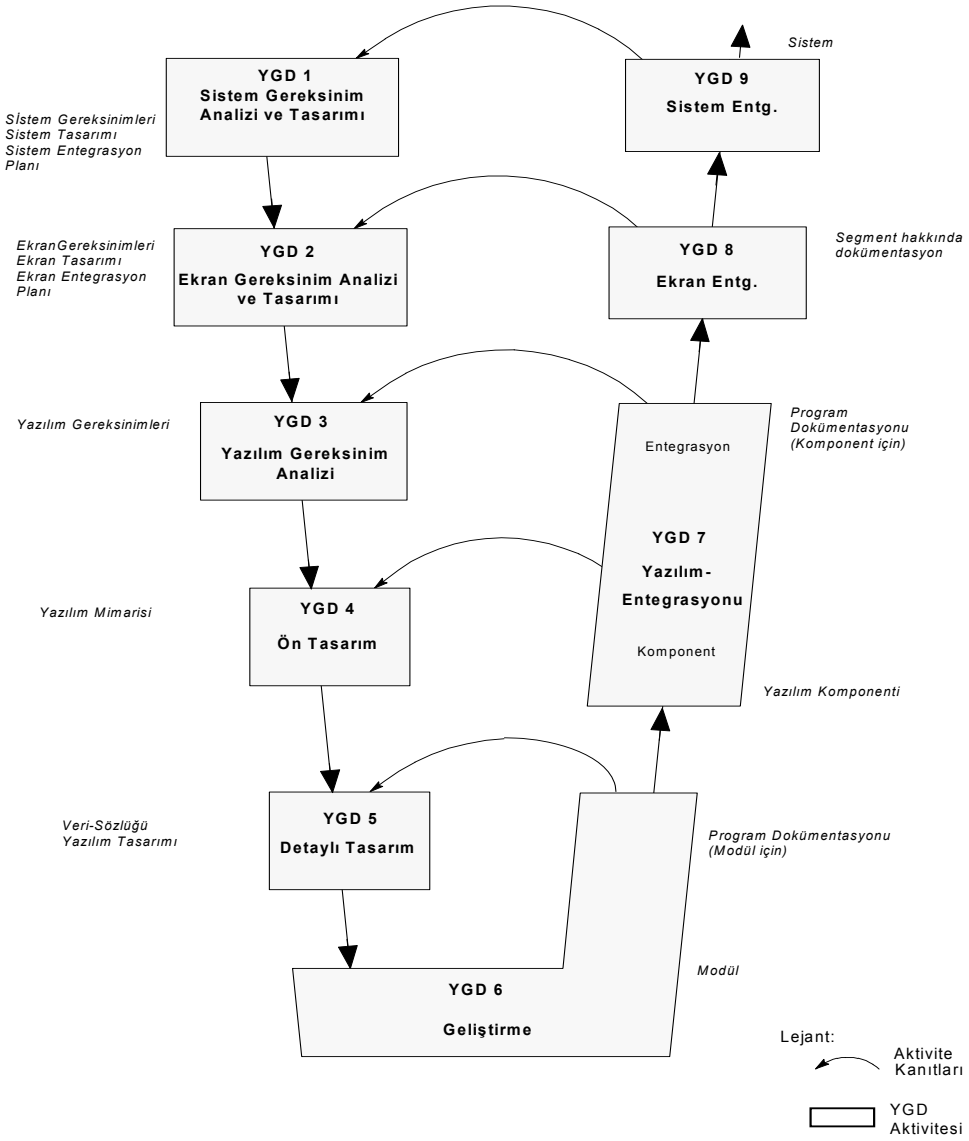
Burada:

1) YGD: Yazılım Geliştirme Departmanı
2) PY: Proje Yönetimi
3) KK: Kalite Kontrol birimi
4) KY: Konfigürasyon ve Güvenlik Yönetimi

Anlamına gelmektedir.

Not edilmesi gereken bir husus da yukarıda bahsi geçen rollerin projenin büyüklüğüne göre diğer bazı rollerle birleştirilmesinin mümkün olduğudur. Böyle bir durumda sistemin adı ‘Overloaded V-Model’ anlamına gelen küçük ve orta ölçekli yazılım uygulama projeleri için yüklü V-Modeli’dir [17].

3.1.4. Yazılım geliştirme sürecinde V-Model şeması



Şekil 3.2. V-Model şeması

3.1.5. V-Model kullanım alanları:

1) E-Devlet uygulamaları
2) Askeri ve Savunma Sanayi Yazılım Projeleri
3) Finansal Yazılımlar
4) Anahtar Teslimi Yazılım Uygulama Projeleri
5) Avrupa Birliği yazılım geliştirme standardına uygun projelerde [17].

3.2. Gelişigüzel model (Random model)

Bu süreç modelinde herhangi bir yöntem yoktur. Yazılım tamamen geliştiren kişiye bağlıdır. İleri zamanlarda geliştirilen yazılımı hazırlayan kişi bile anlamakta zorluk çekebilir. Bu yüzden izlenebilirliği ve bakımı çok zordur. Daha çok 1960'lı yıllarda kullanılan bir yöntemdir. Genellikle tek başına yazılım geliştirenler tarafından tercih edilmiştir. Geliştirilen yazılımın programlaması diğer metotlarla geliştirilen yazılımların programlamasına göre basittir.

Herhangi bir kuraldan bağımsız, yalnızca geliştiren kişiye bağımlı hatta onun bile bir zaman sonra anlayamadığı ve değiştirme zorluğu yaşadığı modeldir.60'lı yıllarda tek kişinin ürettiği programlar böyledir.

Basit öğrenci projeleri böyledir [15].

3.3. Barok modeli (Barok model)

Bu modelde yazılım yaşam döngüsünün temel adımları doğrusal bir biçimde takip edilir. Daha çok 1970'li yıllarda kullanılmış bir yöntemdir. Adımlar arası ilişkilerin tanımlı olmadığı bir yöntemdir. Günümüzde pek kullanılmayan bir yöntemdir. Bunun en büyük nedeni ise "Belgeleme" adımının bu modelde ayrı bir adım gibi ele alınıp yazılımın geliştirilmesi ve testinin ardından yapılmasıdır. Günümüzde kullanılan modellere aykırı bir durumdur. Günümüzde tercih edilen modellerde "Belgeleme" geliştirilen yazılımın ürünü olarak kabul edilmektedir. Ayrıca "Gerçekleştirme" adımını daha fazla ağırlık veren bir modeldir.

Barok Model'in adımları:

1. İnceleme
2. Analiz
3. Tasarım
4. Kodlama
5. Modül Testleri
6. Alt sistem Testleri

7. Sistem Testi
8. Belgeleme
9. Kurulum

Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği modeldir.
70'li yıllarda geliştirilmiştir. Belgelemeyi ayrı bir süreç olarak ele alır ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür. Hâlbuki günümüzde belgeleme yapılan işin doğal bir ürünü olarak görülmektedir. Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değildir.

Gerçekleştirim aşamasına daha fazla ağırlık veren bir model olup, günümüzde kullanımı önerilmemektedir [19].

3.4. Araştırma tabanlı model (Research-based model)

Bu model yap-at prototipi olarak da bilinir. Araştırma ortamları bütünüyle belirsizlik üzerine çalışan ortamlardır. Yapılacak işlerden edinilecek sonuçlar belirgin değildir. Geliştirilen yazılımlar genellikle sınırlı sayıda kullanılır ve kullanım bittikten sonra işe yaramaz hale gelir ve atılır. Model-zaman-fiyat kestirimi olmadığı için sabit fiyat sözleşmelerinde uygun değildir. Bu model için örnek yazılım projeleri olarak şu örnekleri verebiliriz: en hızlı çalışan asal sayı test programı, en büyük asal sayıyı bulma programı, satranç programı gibi [19].

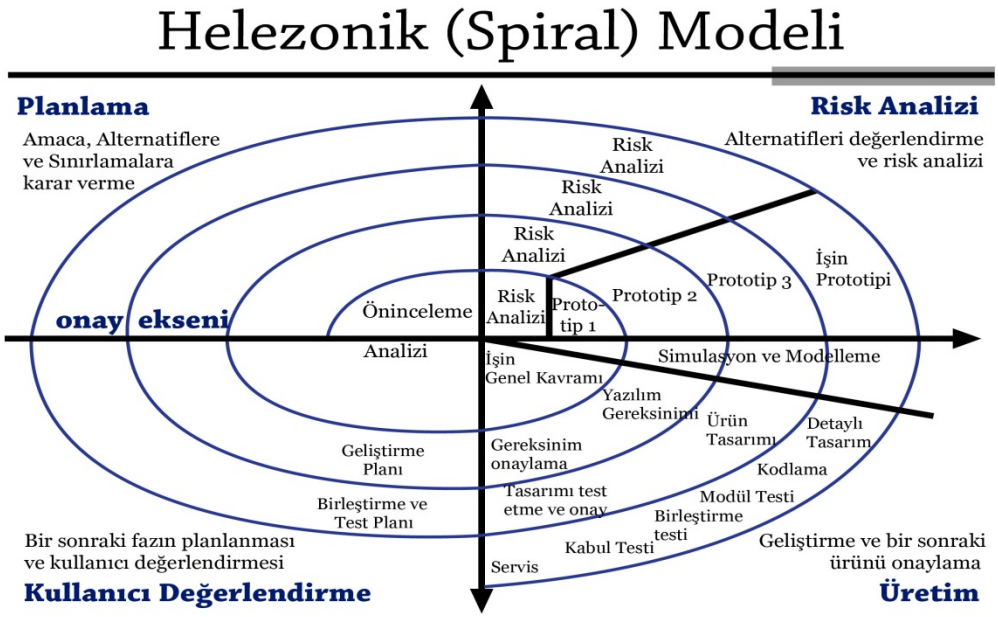
3.5. Helezonik (Sarmal) model (Spiral model)

Artırımsal modelin daha denetimli yapılan ve sistematik bir biçim kazandırılmış halidir. Asıl amaç her bir artırımın güvenli ve hızlı yapılmasını sağlamaktır. Sarmal model kullanıldığında her bir artırımda yazılımın yeni bir sürümü ortaya çıkar. İlk artırımda ortaya çıkarılan kesim adeta bir prototipmiş gibi düşünülebilir. Prototipten tek farkı uygulama ortamında kullanılanın gerçek yazılım olmasıdır. Bir mühendislik uygulamasının gerektirdiği özelliklerde (kapasite, verimlilik, kalite gözetilerek) geliştirilmiştir. Sonraki yinelemelerde üstüne yeni işlevler yine bu yaklaşımla mühendisçe eklenecektir.

Kalite, verimlilik ve kapasite açısından uygulamanın gereklerine uygun özellikleri taşıması her sarmal döngüde (her yeni sürümde) sağlanarak geliştirilme süreci sürdürülür.

Sarmalın her bir döngüsü belli etkinliklerden geçilerek tamamlanır:

- Müşteri ile yapılacaklara ilişkin iletişim; müşteri ile etkili iyi bir iletişimin kurulması yapılması gerekenlerin doğru belirlenmesi açısından zorunludur.
- Planlama; yapılacakların eldeki kaynaklar ve zaman çizelgesi gibi konular göz önünde tutularak planlanması.
- Risk çözümleme; teknik ve yönetsel risklerin çözümlenmesi.
- Mühendislik işinin icrası; gereken işlerin yapılması, yazılımın gerçekleştirimi, çözümleme, tasarım, kodlama.
- Kurma ve sürüm; kurma, sınama, işletme ve kullanıcıya eğitim verilmesi.
- Müşterinin değerlendirmesi; kullanıcı geribildiriminin alınması, yapılanın geçerliliğinin ve gerektiği gibi çalıştığının ortaya konması [19].



Şekil 3.3. Sarmal model yapısı

Sarmalın her bir döngüsü kendi içinde yapılması gereken yukarıdaki etkinliklere ayrılır. Etkinlikler projenin büyüklüğüne ve karmaşıklığına göre ayarlanabilir. Proje küçüldüğünde her bir etkinlikte yapılması gerekenlerin formalitesi azalabilir. Proje karmaşıklaştıkça da görevlerde yerine getirilmesi gerekenlerin biçimsel tanımı artırılarak dikkatli bir biçimde yerine getirilişleri izlenir. Ancak proje küçük ya da karmaşık olsun fark etmez; yazılım kalite güvencesi, risk yönetimi ve yazılım yapılandırma yönetimi gibi şemsiye etkinliklerin özenle yerine getirilmesi beklenir.

Sarmalın her bir dönüşünde evrimleştirici etki artar. İlk döngü bazen yalnızca geliştirilecek yazılımın özelliklerini belirlemeye yönelik olabilir. İkinci döngü gerçekleştirim, üçüncü döngü iyileştirme döngüsü olabilir. Hatta araya gerek duyuluyorsa prototip geliştirme döngüsü sokulabilir.

Sarmal model, yazılım bakım aşamasına da uygulanabilir. Yazılım geliştirme bittikten sonra bir sarmal döngü de bakım için geçilebilir. Böylece modelin tüm yazılım yaşam döngüsünü kapsaması sağlanabilir. Modele proje başlatma noktaları eklenebilir. Bu noktalar ilk eksen üstünde yer alır ve bir sonraki sarmal döngüyü başlatmak için karar noktaları olarak görülebilir [20]. Asıl olan projenin sürdürülmesi olmasına karşılık bu noktalar projeyi gerektiğinde durdurma / sonlandırma noktaları olarak da kullanılabilir.

Sarmal model çok karmaşık büyük projelerde uygulanabilecek gerçekçi bir model olarak görülmektedir. Çünkü süreçte ilerledikçe her bir evrimleşmede geliştiriciler ve müşteri birbirini daha iyi anlamaya başlar ve riskleri paylaşmayı öğrenir, riskleri birlikte karşılamaya alışırlar. Sarmal modelde, eğer yer verilmişse prototip geliştirme döngüsü daha çok risk azaltma mekanizması olarak düşünülür.

Bir yerde doğrusal sıralı modelin adım adım sistemli ve denetimli bir geliştirme yaklaşımı içinde Yazılım Mühendisliğinin kuralları ile kullanılmasını sağlar, böylece uygulamadaki gerçek durumların sisteme yansıması daha çok sağlanmış olur.

Bir başka önemli özelliği risklerin önceden öngörülmesi ile risklerin sorun yaratmasına yol açılmadan fark edilip önlenmesi için hazırlıklı olmaya zorlamasıdır.

3.5.1. Sarmal modelin dezavantajları:

Bu model yazılım geliştirmenin her derdine deva olarak görülemez. Sarmal süreç modelinin, yazılım geliştirilirken karşılaşılan her sorunu ve zorluğu azaltabildiği ancak tamamen ortadan kaldırabildiği söylenemez. Özellikle böyle bir modele dayanarak yazılım geliştirme projesine girmeye müşteriyi razı etmek çok zordur. Özellikle de sözleşme yapılarak geliştirilecek projeler için uygun değildir. Müşterinin izlenecek yaklaşımın en çok denetim sağlayıcı bir süreç modeli olduğuna ikna edilebilmesi çok zordur. Projenin başarısı büyük ölçüde risk yönetimindeki başarıya bağlı olduğundan risk yönetiminde iyice uzmanlaşma gerektirmektedir.

Henüz yeterince kabul gördüğü söylenemez [21].

3.5.2. Sarmal modelin avantajları:

1. Kullanıcı Katkısı

- 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.

2. Yönetici Bakışı

- Gerek proje sahibi, gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme ve hak ediş planlaması yapılır.

3. Yazılım Geliştirici (Mühendis) Bakışı

- Yazılımın kodlanması ve sınanması daha erken başlar.
- Risk Analizi Olgusu ön plana çıkmıştır.
- Her döngü bir fazı ifade eder.
- Doğrudan tanımlama, tasarım… Vs gibi bir faz yoktur.
- Yinelemeli artımsal bir yaklaşım vardır.
- Prototip yaklaşımı vardır [29].

3.6. Prototip model (Prototyping model)

Yazılım geliştirme projelerinde kullanıcıların genellikle gereksinimlerine ilişkin çok genel tanımları ancak yapabildikleri gözlenmiştir. Özellikle girdiler, işlemler ve çıktılar konusunda ayrıntılı tanımlar yapamazlar. Böyle olunca geliştiriciler uygulanacak algoritmaların işlevselliğinden, insan-makine etkileşiminin etkinliğinden, girdi ve çıktı biçimlerinin uygunluğundan emin olamazlar. Bu konularda belirsizliğin çok olduğu durumlarda prototip geliştirme modeli uygulanabilir ciddi bir seçenek oluşturmaktadır [19].

Prototip geliştirme modelinde, gereksinimler hızlıca toplanarak işe başlanılır. Geliştiriciler ve kullanıcılar aynı masa etrafında buluşarak yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına, hangi işlemlere uğrayacağına karar verirler. Daha sonra hızlıca yapılan bir tasarım ile yazılımın kullanıcıya yansıyacak yönünü aktaran bir prototip üretilir.

Prototip kullanıcının kullanımına ve değerlendirilmesine sunulur. Bu değerlendirmelere bakılarak prototip üzerinde gerekli değişiklikler yapılır. Prototipin yeni hali kullanıcı tarafından yeniden değerlendirilir. Böylece kullanıcının istediği yazılıma iyice yaklaşılmış bir prototip üzerinde yazılımın neler yapacağı konusunda kullanıcı ile anlaşmaya varılır.

Bu yaklaşımda gerçekte geliştirilen prototip, kullanıcı gereksinimlerinin iyi belirlenmesinin bir aracı olmaktadır. Eğer çalışan bir prototip BDYM araçları ile hızla oluşturulabiliyorsa gereksinimlerin belirlenmesinde önemli üstünlükler sağlanmaktadır. Gereksinimler prototip ile belirlendikten sonra bir başka uygun bulunan süreç modeli ile tercihan doğrusal sıralı modelle yazılımın gerçekleştirimine devam edilir.

Genelde prototipler çok yavaş, çok verimsiz ve çok hantal programlardır. Gerçek bir uygulamada kullanımları söz konusu olamaz. Yazılımın gerçekleştirimine geçildiğinde prototip atılır. Ancak müşterinin geliştirilen prototipi gerçek yazılım gibi algılama tehlikesi vardır. Böyle olunca kendince yapılıp bitirilmiş bir işin bir

daha üstelik zor ve zahmetli bir biçimde neden yeniden yapılmaya çalışıldığını anlamakta zorlanır.

Geliştirilen prototipte varılan son aşamayı yazılımın kendisi gibi algılayıp kullanmak isteyen müşteriye gerçek yazılıma kazandırılması gereken kapasite özelliklerinin ve verimlilik gibi kalite niteliklerinin nedeni kolay anlatılamaz.

Bu yüzden prototip geliştirmenin yalnızca gereksinimlerin doğru belirlenmesi için olduğunun başta kullanıcının iyice algılamış olması gerekir [29].

3.7. Hızlı uygulama geliştirme modeli
(The RAD - Rapid application development model)

Kullanıcının belli gereksinimlerinin sağlanması için çok kısa süreler içinde ortaya çıkartılan kapsamı daraltılmış alt sistemlere ilişkin yazılımlardır.

Bu model Doğrusal Sıralı Modelin çok hızlandırılmış bir uygulaması gibi görülebilir. Gerçekleştirimin hızlı yapılabilmesi genelde bileşen tabanlı olmasını gerektirir. Hazır bileşenler gereksinimleri karşılayabiliyorsa 60-90 gün içinde çalışan bir program ortaya çıkartılabilir [19].

Bu modelin aşamaları şunlardır:

İş Modelleme:

- Kısıtlı bir işin gerektirdiği bilgi akışının ne olduğu hızlıca ortaya çıkartılır.
- Veri akışı içinde ortaya çıkan bilgilerin neler olduğu saptanır ve hangi işlemler sonucu bu bilgilerin elde edildiği belirlenir.
- Yapılması gereken işlemlerin kimin tarafından nasıl yapıldığı belirlenir.
- Elde edilen bilgileri kimin kullanacağı ve ulaştırılacağı yere karar verilir.

Veri Modelleme: İş modelinin içinde akan bilgilerin veri nesneleri olarak nasıl tanımlanacağına, aralarındaki ilişkilerinin nasıl kurulacağına ve nasıl korunacağına karar verilir. Yazılım veri tabanındaki verilere nasıl erişileceğine, verilerin nasıl

işleneceğine, verilerin nasıl korunacağına ve verilerin gereksinim duyulan yerlere hangi biçimde ulaştırılacağına karar verilir.

İşlem Modelleme: İş modelinde verinin bilgiye dönüşümünü sağlayacak dönüşümlerin nasıl sağlanacağı belirlenir.

Uygulama Geliştirme: Bu model 4. kuşak dillerin kullanılmasını öngörür. 4. kuşak diller programlamayı, pek çok ayrıntısını kodlayıcının üstünden alarak çeşitli çizelgelerde seçeneklerin işaretlenmesine dönüştürmüştür. Dolayısıyla isteneni yapan programlar hızla oluşturulabilir.

Bu dillerin yapılan uygulamanın kendisine ilişkin yeniden kullanılabilir program birimlerini (veri tabanına ilişkin tanımları, işlemlere ilişkin yordamları, girdi /çıktı görüntüleri vb. gibi) oluşturma ve bunları kullandırma yetenekleri bulunmaktadır.

Sınama ve Başa Dönme: Yeniden kullanılabilirliğin sağlanması ölçüsünde sınama gereksinimi azalır. Çünkü yeniden kullanılan birimler daha önce sınanmış olacağından genelde bir daha sınanmalarına gerek kalmaz. Bu yüzden sınama zamanı da kısalır. Yalnızca yeni geliştirilmiş birimler ve ara yüzler sınanır.

Bu modelin kullanılabilmesi için gereksinimlerin dar kapsamlı iş modellerine ayrılabilmesi gerekir. Bunun sağlanması durumunda hızlı uygulama geliştirme modeli kullanılarak her iş modeli 2-3 aylık zaman dilimleri içinde gerçekleştirilerek istenen yazılım tamamlanabilir. Eğer daha hızlanmak istenirse birbiri ile eş zamanlı çalışarak farklı iş modellerini gerçekleştirecek çalışma takımları oluşturulur.

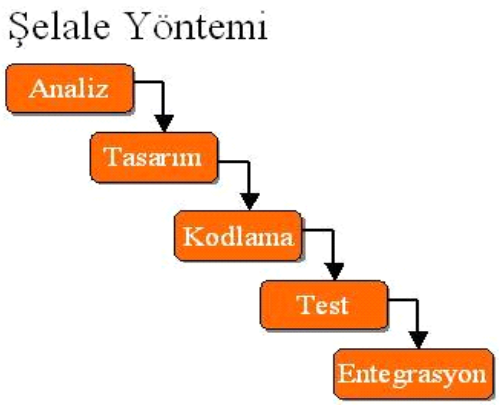
Sakıncaları:

- Kapsamlı büyük projelerde çok sayıda çalışma takımına gereksinim olacağı için pratik olmayabilir.
- Hem geliştiricilerin hem de müşterinin hızlı çalışmaya bunun gereklerini yerine getirmeye kararlı olması gerekir. Her iki tarafın buna hazır olmaması durumunda başarı şansı yoktur.
- Her tür uygulama için uygun değildir. Kendi doğası nedeniyle küçük modüllere bölünebilir olmayan uygulamalarda sorunlarla karşılaşılır.

- Performansın önemli olduğu uygulamalar için uygun değildir.
- Teknik risklerin çok olduğu uygulamalar için uygun değildir. Eğer uygulama yeni bir teknolojiyi yoğun bir biçimde kullanacaksa ya da yeni geliştirilecek yazılım mevcut yazılımlar ile birlikte yoğun bir ilişki içinde ilişkilendirilerek işletilecekse riskler artar [29].

3.8. Çağlayan (Şelale) modeli (Waterfall model)

Şelale yönteminde yazılım geliştirme süreci analiz, tasarım, kodlama, test, sürüm ve bakım gibi safhalardan oluşur. Geleneksel yazılım metotlarında bu safhalar şelale modelinde olduğu gibi lineer olarak işler. Her safha, başlangıç noktasında bir önceki safhanın ürettiklerini bulur. Kendi bünyesindeki değişikler doğrultusunda teslim aldıklarını bir sonraki safhanın kullanabileceği şekilde değiştirir [30].



Şekil 3.4. Şelale yöntemi

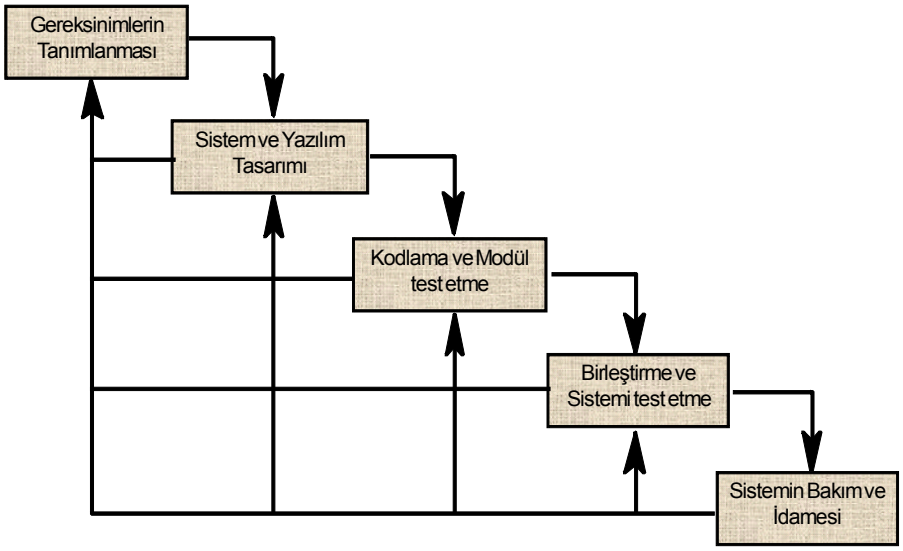
3.8.1. Şelale modelinin özellikleri

- Şelalenin her basamağında yer alan aktiviteler eksiksiz olarak yerine getirilir. Bu bir sonraki basamağa geçmenin şartıdır.
- Her safhanın sonunda bir doküman oluşturulur. Bu yüzden şelale modeli doküman güdümlüdür.
- Yazılım süreci lineerdir, yani bir sonraki safhaya geçebilmek için bir önceki safhada yer alan aktivitelerin tamamlanmış olması gerekir.

- Kullanıcı katılımı başlangıç safhasında mümkündür. Kullanıcı gereksinimleri bu safhada tespit edilir ve detaylandırılır. Daha sonra gelen tasarım ve kodlama safhalarında müşteri ve kullanıcılar ile diyaloga girilmez [31].

3.8.2. Şelale modelinin dezavantajları

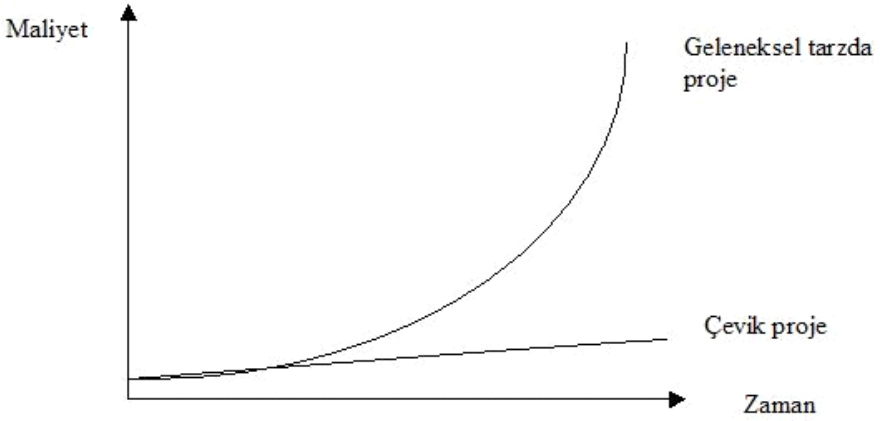
- Safhaların birbirinden kesin olarak ayrı tutulmaları gerçekçi değildir. Projelerde safhalar arasındaki bu sınırlar yok olabilir.
- Teoride safhalar birbirlerini takip edeler. Projelerde bunun bazen mümkün olmadığını ve önceki safhalara geri dönülmek zorunda kalındığını görebiliriz.
- Safhalar arası geri dönüş yetersizdir. Model değişikliğe açık değildir.
- Müşteri gereksinimlerinin proje öncesi detaylı olarak kâğıt üzerinde oluşturulması ilerde sorun yaratabilir. Müşteri gereksinimleri değişikliğe uğrayabileceği için, yazılım sisteminin de yapısal değişikliğe uğraması kaçınılmaz olabilir. Böyle bir durum maliyeti artırır, çünkü yeni ve değişen gereksinimleri implemente edebilmek için modelde yer alan safhaların birkaç kere uygulanması gerekebilir.
- Sistemin kullanılır hale gelmesi uzun zaman alabilir.
- Başlangıçta yapılan hataların tespiti çok uzun zaman alabilir. Bu hataların giderilmesi maliyeti yükseltir.
- Modül implementasyonları için zaman tahminleri proje planlarını oluşturan yöneticiler tarafından yapılır. Teknik bilgiye sahip olmayan şahıslar tarafından yapılan bu tahminler çoğu zaman doğru değildir. Bu durum proje planlama sürecini negatif etkiler [31].



Şekil 3.5. Şelale model işleyişi

Proje başlangıcında her detayı göz önünde bulundurmak mümkün olmadığı için, şelale modeliyle geliştirilen yazılım sistemlerinin müşteri gereksinimlerini tam tatmin etmediğini görmekteyiz. Bunun önüne geçebilmek için projenin başlangıç safhasında analiz için çok zaman harcanır ve müşteri gereksinimleri en ince detayına kadar tespit edilir. Aslında proje başlangıcıyla oluşturulan dokümanlar obsolet (eskimiş) hale gelmiştir, çünkü müşteri gereksinimleri piyasa ve rekabet koşulları gereği değişikliğe uğramış olabilir. Ne yazık ki şelale modeli bunları dikkate almaz ve müşterinin talep ettiği değişiklikleri aza indirmeye çalışır. Bunun bir sebebi de sonradan gelen değişiklik taleplerinin maliyeti yükseltmesidir, çünkü bu durumda şelale modelinde yer alan safhaların birkaç kere uygulanması gerekebilir [31].

Bu model kullanıp, ilk sürüm teslimat safhasına geldiğinizde yeni bir yazılım gereğine cevap vermeniz pek mümkün olamayabilir çünkü bütçenin %90'ına yakınını sarf etmiş olabilirsiniz.



Şekil 3.6. Şelale yöntemi grafiği

Bu çerçeveden bakıldığında proje sonunda oluşan program müşterinin aktüel gereksinimlerini tatmin etmez durumdadır. Program daha çok müşterinin proje başlangıcında sahip olduğu gereksinimleri tatmin edecek şekilde tasarlanmıştır. Projelerin birkaç sene boyunca sürebileceğini düşünürsek, aslında bu süreç sonunda oluşan program aktüel değildir [30].

3.8.3. Modelin uygulanmasında karşılaşılan sorunlar

- Müşteri ne istediğini tam olarak bilmeyebilir. Bu yüzden proje öncesi detaylı analiz yapılması, müşterinin her gereksinimi dile getirdiğinin garantisi değildir. Müşterinin bazı gereksinimlerini projenin ilerleyen safhalarında keşfetmesi durumunda, oluşan değişikliklerin projeye dâhil edilmesi hemen hemen imkânsız olacaktır. Bunun en büyük sebeplerinden birisi de yazılım için oluşturulan tasarımın projesi öncesi belirlenmesi ve bu yüzden ilerde meydana gelebilecek değişikliklerin göz önünde bulundurulmamış olmasıdır. Projenin ilerleyen safhalarında meydana gelen her değişiklik tasarımı zorlar. Tasarım çevik olmadığı için değişiklikleri taşıyamaz.
- Müşteri ne istediğini doğru olarak ifade edemeyebilir. Bu durumda proje öncesi yapılan analizler doğru olmayacaktır. Bu müşterinin istemediği bir yazılım sisteminin meydana gelmesine sebep olur. Şelale yöntemi müşteri ile devamlı diyalog içinde olunmasını engeller. Müşteriden geri dönüş sağlanamadığı için, müşterinin analiz safhasında meydana gelen yanlış anlaşılmaları düzeltmesi mümkün değildir.

- Şelale yönteminde proje akışı bir sonraki safhaya geçiş yönündedir. Bu yüzden iletişim tek yönlüdür. Safhalar arası geri dönüş mümkün değildir. Bu yapılan hataların tamir edilmesini engeller.
- Şelale yöntemi ile müşterinin istediği yazılım sistemi proje sonunda tamamlanır. Ancak bu safhada müşteri yazılım sistemini test edebilir.

3.8.4. Şelale modelinin avantajları

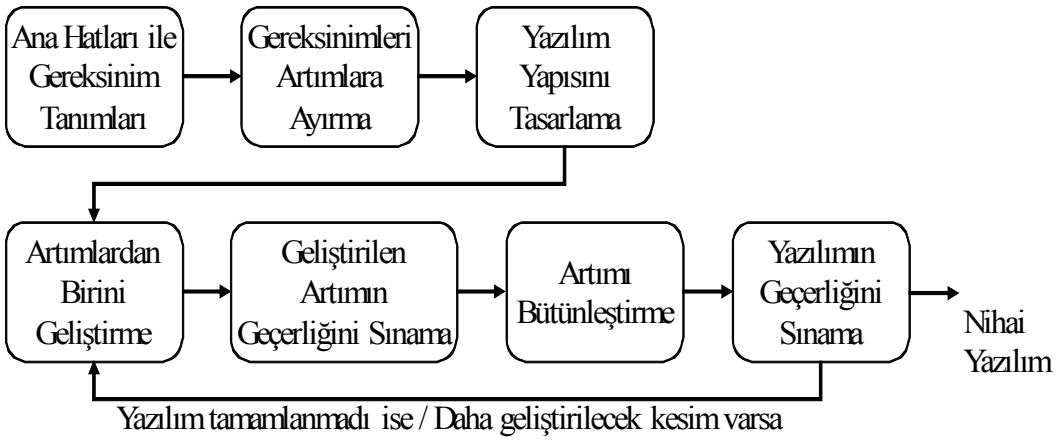
Yukarıdaki sorunlar yazılım geliştirme açısından iyice ciddiye alınması gereken durumlardır. Ancak doğrusal sıralı model, yazılım geliştirmenin nasıl yapılacağı konusunda tanımlı bir dayanak oluşturduğundan, bu modelle üstesinden gelinebileceği kestirilen projeler için uygulanabilir niteliktedir.

Ne istendiğinin iyi bilindiği, gereksinimlerin proje başladığında eksiksiz ve tutarlı tanımlanabileceğine inanılan projeler için uygundur [31].

Sadece gereklerin tam ve kesin olarak belirlendiği projelerde (ki böyle müşteri isteklerinin ve gereklerin başlangıçta sabitlenebildiği projeler pek görülmez) bu model kullanılabilmektedir [32].

3.9. Artırımsal model (Incremental model)

Doğrusal sıralı modele yinelemeli bir özellik katılarak artırımsal yazılım geliştirme modeli oluşturulmuştur. Artırımsal model bir takvime bağlı olarak yazılımı kesim kesim geliştirip teslim etmeye dayanır. Her bir yeni kesim öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür.



Şekil 3.7. Artırımsal model uygulama şeması

Anlaşılabilir olması için bir metin düzenleyici yazılımın geliştirileceğini varsayalım (uygulama alanı olarak pek gerçekçi olmamasına rağmen anlatımsal olarak amaca uygun bir örnek);

- Çekirdek Yazılım-1 (ilk artırım) temel kütük yönetimi ile metin düzenlemenin ve belge oluşturmanın temel işlevlerini içerebilir,
- Genişletilmiş Yazılım-2 (ikinci artırım) sıradan kullanıcılar açısından pek de gereksinim duyulmayabilecek belge oluşturmanın karmaşık işlevlerini içerebilir,
- Genişletilmiş Yazılım-3 (üçüncü artırım) heceleme ve dilbilgisi kuralları için sınama yapma olanaklarını yazılıma katmayı amaçlayabilir,
- Genişletilmiş Yazılım-4 (dördüncü artırım) sayfa üstüne basmada matbaa olanakları denebilecek özellikleri yazılıma kazandırmayı amaçlayabilir [20].

Artırımsal model kullanıldığında ilk artırım yazılımın çekirdek kesimine karşılık gelir. Yazılımın çekirdek kesiminin temel gereksinimleri eksiksiz karşılaması gerekir. Bu sürüm her açıdan kendini kullanıcısına kabul ettirebilmelidir. Böylece yazılımın tümü oluşmadan en temel kesimleri en geniş biçimde gerçek uygulamalarda yaygın olarak kullanılarak gözden geçirilmiş olur. Çekirdek kesimin hemen devreye girmesi geliştirimi yapılacak sonraki artırımsal kesimler için de iyi bir geribildirim sağlar. Yazılımın geliştirilmesi tamamlanmadan önce kullanılacağından çekirdek kesimin gereksinimlere daha uygun biçime getirilme şansı artar. Sonraki artırımlar için de aynı durum söz konusudur.

Artırımsal model yazılım geliştirmenin kısıtlı sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğü vardır. Ayrıca çalışanlar da her artırım geçildiğinde uygulama alanına ilişkin daha çok deneyim kazanmış olurlar [29].

3.9.1. Avantajları

Artımsal modelin en büyük avantajı, müşterinin süreç içerisinde daha fazla yer almasının sağlanabilmesidir. Kullanıcı, sistemin gereklerinin tek tek yerine getirildiğini gözlemleyebilmekte, varsa değişiklik önerisini hemen verebilmektedir.

Örneğin ŞELALE modelinde, müşteri son safhaya kadar sistemi görememekte, sürece katılımı minimum düzeyde kalmakta idi. Doğru yazılım ürünlerinin geliştirilebilmesi için müşterinin yeterli düzeyde süreç içerinde yerini alması ve sürece katılması gerekmektedir.

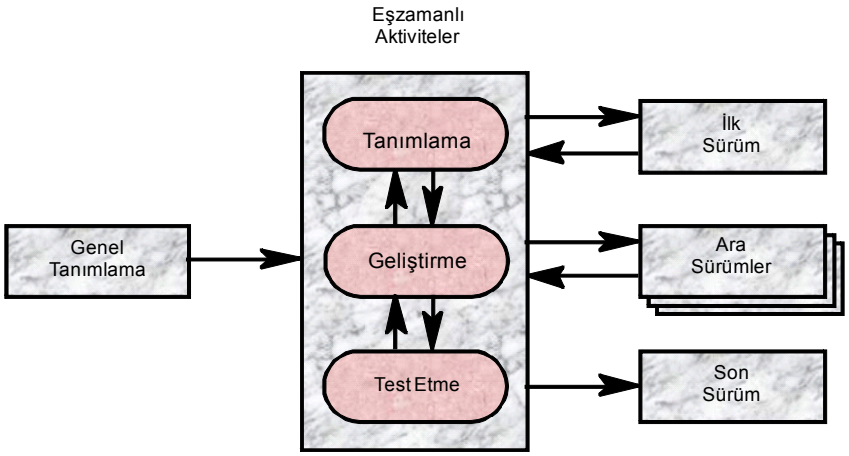
Bu kural PROTOTİP modelinde bir ölçüde yerine getirilebilmekte, bununla beraber evrimsel süreçlerde tam anlamı ile gerçekleştirilebilmektedir. Müşterinin süreç içerisinde yerini alması projenin başarısızlık riskini azaltan bir olaydır. Bu modelin bir diğer avantajı ise, yüksek öncelikli gereklerin erken geliştirilerek, daha fazla test edilebilmesine olanak sağlanmasıdır [29].

- Yazılım tam tamamlanmadan önce kullanılmaya başlanır.
- Kullanıcılar ilk yazılım kesimlerinden kazanacağı deneyimle daha sonraki kesimlerde ne isteyeceğini daha iyi bilmeye başlar.
- Tüm projenin başarısızlığa uğrama riskini en aza indirir.
- Kullanıcının yazılım işlevlerinden en çok önem verdiklerini içeren kesimlerin geliştirilmesine öncelik verilir [20].

3.10. Evrimsel model (Evolutionary model)

İlk tam ölçekli modeldir.
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.
Modelin başarısı ilk evrimin başarısına bağımlıdır [20].



Şekil 3.8. Evrimsel geliştirme süreç modeli

Yazılımlar bütün karmaşık sistemler gibi zaman içinde evrimleşir. İşe ve ürüne ilişkin gereksinimler yazılımlar daha geliştirilirken bile değişikliğe uğrar, bu yüzden işin başından sonuna doğrusal bir model öngörerek geliştirme yapmak hiç de gerçekçi gözükmemektedir.

Karmaşık büyük bir yazılımı her şeyiyle bir defada eksiksiz geliştirilmesini beklemek günümüz pazar koşulları açısından da pek anlamlı gözükmemektedir. Rekabet ve iş koşulları kısıtlı bir sürümün temel sayılabilecek bir çekirdek çerçevesinde önce oluşturulmasını, sonra bunun üstüne eklenen özellikler ile yeni sürümlerinin çıkarılmasını zorunlu kılmaktadır.

Yazılım mühendisleri pazarın da zorlaması ile bir yazılımın önce işe yarayacak bir ilk sürüm olarak elde edilmesini sağlayacak sonra da zaman içinde evrimleştirecek bir yazılım geliştirme modeli aramışlardır. Evrimleştirici modellerin süreç aşamaları yinelenebilen bir özellik taşır. Sonraki sürümlerin de ilk sürüm gibi benzer biçimde elde edilebilmeleri öngörülür [29].

Örnek:

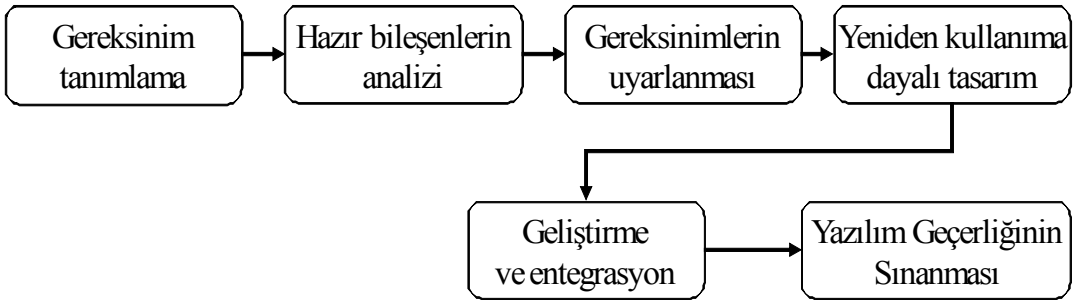
- Çok birimli banka uygulamalarıdır.
- Önce sistem geliştirilir ve Şube-1'e yüklenir.
- Daha sonra aksaklıklar giderilerek geliştirilen sistem Şube-2'ye yüklenir.
- Daha sonra geliştirilen sistem Şube-3'e yüklenir.
- Belirli aralıklarla eski şubelerdeki güncellemeler yapılır.

Aksaklığı:

- Değişiklik denetimi
- Konfigürasyon Yönetimidir
  - Sürüm Yönetimi
  - Değişiklik Yönetimi
  - Kalite Yönetimi

3.11. Yeniden kullanılabilir model (Reusable model)

Organizasyon tarafından daha önce hazırlanmış veya dışarıdan temin edilmiş yazılımların (veya yazılım parçalarının) kullanılması ile geliştirme yapılması son yıllarda popülaritesi artan bir yaklaşımdır. Organizasyonların olgunlukları arttıkça, bu tür uygulamalar yapabilmek için alt yapı kurulmuş olmaktadır. Şekil 10'da bu modelin işleyişi kısaca görülmektedir [21].



Şekil 3.9. Yeniden kullanılabilir süreç modeli

3.11.1. Avantajlar:

Maliyetler azalıyor ve risk düşüyor.

Yazılımlar daha hızlı gerçekleştiriliyor [20].

3.11.2. Problemler

Gereksinimlerin uyarlanması gerektiğinden kullanıcının gerçek gereksinimlerinin karşılanmasından uzaklaşılabiliyor.

Yazılımların geliştirilmesinde hazır kullanılan yazılım bileşenlerinin sahipliği dışarıda olduğundan bakım güçlükleri çekilebiliyor [20].

Bu yaklaşımın popülaritesinin sürekli artmasına karşın, yazılım endüstrisinde bu yönde hizmet veren firmalar kısıtlı biçimde yer almaktadır. Örneğin, gerçek zamanlı ve/veya gömülü yazılım geliştiren firmalar için bu yönde kısıtlı (belirli donanımlara özgü) hizmet veren organizasyonlar bulunmaktadır.

Bu sürecin aşamaları:

- Bileşen analizi,
- Gereklerin modifikasyonu,
- Tekrar Kullanım ile sistem tasarımı,
- Geliştirme ve entegrasyondur [32].

3.12. Agile (Çevik) modelleme

Çevik modelleme (agile methods, yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve dokümantasyonunu yapmaya yönelik pratiğe dayalı yöntemlere verilen genel addır.

Çevik modellemenin başlıca özelliği veri modelleri ve ara yüzü modelleri gibi modelleme tekniklerinin neler olduğunu ve bunların ayrıntılarını söylemek yerine bu tekniklerin nasıl uygulanması gerektiğini söylemesidir. Mesela; yapılan projelerin test edilmesi gerektiğini belirtse bile nasıl test hazırlanacağına değinmemesi gibi. O sadece bir yöntemler biçimidir ve bir projenin etkili, hızlı bir şekilde ortaya çıkarılmasında, müşterinin ihtiyaçlarını karşılamasında ve aynı zamanda da her türlü değişikliğe kolayca adapte olabilmesinde geliştiricilere yol gösterir.

Ayrıca Hızlı bir geliştirme sürecine sahip, kaliteli ürünlerin ortaya çıkması için uygulanan bir geliştirme yöntemidir. Agile geliştirme yöntemlerinde en önemli sonuç kriteri müşteri memnuniyetidir. Asıl amaç müşteriyi memnun eden bir ürün ortaya koyabilmektir. Ortaya çıkan ürünün çalışan ve müşterinin ihtiyaçlarını tam olarak karşılayabilen bir ürün olması en temel amaçlardan biridir [27].

3.12.1. Hangi durumlarda kullanılabilir?

Bu metotların kullanılmasının en uygun olduğu durumlar şunlardır:
- Projenin yazılım evresinde müşteriden gelebilecek talep değişikliklerinin tahmin edilemez olması
- Projenin parçalarının önce tasarlanıp ardından hemen geliştirilmesinin gerekmesi ve önceden ne yapılacağını, detaylı yol haritasını ve tasarımını tahmin etmenin çok güç olması
- Analiz, tasarım ve test etme süreçlerinin ne kadar zaman alacağının önceden bilinememesi
- Yazılım ekibinin birlikte çalışmak, hiyerarşiye önem vermemek, sağlam iletişim kurmak gibi özelliklere sahip olması

Elbette bazı durumlarda çevik programlamadan vazgeçilmesi gerekebilir. Örneğin; çok kişinin dâhil olduğu projelerde çevik metotlar ile proje geliştirmek mümkün olmayabilir ya da aynı yerde bulunmayan takım arkadaşları ve hiyerarşik yapının her an hâkim olduğu şirket ortamlarında klasik yöntemler daha uygun olabilir. Zaten, düşünülenin aksine, bu metotları kullanarak programlama ve modelleme yapılmasını öneren insanlar bu gibi durumlarda klasik yöntemlerin kullanılmasına karşı değildir.

3.12.2. Çevik programlama metotlarının özellikleri:

Çevik metotlar tam bir yazılım süreci değildir ama kapsamlı yazılım geliştirme yöntemlerini tamamlayıcı niteliktedir. Başlıca çevik süreç modelleri:
- Sınırsal programlama(Extreme Programming-XP)
- Çevik Birleştirilmiş Süreç (Agile Unified Process)

- Scrum
- Test Güdümlü Geliştirme (Test-driven Development)
- Çevik bilgi Metodu (Agile Data Method)
- Özellik güdümlü geliştirme (Feature-Driven Programming)

Çevik yazılım geliştirme, takım çalışmasını desteleyen liderlik vasıflarını, sorumluluk alabilme ve kendi kendini organize edebilme özelliklerini ve nitelikli ve yeterince hızlı yazılım geliştirmeye yarayan mühendislik uygulamalarını destekleyen proje yönetim sürecini kapsar. Bu yönetim süreci hem müşteri taleplerini hem de şirket amaçlarını düzenleyerek olumlu bir geliştirme ortamı ve yaklaşım sunar.
Çevik modelleme işleri kısa vadeli planlar ve küçük gelişmeler halinde yapmayı uygun görür. Kısa vadeli planlar yineleme (iteration) olarak bilinir. Her yineleme sürecinde belli bir takım yazılım ya da modelle üzerinde çalışır ve bu süreç planlama, talep analizi, tasarım, kodlama, birim testi ve kabul testi gibi süreçleri kapsar. Yineleme süreci değişikliklere uyum sağlamayı kolaylaştırırken genel riski de azaltır. Tek bir yineleme bir ürünü piyasaya sürmek için ona yeterli işlevsellik katmayabilir ancak amaç her yineleme sonunda en az sorunla çalışan mevcut bir sürüm elde edebilmektir. Bir ürünü sürüm olarak piyasaya sürmek ve yeni özellikler eklemek için birden çok yineleme gerekir.

Çevik programlama ve modellemede dokümantasyon da yapılır. Ancak dokümantasyon ürünün kullanılabilirliği ve verimliliğinin önüne geçmez. Aksine dokümantasyon sadece gerekli görülen yerlerde yapılır. IBM ’de çalışan ve aynı zamanda da “Effective Practices for Modeling and Documentation” kitabının yazarı olan scott ambler’e göre; geliştiricilerin tahtada yazıp çizdikleri modeller uzun uzadıya yazılan dokümanlardan daha kullanışlıdır ve insanların aylarca dokümantasyon yapmasının önlenmesi gerekmektedir [26].

Çevik projedeki takım yapısı çoğunlukla iş değişimine dayanan ve kendi kendini organize eden bir yapıdır. Firma içindeki hiyerarşik durumlar takım içinde göz ardı edilir ve çalışmalar bu şekilde yürütülür.
Birçok projeden insanlar daha ilerideki işler için önceden çalışmaya başlarlar. Örneğin; gelecekte yapılacak işler için karmaşık altyapılar geliştirmek ve modelleme

yapmak. Bu durum çok büyük bir zaman kaybına yok açar ki, taleplerde bir değişiklik olması durumunda bu çalışan insanlara fazladan emek ve iş gücü olarak geri döner. Çevik modeller ihtiyaç olmadan herhangi bir şeyin yapılmasına ve boşa zaman kaybedilmesine karşıdır.

Proje aşamasında test süreci, önemli bir yer kaplar. Tüm testlerin son ana bırakılması yanlış anlaşılmaların projeyi çok farklı yerlere götürmesine sebep olabilir. Henüz yapım aşamasındayken, yapılan kısmı farklı kişilerin test edip geri bildirimde bulunması, bu açıdan hayati bir süreçtir. Bu sayede yanlış henüz yeni yapılmışken tespit edilip ortadan kaldırılır. Çevik programlama ve modelleme geliştiricilerin her an yan yana bulunarak bu iletişimde olmasını ve aynı zamanda da müşteri ile iletişim içinde bulunmasını öngörerek bu sorunu aşar. Kabul testleri ise müşteri tarafından en son aşamada yapılacak testlerdir. Burada geri dönülen hatalar yeniden modellemeye aktarılabilir. Çevik modellemede en son yapılan değişiklikler bile olumlu karşılanır.

Agile Yazılım Geliştirmede, takım çalışması, güven, iletişim gibi insanlar ile olan etkileşim çok önemli. Sürekli bilgi alışverişi ve beyin fırtınalarına dayanan bir geliştirme süreci olduğundan bu kavramların bilincinde olmak geliştirme sürecine artılar katabiliyor. Geliştirme sırasında ya da öncesindeki analiz kısımlarında sürekli yüz yüze bir iletişim ile gereksinim ve ihtiyaçların sürekli üzerinden geçilmesi daha sağlıklı sonuca ulaşmak için kaçınılmaz ihtiyaçtır.

Agile yazılım geliştirme süreçlerinde her zaman çalışan bir yazılım aranan bir kavram. Zaten soyut kavramlardan oluşan yazılım dünyası, bu dünyaya uzak olan müşteri için pek bir şey ifade etmeyecektir. Bu yüzden bazı şeyleri biraz daha somutlaştırmak adına çalışan bir yazılım her zaman aranan bir kavramdır. Uygulamaların nasıl çalışacağına ya da ne yapacağına dair uzun ve karışık dokümanlardan çok müşteriye çalışan bir uygulama göstermek her zaman daha faydalı olacaktır.

Agile'ın altını çizdiği bir başka husus da müşteri ilişkileri. Belli bir sözleşme yerine sürekli müşteri ile iş birlik halinde olunmasının daha faydalı olacağı manifestoda söylenen diğer bir şey. Müşterinin ihtiyaçlarını sürekli sorgulamak ve işi onunla beraber yapmak ortaya çıkacak ürünün kalitesi adına önemli bir nokta. Bu noktada,

ihtiyaçların sürekli sorgulanmasından dolayı da ortaya çıkacak değişiklik taleplerine de açık olmak gerekmekte. Bu değişikliklerin yönetilmesi sayesinde ürünlerin canlılığı ve tam olarak ihtiyaca uygun olması sağlanabiliniyor [26].

BÖLÜM 4. GÜVENLİ YAZILIM GELİŞTİRME SÜREÇ MODELİNİN SEÇİMİ

Güvenli yazılım süreç modelinin seçilebilmesi için altı fonksiyonel nokta değerli ve otuz beş kriterli bir set oluşturduk. En çok kullanılan altı yazılım geliştirme süreç modelini de bu kriter setine göre inceledik. Aynı zaman da bu kriter setine göre de istenilen güvenli modeli belirleyip bu altı yazılım geliştirme süreç modelini istenilen güvenli modele göre karşılaştırdık.

4.1. Fonksiyon Değerlerinin Atanması

Bu çalışmamızda yazılım süreç geliştirme modellerinden altı tanesini karşılaştıracağız. Bunlar:

1. Şelale Modeli
2. Artırımsal Model
3. Yeniden Kullanılabilir Model
4. Evrimsel Prototip Model
5. Helezonik (Sarmal) Model
6. Hızlı Prototip Modeli

4.2. Kriter Setinin Belirlenmesi

Kullanılabilirlik, teknolojik ilerlemeler, insan kaynakları, verimlilik, finansal kaynaklar ve yönetilebilirlik Tablo 4.1'de gösterildiği gibi, bir süreç seçim kriteri belirlenmesinde ek faktörler olarak önerilmektedir. Uygun modeli seçmek için belirtilen her kriter kendi içinde beş kategoriye bölünmüştür. Bunlar: ürün, personel organizasyon ve kaynaktır.

Problem tanımlamadan başlayarak sistem güvenliğinin ölçülmesi kriterine "problemin güvenlik tanımı" denilmektedir. Bu kriter problem kategorisinde kabul edilmektedir ve problemlerin tanımlanmasındaki zayıflıklarla ilgilenmektedir.
Personel kategori kısmında, uygun maliyetli güvenlik planlarının geliştirilmesi ve uygulanabilmesi için gerekli olan bilinçlendirme ve becerilerin geliştirilmesi ölçülmektedir.

Kullanıcı katılım ve güvenlik kriterinde sistem kullanıcılarını rolü saygıyla karşılanmaktadır ve bu kriterde yazılım geliştirme sürecinde kullanıcının duyarlılık seviyesi ölçülmektedir. Kullanıcı tarafından yapılan güvenlik uygulamaları yazılım geliştirme takımı için faydalı olmaktadır.

Kaynak kategorisi, kaynakların geliştirilmesi için üç farklı teknoloji ile ilgilenmektedir. Bunlar; yazılım geliştirme aşamasında teknolojinin kullanılabilirliğini ölçmek, yazılım geliştirme takımının işlerini etkili bir biçimde başarabilmeleri için zamanında doğru bilgiyi almak ve alınan operasyonel unsurlar ile bilgisayar güvenliği arasındaki etkileşimin seviyesini ölçmektir.

Kullanım profilinde ortaya çıkan sonucun ölçümü ürün kriteri bölümünde anlatılmıştır. Yazılım geliştirme işlemlerinde başlangıçtan sona kadar oluşan güvenlik endişeleri ise güven yönetimi, risk ve kontrol ölçümlerine dâhil edilmiştir [10].

Model çerçevesinde altı fonksiyonel nokta değerli ve otuz beş kriterli bir set oluşturulmuştur.

	Kriter (Ci)	F1	F2	F3	F4	F5	F6
1	Uygulamanın Olgunluğu	Güçlü	Yeni	Benzer	Standart	İyi-Anlaşılır	Ana-Kapalı
2	Problem Karmaşıklığı	Önemsiz	Basit	Titiz	Zor	Karmaşık	Zorlu
3	Kısmi İşlevsellik Şartı	Arzu Edilmiyor	İsteğe Bağlı	Arzu Edilir	Kritik	Kaçınılmaz	Zorlu
4	Değişim Frekansı	Seyrek	Az	Orta	Hızlı	Daha Hızlı	Gösterişli
5	Değişim Büyüklüğü	Önemsiz	Seyrek	Az	Orta	Büyük	Aşırı
6	Tanımlı Güvenlik Sorunu	Önemsiz	Az Önemli	Önemli	Daha Önemli	Çok Önemli	En Önemli
7	Kullanıcı Deneyimi	Acemi	Bilgili	Deneyimli	İyi Deneyimli	Uzman	Deneyimli-Uzman
8	Kullanıcı İfade Yeteneği	Pervasız	Kararsız	Sessiz	İletişimli	Etkileyici	Tanımlayıcı
9	Uygulama Geliştirici Deneyim Alanı	Acemi	Bilgili	Deneyimli	İyi Deneyimli	Uzman	Deneyimli-Uzman
10	Yazılım Mühendisliği Deneyim Gelişimi	Acemi	Bilgili	Deneyimli	İyi Deneyimli	Uzman	Deneyimli-Uzman
11	Sistem Güvenliği Farkındalığı	Önemsiz	Kıt	Sınırlı	Az Yeterli	Yeterli	Çok
12	Kullanıcı Katılımı & Güvenlik	Önemsiz	Kıt	Sınırlı	Az Yeterli	Yeterli	Çok
13	Finansman	Düzensiz	Düşük-Sabit	Düşük-Yüksek	Yüksek-Düşük	Yüksek-Kararlı	Yüksek-Artış
14	Para Kullanılabilirliği	Önemsiz	Kıt	Sınırlı	Az Yeterli	Yeterli	Çok
15	Personel Durumu	Düzensiz	Düşük-Kararlı	Düşük-Yüksek	Yüksek-Düşük	Yüksek-Kararlı	Yüksek-Artış
16	Personel Kullanılabilirliği	Önemsiz	Kıt	Sınırlı	Az Yeterli	Yeterli	Çok
17	Kullanıcı Erişimi	Yok	Kısıtlayıcı	Sınırlı	Orta	Kontrollü	Çok
18	Teknoloji Profili	Küçük	Yetersiz	Yetersiz	Yeterli	Çok	Çok
19	Teknoloji Kullanılabilirliği	Önemsiz	Kıt	Sınırlı	Az Yeterli	Yeterli	Çok
20	Teknoloji Oranları	Uygun	Değişken	Toplu	Zamanlı	İyi-Zamanlı	Çevrimiçi
21	Bağımsız Teknoloji Etkileşimi	Önemsiz	Çok Düşük	Düşük	Yüksek	Çok Yüksek	En Yüksek
22	Ürün Kullanım Şartı	Önemsiz	Az	Yararlı	Önemli	Kritik	Titiz
23	Ürün Boyutu	Çok Küçük	Küçük	Orta	Büyük	Çok Büyük	Aşırı

24	Ürün Karmaşıklığı	Önemsiz	Basit	Talebe Bağlı	Zor	Karmaşık	Zor
25	İnsan Ara yüzü Şartı	Önemsiz	Küçük	Yararlı	Önemli	Kritik	Titiz
26	Ürün Mobilite Gereksinimi	Önemsiz	Az	Yararlı	Önemli	Kritik	Titiz
27	Beklenen Ömrü	Atılabilir	Çok Kısa	Kısa	Uzun	Çok Uzun	Sonsuz
28	Güvenlik Gereksinimi	Önemsiz	Az	Yararlı	Önemli	Kritik	Titiz
29	Performans Gereksinimi	Önemsiz	Az	Yararlı	Önemli	Kritik	Titiz
30	Kullanılabilirlik Profili	Düzensiz	Düşük-Kararlı	Düşük-Yüksek	Yüksek-Düşük	Yüksek-Kararlı	Yüksek-Artış
31	Yönetim Yeteneği	İlgisiz	Kılavuzlu	Esnek	Önemli	Zorunlu	Tam
32	Kalite Güvencesi ve Konfigürasyon Yönetim Yeteneği	Önemsiz	Basit	Orta	Önemli	Gelişmiş	Tam
33	Güven Yönetimi	Önemsiz	Çok Düşük	Düşük	Yüksek	Çok Yüksek	En Yüksek
34	Risk Yönetimi	Önemsiz	Çok Düşük	Düşük	Yüksek	Çok Yüksek	En Yüksek
35	Güvenlik Kontrol Yönetimi	Önemsiz	Çok Düşük	Düşük	Yüksek	Çok Yüksek	En Yüksek

Şekil 4.1. SPSM için otuz-beş parçalık ve altı fonksiyonel nokta değerli kriter seti

SPSM için en iyi seçim kriterleri analizi ölçekleme işlemi beş adımda anlatılmıştır:

Adım 1: Projenin özellikleri çok dikkatli bir biçimde analiz edildikten sonra projeyi en iyi tanımlayabilecek her kriter için 0 veya 1'lik ikili gösterge belirlenir. Bir (1) , özel yazılım projesi için sürecin o fonksiyon kriterinin uygun olduğunu gösterirken sıfır(0) ise o fonksiyon kriterinin uygun olmadığını gösterir.

Adım 2: Her satır için, Ci fonksiyon nokta değerlerinin toplam puanlarını hesaplar. Düşük-orta değerleri satırındaki ilk üç fonksiyon noktası değerleri ile yüksek-orta değerleri satırındaki son üç fonksiyon noktası değerlerini alır ve her üç fonksiyon noktası değerlerinin maksimum puanı ile toplar.

Adım 3: Düşük-orta değerleri ile yüksek-orta değerlerinin toplamları hesaplanır ve kriterlere bağlı istenilen model ortaya çıkarılır. İstenilen modelin türetimi tamamen müşterinin kendi projesine bağlı olarak düşünülür.
Adım 4: Bütün bu işlemlerden sonra süreç modelleri için bir sıralama yapılır. Bütün elde edilen düşük-orta puanlar ile istenilen düşük-orta puanlar ve elde edilen yüksek-orta puanlar ile istenilen yüksek-orta puanlar karşılaştırılır.

Adım 5: Final puanları sıralanır. Türetilen sonuçlar ölçekleme sıralarının bulunması için değerlendirilir. İstenile yüksek-orta puanına yakın en iyi türetilen yüksek-orta puanlı süreç en iyisi seçilir. İstenilen yüksek-orta puanın altında kalan puanlar en az dikkate alınanlardır ve güvenli bir yazılım süreç modeli seçiminde istenilen düşük-orta puanlar pek yararlı değildir [11].

Bir yazılım projesinin karakteristiğini anlayabilmek için, yazılımın geliştirildiği ortamın, etrafını saran çevresel faktörlerin ve yazılımı geliştiren organizasyonun dikkatli bir şekilde analiz edilmesi gerekmektedir.

4.3. Süreç Modellerinin Karşılaştırılması

Oluşturulan altı fonksiyonel nokta değerli ve otuz beş kriterli bir sete göre, seçilen altı yazılım geliştirme süreç modeli değerlendirildi. Bu değerlendirmeye göre oluşturulan kriter tabloları aşağıda sunulmuştur.

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	1	1	1	1	4
C2	1	1	1	1	1	1	6
C3	1	1	0	0	0	0	2
C4	0	0	1	1	1	1	4
C5	1	1	1	1	0	0	4
C6	0	0	0	0	0	0	0
C7	1	1	1	1	1	1	6
C8	1	1	1	1	1	1	6
C9	0	1	1	1	1	1	5
C10	0	1	1	1	1	1	5
C11	1	1	1	0	0	0	3
C12	1	1	0	0	0	0	2
C13	1	1	1	1	1	1	6
C14	1	1	1	1	1	1	6
C15	0	1	0	1	1	1	4
C16	0	0	1	1	1	1	4
C17	0	1	0	0	0	0	1
C18	0	0	1	1	1	1	4
C19	0	0	1	1	1	1	4
C20	1	0	1	0	1	1	4
C21	0	0	0	1	1	1	3
C22	1	0	1	1	1	1	5
C23	1	1	1	1	1	1	6
C24	0	0	1	1	1	1	4
C25	1	1	1	1	1	1	6
C26	0	0	0	1	1	1	3
C27	0	0	0	1	1	1	3
C28	0	0	0	1	1	1	3
C29	0	0	1	1	1	1	4
C30	1	1	1	1	1	1	6
C31	0	0	0	1	1	1	3
C32	0	0	0	1	1	1	3
C33	0	0	0	1	1	1	3
C34	0	0	0	1	1	1	3
C35	0	0	0	1	1	1	3
Toplam							138

Şekil 4.2. Şelale modeli

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	1	1	1	1	4
C2	1	1	0	0	0	0	2
C3	0	0	1	1	1	1	4
C4	0	1	1	1	1	1	5
C5	0	0	0	1	1	1	3
C6	1	1	0	0	0	0	2
C7	1	1	1	1	1	1	6
C8	1	1	1	1	1	1	6
C9	0	1	1	1	1	1	5
C10	0	1	1	1	1	1	5
C11	0	0	1	1	1	1	4
C12	1	1	1	1	1	0	5
C13	1	1	1	1	1	1	6
C14	1	1	1	1	1	1	6
C15	0	1	0	1	1	1	4
C16	0	0	1	1	1	1	4
C17	1	1	1	1	1	1	6
C18	0	0	1	1	1	1	4
C19	1	0	0	1	1	1	4
C20	1	0	1	1	1	1	5
C21	0	0	0	1	1	1	3
C22	1	0	1	1	1	1	5
C23	1	1	1	1	1	1	6
C24	0	0	1	1	1	1	4
C25	0	0	1	1	1	1	4
C26	0	0	0	1	1	1	3
C27	0	0	0	1	1	1	3
C28	0	0	0	1	1	1	3
C29	0	0	1	1	1	1	4
C30	1	1	1	1	1	1	6
C31	0	0	0	0	1	1	2
C32	0	0	0	1	1	1	3
C33	0	0	0	1	1	1	3
C34	0	0	0	1	1	1	3
C35	0	0	0	1	1	1	3
Toplam							145

Şekil 4.3. Artırımsal model

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	1	1	1	0	0	0	3
C2	0	0	1	1	1	1	4
C3	0	1	1	1	1	1	5
C4	0	0	1	1	1	1	4
C5	1	0	1	0	0	1	3
C6	1	0	0	1	0	0	2
C7	0	0	0	0	0	0	0
C8	0	0	1	1	1	1	4
C9	0	1	1	1	1	1	5
C10	0	0	0	1	1	1	3
C11	1	0	0	1	0	0	2
C12	1	0	0	1	0	0	2
C13	0	0	0	1	1	0	2
C14	0	0	0	1	0	0	1
C15	0	0	1	0	1	1	3
C16	0	1	0	1	1	0	3
C17	0	0	0	0	0	1	1
C18	0	0	1	0	1	1	3
C19	1	1	1	1	1	1	6
C20	1	1	1	1	1	1	6
C21	0	0	0	1	1	0	2
C22	0	0	1	1	1	1	4
C23	0	0	1	1	1	1	4
C24	0	0	1	1	1	0	3
C25	0	0	0	1	1	1	3
C26	0	0	0	1	1	1	3
C27	0	0	0	1	1	1	3
C28	0	0	0	1	1	1	3
C29	0	0	1	1	1	1	4
C30	0	0	0	0	1	1	2
C31	1	1	1	1	0	0	4
C32	0	0	0	1	1	1	3
C33	0	0	1	1	0	0	2
C34	0	0	0	1	1	1	3
C35	0	0	1	0	0	0	1
Toplam							106

Şekil 4.4. Yeniden kullanılabilir model

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	1	1	1	1	4
C2	0	0	1	1	1	1	4
C3	0	0	1	1	1	1	4
C4	0	0	0	1	1	1	3
C5	0	0	0	0	1	1	2
C6	1	0	0	0	0	0	1
C7	1	1	1	1	1	1	6
C8	1	1	1	1	1	1	6
C9	0	1	1	1	1	1	5
C10	0	1	1	1	1	1	5
C11	0	1	1	1	1	1	5
C12	1	1	1	1	0	0	4
C13	1	1	1	1	1	1	6
C14	1	1	1	1	1	1	6
C15	0	1	1	1	1	1	5
C16	0	1	1	1	1	1	5
C17	0	0	0	1	1	1	3
C18	0	0	0	1	1	1	3
C19	0	0	0	0	1	1	2
C20	1	0	0	1	1	1	4
C21	0	0	0	1	1	1	3
C22	1	1	1	1	1	1	6
C23	0	0	0	1	1	1	3
C24	0	0	1	1	1	1	4
C25	0	0	1	1	1	1	4
C26	0	0	0	1	1	1	3
C27	0	0	1	1	1	1	4
C28	0	1	0	1	1	1	4
C29	0	0	1	1	1	1	4
C30	0	1	1	1	1	1	5
C31	0	1	1	1	1	1	5
C32	0	0	0	1	1	1	3
C33	0	0	0	1	1	1	3
C34	0	0	0	1	1	1	3
C35	0	0	0	1	1	1	3
Toplam							140

Şekil 4.5. Evrimsel prototip modeli

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	0	1	1	1	3
C2	0	0	1	1	1	1	4
C3	0	1	0	0	0	0	1
C4	0	0	0	1	1	1	3
C5	0	0	0	0	1	1	2
C6	1	0	0	1	0	0	2
C7	0	1	1	1	1	1	5
C8	0	0	0	1	1	1	3
C9	0	0	1	1	1	1	4
C10	0	0	0	1	1	1	3
C11	0	0	0	1	1	1	3
C12	0	0	0	1	1	1	3
C13	0	0	0	0	1	1	2
C14	1	1	1	1	1	1	6
C15	0	1	1	0	1	1	4
C16	0	0	0	1	1	1	3
C17	0	0	0	0	1	1	2
C18	0	1	1	1	1	1	5
C19	1	0	0	1	1	1	4
C20	1	0	0	1	1	1	4
C21	0	0	0	1	1	1	3
C22	0	0	1	1	1	1	4
C23	0	0	0	1	1	1	3
C24	0	0	1	1	1	1	4
C25	0	0	1	1	1	1	4
C26	0	0	1	1	1	1	4
C27	0	0	0	1	1	1	3
C28	0	0	1	1	1	1	4
C29	0	0	1	1	1	1	4
C30	0	0	1	1	1	1	4
C31	0	0	0	1	1	1	3
C32	0	0	0	1	1	1	3
C33	0	0	0	1	1	1	3
C34	0	0	0	1	1	1	3
C35	0	0	0	1	1	1	3
Toplam							118

Şekil 4.6. Helezonik (Sarmal) model

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	0	0	0	0	0
C2	1	1	0	0	0	0	2
C3	0	0	1	1	1	1	4
C4	0	0	0	1	1	1	3
C5	0	0	0	0	1	1	2
C6	1	0	0	0	0	0	1
C7	0	1	0	0	0	0	1
C8	1	1	0	1	0	0	3
C9	0	1	1	1	1	1	5
C10	0	1	1	1	1	1	5
C11	0	1	1	0	0	0	2
C12	0	0	0	1	1	1	3
C13	0	0	0	1	1	1	3
C14	0	0	0	1	1	1	3
C15	0	0	1	0	1	1	3
C16	0	1	1	0	0	0	2
C17	0	1	1	0	1	1	4
C18	0	0	1	0	0	1	2
C19	0	1	1	1	1	1	5
C20	0	0	0	1	1	1	3
C21	0	0	0	1	1	0	2
C22	0	0	0	1	1	1	3
C23	0	0	0	1	1	1	3
C24	0	0	1	0	1	1	3
C25	0	0	0	1	1	1	3
C26	0	0	0	1	1	1	3
C27	0	0	0	1	1	1	3
C28	0	0	0	1	1	1	3
C29	0	0	0	1	1	1	3
C30	1	0	1	0	1	1	4
C31	1	1	1	0	0	0	3
C32	0	0	0	1	1	1	3
C33	1	1	1	0	0	0	3
C34	1	1	1	0	0	0	3
C35	1	1	1	0	0	0	3
Toplam							101

Şekil 4.7. Hızlı prototip model

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	1	1	1	1	4
C2	1	1	1	1	1	1	6
C3	1	1	1	1	1	1	6
C4	1	1	1	1	1	1	6
C5	1	1	1	1	1	1	6
C6	0	1	1	1	1	1	5
C7	1	1	1	1	1	1	6
C8	1	1	1	1	1	1	6
C9	0	1	1	1	1	1	5
C10	0	0	0	1	1	1	3
C11	0	0	0	1	1	1	3
C12	0	0	0	1	1	1	3
C13	0	0	0	1	1	1	3
C14	0	0	0	1	1	1	3
C15	0	0	0	0	1	1	2
C16	0	0	0	0	1	1	2
C17	0	0	0	1	1	1	3
C18	0	0	0	0	1	1	2
C19	0	0	0	0	1	1	2
C20	0	0	0	0	1	1	2
C21	0	0	0	1	1	1	3
C22	0	0	1	1	1	1	4
C23	1	1	1	1	1	1	6
C24	1	1	1	1	1	1	6
C25	0	0	0	0	1	1	2
C26	0	0	0	0	1	1	2
C27	0	0	1	1	1	1	4
C28	0	0	0	0	1	1	2
C29	0	0	0	1	1	1	3
C30	1	0	1	1	1	1	5
C31	0	0	1	1	1	1	4
C32	0	0	0	1	1	1	3
C33	0	0	0	1	1	1	3
C34	1	1	0	0	0	0	2
C35	0	0	0	0	1	1	2
Toplam							128

Şekil.4.8. İstenilen güvenli model

Kriter setine göre olması gereken güvenli bir yazılım geliştirme süreç modeli oluşturuldu ve altı yazılım geliştirme süreç modeli de istenilen güvenli modele göre karşılaştırılarak değerlendirildi. Bu değerlendirme sonucunda da içlerinden en güvenli ve de en güvensiz yazılım geliştirme süreç modeli belirlendi.

BÖLÜM 5. SONUÇLAR VE YÖNTEMİN ÖRNEK YAZILIM PROJELERİNE UYGULANMASI

Bu çalışmamızda yazılım geliştirme süreç tekniklerinin yazılım geliştirme hayat döngüsü içerisindeki öneminden bahsedildi. Süreç modelleri arasından altı tanesi detaylı olarak anlatıldı. Bunlar: Çağlayan (Şelale) Modeli, Artırımsal Model, Yeniden Kullanılabilir Yazılım Modeli, Prototip Modeli, Helezonik (Sarmal) Model ve Evrimsel Modeldir. Bu modelleri karşılaştırmak için altı fonksiyonel değerli otuz beş parçalık bir kriter seti oluşturuldu. Yine bu sete göre olması gereken istenilen güvenli yazılım geliştirme modeli oluşturuldu ve bu altı yazılım geliştirme süreç modeli de kriter setine uyarlanarak olması gereken yazılım geliştirme süreç modeline göre karşılaştırıldı.

Karşılaştırmanın sonucunda da çok büyük projelerde uygulanabilirliği açısından, yazılım sürecinde kullanıcının projeyi her aşamada test edebiliyor ve de proje yöneticisinin projeyi her aşamada test edebiliyor olmasından dolayı en çok tercih edilen model ve en güvenilir yazılım geliştirme modeli olarak belirlendi. Prototiplerin yavaş ve hantal programlar olması ve de müşterilerin geliştirilen prototipi geçek program gibi algılaması ihtimali olduğundan prototip modeli en az tercih edilen ve de en güvensiz yazılım geliştirme süreç modeli olarak belirlendi.
Bu modellerin seçilmesinde Şekil 4.1'de belirtilen kriterlere göre birbirine bağlı dağıtılan sistemlerde en iyi güvenlik konusunu ele alacak olan kriter Yüksek-Orta (f4-f5-f6) toplamlarıdır çünkü bu çözüm için en iyi alan adreslerini sunar.

5.1. Karşılaştırma Sonuçları

Kurulmuş bir uygulama alanında yazılım geliştirmek geliştiren takım ve karmaşık sistem sürümleri için büyük fayda sağlamaktadır. Aynı zamanda kısmi işlevler gereksinimi için, birçok müşteri hedef ürünün tam işlevsellik kazanmasından önce pratik bir orta dereceli ürün talep etmektedir.
Deneyimlere göre yazılım mühendisliği ve yazılım ürünleri her zaman üretim sırasındaki değişikliklere bağlı kalırlar. Kısacası henüz ilk aşama için hiçbir

değişiklik yapmadan yazılımı üretmek hala daha Yüksek-Orta Alana doğru eğilmektedir.

Etkili güvenlik yönetimi başlangıçtan yazılım geliştirme döngüsüne kadar sistem yönetimine uyum sağlamak zorundadır. Bu nedenle güvenlik problem tanımlama aşamasında doğru olarak bulunmalıdır. Yeterli seçenek ve uygun teknik kontrollerinin uygulanması ile problem tanımındaki zayıflıklarla ilgilenen güvenlik yordamları olmak zorundadır.

İstenilen süreç modelindeki güvenlik ölçümleri Yüksek-Orta alana doğru eğimlidir. Verilen yazılım projesindeki takım yönetimine olan güvenin derecesi yüksek olmalıdır. Eğer güven düşük olursa, proje başlamadan önce bütün takım yönetimi değiştirilmelidir yoksa sözde güvenilir çalışan sistem veritabanı bütünlüğüne ve sistem mimarisine zarar verecek hatalar oluşturur. Güvenlik proje sistem yaşam döngüsü içinde yazılım geliştiren tüm katılımcılar arasında sağlanmalıdır. Bu kriter güvenliğin gözden kaçırmadığına emin olmak için bir güvenlik planı inşaatını tetikler.

Kullanıcıların davranışları ve anlatım kabiliyetleri, Yüksek-Orta alana dayalı bir takım tasarlamak için çok yardımcı ve takdir edilen olacaktır. Ayrıca problem alanında deneyimli ve uzman olmak; yazılım araçları, yöntemleri, teknikleri, ürün ve geliştirme aşamasındaki diller hakkında temel bilgi sahibi olmak ile de daha iyisi yapılacaktır [11].

Yazılım geliştirmede kullanılabilir başka kaynaklar da vardır; kaynak personel, para kaynağı, kullanılabilir teknoloji ve yazılım geliştirme takımının görevlerini zamanında daha etkili bir şekilde yerine getirebilmek ve teknolojik bilginin dağılımı sağlayacak olan artış oranı.

	Süreç Modeli	Elde Edilen Düşük-Orta	İstenilen Düşük-Orta	Elde Edilen Yüksek-Orta	İstenilen Yüksek-Orta	Sıralama
1	Şelale	51	36	87	92	4.(-5) düşük
2	Artırımsal	48	36	97	92	2. (5)
3	Yeniden Kullanılabilir	33	36	73	92	5.(-19) düşük
4	Evrimsel Prototip	42	36	98	92	3.(6)
5	Sarmal	22	36	96	92	1. (4)
6	Hızlı Prototip	36	36	65	92	6.(-27) düşük

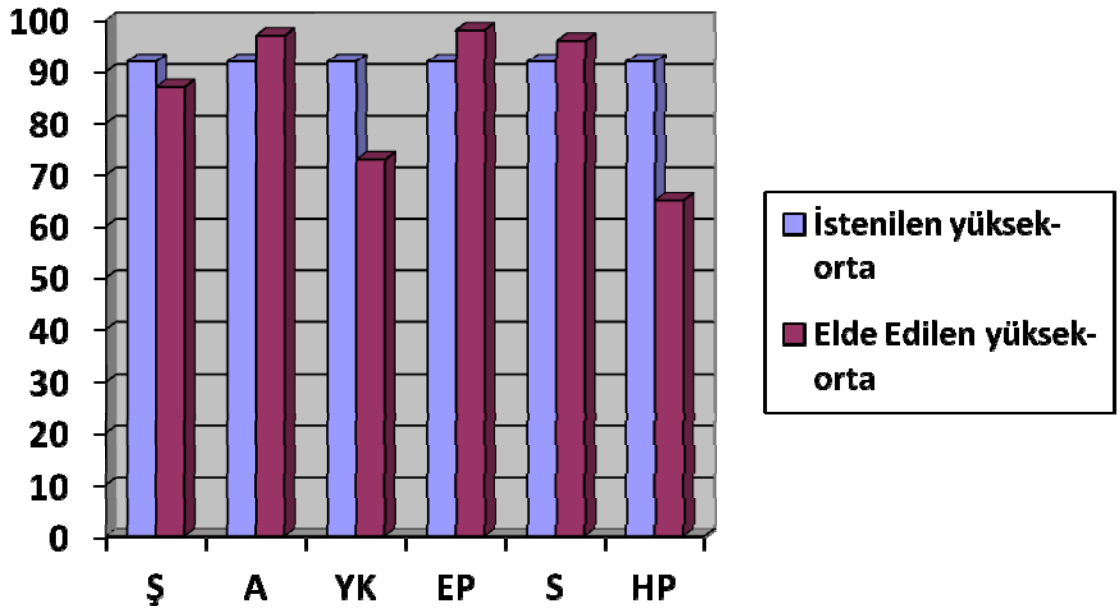
Şekil 5.1. Süreç modellerinin değerlendirme sırası

Şekil 5.1'de görüldüğü gibi istenilen yüksek-orta puan ile elde edilen yüksek-orta puan arasında 4 puan farkla en yüksek oranı alan model Sarmal Model'dir. Onu da Artırımsal Model ve sonra da 98 puanla Evrimsel Prototip Model takip etmektedir. İstenilen yüksek-orta puandan en düşük yüksek-orta puan elde edilen Hızlı Prototip Model sonuncu olmuştur. Bizim ölçekleme mekanizmamıza göre de Hızlı Prototip en düşük güvenli yazılım süreç modeli olmuştur.

Sarmal model çok büyük projelerde uygulanabilirliği açısından daha geçekçi ve kullanılışlı bir model olarak görülmektedir. Yazılım sürecinde kullanıcının projeyi her aşamada test edebiliyor olması, proje yöneticisinin projeyi her aşamada gözlemleyebiliyor olması bu modeli en çok tercih edilen model haline getirmektedir.

Prototip modellerde prototiplerin yavaş ve hantal programlar olmaları sebebiyle gerçek uygulamalarda kullanılabilirliği söz konusu olmuyordu. Fakat müşterilerin geliştirilen prototipi gerçek proje gibi algılaması durumu söz konusuydu. Bu da müşteri memnuniyetsizliğine neden olmaktaydı. Bu yüzden prototip geliştirmenin yalnızca gereksinimlerin doğru belirlenmesi için olduğunun başta kullanıcının iyice algılamış olması gerektiği için de prototip modeli en düşük güvenli yazılım süreç modeli olmaktadır.

Şekil 5.2 Altı süreç modeli arasındaki ilişkiyi göstermektedir.



Şekil 5.2. Elde edilen yüksek-orta puanlar ile istenilen yüksek-orta puanlar arasındaki ilişki

Ş : Şelale Modeli
A : Artırımsal Model
YK: Yeniden Kullanılabilir Model
EP : Evrimsel Prototip Model
S : Sarmal Model
HP : Hızlı Prototip Model

Şekil 5.2'de karşılaştırdığımız süreç modelleri ile istenilen güvenli model arasındaki farklar grafiksel olarak gösterilmiştir. Sarmal model istenilen modele en yakın model olduğu için fark en azdır. Bu yüzden de en güvenilir yazılım geliştirme süreç modeli seçilmiştir. Hızlı prototip modelin de istenilen modele en uzak model olduğu için fark en yüksektir. Bu yüzden de en güvensiz yazılım geliştirme süreç modeli seçilmiştir.

Bu çalışmada, bir yazılım geliştirme projesinin güvenliği için kullanılması gereken yazılım süreç geliştirme modelleri incelendi. Her bir kriter için deneysel olarak ikili göstergeler kullanılarak projenin özelliklerini en iyi yansıtan model dikkatlice seçildi. Her satır için, Ci fonksiyon nokta değerlerinin toplam puanlarını hesapladı.

Düşük-orta değerleri satırındaki ilk üç fonksiyon noktası değerleri ile yüksek-orta değerleri satırındaki son üç fonksiyon noktası değerlerini aldı ve her üç fonksiyon noktası değerlerinin maksimum puanı ile topladı. Daha sonra da müşterinin istekleri göz önüne alınarak istenilen model ortaya çıkarıldı. Elde edilen düşük-orta değerler ile istenilen düşük-orta değerler ve elde edilen yüksek-orta değerler ile istenilen yüksek-orta değerler karşılaştırıldı. En sonunda da istenilen yüksek-orta değere yakın elde edilen yüksek-orta değerli süreç en iyisi seçildi.

Bu çalışma da süreç modelleri arasındaki ilişkileri ölçebilmek için otuz beş parçalık altı fonksiyonel noktalı bir kriter seti oluşturuldu. Daha iyi gözlemleyebilmek için bu kriter setinin parçalarını artırmak daha faydalı olabilir. Örnek olarak daha fazla canlı yazılım projeleri kriter setine göre incelenmesi de en çok tercih edilen güvenli yazılım süreç geliştirme modelinin belirlenmesinde daha çok yol gösterici olabilir. Yine aynı şekilde süreç modelleri arasındaki ilişkileri ölçüp en iyi sonucu belirleyebilmek için veri madenciliği veya bulanık mantık algoritmaları da bu konuda yol gösterici olabilir.

5.2. Örnek Uygulamalar

5.2.1. Karşılaştırma İşlemleri İçin Oluşturulan Algoritma

Projenin özellikleri çok dikkatli bir biçimde analiz edildikten sonra projeyi en iyi tanımlayabilecek her kriter için 0 veya 1'lik ikili gösterge belirlenir. Bir (1) , özel yazılım projesi için sürecin o fonksiyon kriterinin uygun olduğunu gösterirken sıfır(0) ise o fonksiyon kriterinin uygun olmadığını gösterir.

Karşılaştırma işlemleri yapılırken öncelikle proje ekibinden bir kişiye Şekil 4.1'deki sorular sorulur. Alınan yanıtlara göre fonksiyonel noktalara "0" ve "1" değerleri atanır. Daha sonra bütün "1" değerleri toplanır. Elde edilen sonuca göre incelenen projenin hangi süreç modeline yakın olduğu belirlenir. Öncelikle kriter tablosunu oluşturduğumuz süreç modellerinin kriter toplamı sonuçlarını küçükten büyüğe doğru sıralanışı şu şekildedir:

	Süreç Modeli	Elde Edilen Düşük-Orta	Elde Edilen Yüksek-Orta	Toplam
1	Hızlı Prototip	36	65	101
2	Yeniden Kullanılabilir	33	73	106
3	Sarmal	22	96	118
4	Şelale	51	87	138
5	Evrimsel Prototip	42	98	140
6	Artırımsal	48	97	145

Sekil.5.3. Süreç modelleri hesaplamaları

İncelenen projenin kriter toplamının hangi modele yakın olduğunu görmek için yapılan matematiksel işlemleri ve oluşturulan algoritmanın adım adım açıklanması şu şekildedir:

1. Hızlı Prototip Model ile hemen altındaki Yeniden Kullanılabilir Model'in sonuçlarının ortalaması bulundu: (101 + 106) / 2 =103.5

103. 5 ondalıklı bir değer olduğu için şuna karar verildi: Eğer bulunan değer (sonuç) 0 ile 103 arasında ise; yani

(0 <= sonuç <= 103) ise bu model "Hızlı Prototip Model' e yakındır."

2. Yeniden Kullanılabilir Model ' in değeri, Hızlı Prototip Model' in sınırı 103 olarak belirlendiğinden 104 ile başlatıldı ve (106 + 118) / 2= 112 olduğundan 112 ile sınırlandırıldı. Alınan karar:

(104<= sonuç <= 112) ise "bu model Yeniden Kullanılabilir Model' e yakındır."

3. Helezonik (Sarmal) Model için:

(118 + 138) / 2= 128 olduğundan (113 <= sonuç <= 128) ise "bu model Helezonik (Sarmal) Model' e yakındır."

4. Şelale (Çağlayan) Modeli için;

(138 + 140) / 2= 139 olduğundan (129 <= sonuç <= 139) ise "bu model Şelale (Çağlayan) Model' e yakındır."

5. Evrimsel Prototip Model için:

(140 + 145) / 2= 142,5 olduğundan (140 <= sonuç <= 142) ise “bu model Evrimsel Prototip Model’ e yakındır.”

6. Artırımsal Model için:

(sonuç >= 143) ise “bu model Artırımsal Model’ e yakındır.” Denilebilir.

Toplamları birbirlerine yakın olan süreç modellerinde elde edilen Düşük-Orta ve elde edilen Yüksek-Orta değerlerine de bakılması gerekir.

5.2.2. Uygulama 1

İncelenen Proje: MRP Uygulaması

Malzeme ihtiyaç planlama, üretim aşamasından gelen ham maddenin just in time a göre tezgâhlara alınmasını sağlayan tedarik ve malzeme yönetim sistemidir.

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	0	1	1	0	2
C2	0	0	0	1	1	1	3
C3	1	1	1	1	1	1	6
C4	1	1	1	0	0	0	3
C5	0	0	1	1	1	1	4
C6	0	1	1	1	1	0	4
C7	1	1	1	1	1	1	6
C8	1	1	1	1	1	1	6
C9	0	0	0	1	1	1	3
C10	0	0	0	0	1	1	2
C11	0	0	0	1	1	1	3
C12	0	0	0	1	1	1	3
C13	0	0	1	1	1	0	3
C14	0	1	1	1	0	0	3
C15	0	0	1	0	0	0	1
C16	0	0	0	1	1	1	3
C17	0	0	1	1	1	0	3
C18	0	0	0	0	0	1	1
C19	0	0	0	1	1	1	3
C20	0	0	0	1	1	1	3
C21	0	0	0	0	1	1	2
C22	0	0	0	1	1	1	3
C23	0	0	1	1	1	1	4
C24	0	0	1	1	1	1	4
C25	0	1	1	1	0	0	3
C26	0	0	0	0	1	1	2
C27	0	0	0	0	1	1	2
C28	1	1	1	0	0	0	3
C29	0	0	0	1	1	1	3
C30	0	0	0	1	1	1	3
C31	0	0	0	1	1	1	3
C32	1	1	0	0	0	0	2
C33	0	0	0	0	1	1	2
C34	1	1	0	0	0	0	2
C35	0	1	1	0	0	0	2
Toplam							105

Şekil.5.4. Örnek süreç modeli 1

5.2.2.1. Kullanılan süreç modelinin bulunması

Elde Edilen Düşük-Orta: 33
Elde Edilen Yüksek-Orta: 72

İncelenen uygulamada elde edilen Düşük-Orta, Yüksek-Orta ve toplam değerlerine bakılarak Yeniden Kullanılabilir Yazılım geliştirme Süreç Modeli kullanıldığını söyleyebiliriz.

5.2.3. Uygulama 2

İncelenen Proje: Balans – SkorCard Uygulaması

Şirket hedeflerini en üst seviyeden en alt seviyeye kadar tüm personele göre görevlendiren ve verilen görevlerin zamanında yapılıp yapılmadığını araştırıp verilen hedefe katkısını inceleyen yazılımdır.

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	0	0	1	1	2
C2	0	0	1	1	1	1	4
C3	1	1	1	1	1	0	5
C4	1	1	1	1	1	1	6
C5	0	1	1	1	1	1	5
C6	0	0	1	1	1	1	4
C7	0	1	1	1	1	1	5
C8	1	1	1	1	1	1	6
C9	0	0	1	1	1	1	4
C10	0	0	0	0	1	1	2
C11	0	0	1	1	1	1	4
C12	0	1	1	1	1	1	5
C13	1	1	1	0	0	0	3
C14	1	1	1	0	0	0	3
C15	1	1	1	1	1	0	5
C16	1	1	1	1	0	0	4
C17	1	1	1	1	1	1	6
C18	0	0	0	1	1	1	3
C19	0	0	1	1	1	1	4
C20	0	0	0	1	1	1	3
C21	0	0	0	0	1	1	2
C22	0	0	1	1	1	1	4
C23	0	1	1	1	1	0	4
C24	0	1	1	1	1	1	5
C25	0	1	1	1	1	0	4
C26	0	0	1	1	1	1	4
C27	0	0	0	1	1	1	3
C28	0	0	0	1	1	1	3
C29	0	0	0	1	1	1	3
C30	0	0	0	0	1	1	2
C31	0	1	1	1	1	1	5
C32	0	0	1	1	1	1	4
C33	0	0	0	0	1	1	2
C34	1	1	1	1	0	0	4
C35	0	0	1	1	1	1	4
Toplam							136

Şekil.5.5. Örnek süreç modeli 2

5.2.3.1. Kullanılan süreç modelinin bulunması

Elde Edilen Düşük-Orta: 50
Elde Edilen Yüksek-Orta: 86

İncelenen uygulamada elde edilen Düşük-Orta, Yüksek-Orta ve toplam değerlerine bakılarak Şelale Yazılım geliştirme Süreç Modeli kullanıldığını söyleyebiliriz.

5.2.4. Uygulama 3

İncelenen Proje: CRM Uygulaması

Gelen işlerin projelendirilip ürün ağaçlarına göre parçalanarak personele dağıtıldığı; personelin yapılan işleri, gereken malzemeleri ve tarihlerini girdiği; her proje için kullanılan malzemenin depoda takip edildiği; bunlara ek olaraktan bütçe planlamasının yapıldığı yazılımdır.

Ci	F1	F2	F3	F4	F5	F6	Toplam
C1	0	0	0	1	1	1	3
C2	0	0	0	1	1	1	3
C3	0	1	1	1	1	1	5
C4	0	1	1	1	1	0	4
C5	0	1	1	1	1	1	5
C6	0	0	0	1	1	1	3
C7	0	1	1	1	1	1	5
C8	1	1	1	1	1	1	6
C9	0	0	0	1	1	1	3
C10	0	0	0	1	1	1	3
C11	0	0	0	1	1	0	2
C12	0	0	0	1	1	1	3
C13	0	0	0	1	1	1	3
C14	0	0	0	0	1	1	2
C15	0	1	1	1	0	0	3
C16	0	1	1	1	0	0	3
C17	0	1	1	1	1	0	4
C18	0	0	0	0	1	1	2
C19	0	0	0	1	1	1	3
C20	0	0	0	1	1	1	3
C21	0	0	0	0	1	1	2
C22	0	0	0	1	1	1	3
C23	0	0	1	1	1	0	3
C24	0	0	1	1	1	1	4
C25	0	1	1	1	1	0	4
C26	0	0	0	1	1	1	3
C27	0	0	0	0	1	1	2
C28	0	0	0	1	1	1	3
C29	0	0	1	1	1	1	4
C30	0	0	0	0	1	1	2
C31	0	0	1	1	1	1	4
C32	0	0	0	0	1	1	2
C33	0	0	0	1	1	1	3
C34	0	0	0	1	1	1	3
C35	0	0	0	1	1	1	3
Toplam							113

Şekil.5.6. Örnek süreç modeli 3

5.2.4.1. Kullanılan süreç modelinin bulunması

Elde Edilen Düşük-Orta: 23
Elde Edilen Yüksek-Orta: 90

İncelenen uygulamada elde edilen Düşük-Orta, Yüksek-Orta ve toplam değerlerine bakılarak Helezonik(Sarmal) Yazılım geliştirme Süreç Modeli kullanıldığını söyleyebiliriz.

BÖLÜM 6. TARTIŞMA VE ÖNERİLER

Yazılım süreç seçim kriterlerinin seçilebilmesi için herhangi basit bir yaklaşım bulunmamaktadır. Çeşitli yazılım uygulamaları ve süreçleri incelendiğinde, değişik yazılım projeleri için bir seçim yöntemi tanımlamak için bazı kurallar tanımlanmış olduğu görülmektedir. Bir gerçek vardır ki bir proje için yeterli olan diğer bir proje için yetersiz olmaktadır.

Bu çalışma da süreç modelleri arasındaki ilişkileri ölçebilmek için otuz beş parçalık altı fonksiyonel noktalı bir kriter seti oluşturuldu. Daha iyi gözlemleyebilmek için bu kriter setinin parçalarını artırmak daha faydalı olabilir. Kriter setine göre belirlenen istenilen yazılım geliştirme süreç modeli de yazılım türlerine göre farklılık gösterebilir. Örnek olarak daha fazla canlı yazılım projeleri kriter setine göre incelenmesi de en çok tercih edilen güvenli yazılım süreç geliştirme modelinin belirlenmesinde daha çok yol gösterici olabilir. Uygulamamızdaki kriter tablomuzun sınırları çok keskin olduğundan bulanık mantık algoritmaları da uygun yazılım geliştirme süreç modelini seçmek için kullanılabilir.

Bir yazılım projesinin karakteristiğini anlayabilmek için, yazılımın geliştirildiği ortamın, etrafını saran çevresel faktörlerin ve yazılımı geliştiren organizasyonun dikkatli bir şekilde analiz edilmesi gerekmektedir. Yazılım projesine başlamadan önce yazılım ihtiyaçları göz önüne alınıp güvenliği için çalışmalara başlanmalıdır.

KAYNAKLAR

[1] OLSON, T.G., REIZER, N.R., OVER, J.W., Handbook CMU/SEI-94-HB-01: A Software Process Framework for the SEI Capability Maturity Model, SEI, September 1994.

[2] GHEZZI, C., JAZAYERI, M., MANDRIOLI, D., Fundamentals of Software Engineering, Prentice Hall, Pearson Education Inc., New Jersey, 2003.

[3] http://www.bilgininadresi.net/Madde/9274/Yazılım-Geliştirme-Sürecinin-Verimliliğini-Arttırmak:-Bir-Bilgi-Sistemi-Önerisi, 2010.

[4] NAUR, P., RANDALL, B., "Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee", NATO, 1968.

[5] PARNAS, D.L., "Software Engineering Programs Are Not Computer Science Programs", IEEE Software, November/December 1999, s. 19-30.

[6] POUR, G., GRISS, M. , LUTZ, M. "The Push to Make Software Engineering Respectable", IEEE Computer, May 2000, s. 35-43.

[7] ABRAN, A., MOORE, J., "Guide to the Software Engineering Body of Knowledge SWEBOK". IEEE Computer Society Press, 2001.

[8] GHOSH, A., MCGRAW, G., CHARRON, F., MILLER, E., Defining an Adaptive Software Security Metric from a Dynamic Software Failure Tolerance Measure.

[9] ONIBERE, E.A., EKUBASE, G. O., Enhanced Software Process Selection Criteria. Jour. Inst. Math and Computer Sciences (Comp. Sc. Ser.), 2006, 17, (1):17-32.

[10] ALEXANDER, L., DAVIS, A., "Criteria For Selecting Software Process Models", 1991.

[11] DAVIS, A. M., BERSOFF, E.H., COMER, E. R., A strategy for comparing alternative software development life cycle models. Software engineering, 1988, s. 1453-1461.

[12] BROOKS, F.P., "No Silver Bullet Essence and Accidents of Software Engineering", Computer Magazine, 1987.

[13] REDWINE, S. T., DAVIS, N., "Processes to Produce Secure Software." Improving Security across the Software Development Lifecycle, APPENDIX, B., 2004.

[14] CMMI yearly report, http://www.sei.cmu.edu/ appraisal-program/profile/pdf/CMMI/2009MarCMMI.pdf, 2009.

[15] Yazılım Mühendisliği, ARIFOĞLU, Ali, DOĞRU, Ali, Ankara, 2000.

[16] www.btom.org.tr/Duyuru/BTOM-VMODEL.doc. (Erişim tarihi: 2010)

[17] IABG Information Technology Web Site (V-Model Lifecycle Process Model). (Erişim tarihi : 2004)

[18] http://www.rhxo.com.tr/surecler/yazilim-muhendisligi.html. (Erişim tarihi: 2010)

[19] http://www.mehmetduran.com/detay.aspx?detay=299. (Erişim tarihi: 2011)

[20] www.bilmuh.gyte.edu.tr/BIL441/studentfiles. (Erişim tarihi: 2010)

[21] web.iku.edu.tr/~gyilmaz/Notes/YazilimMuhendisligiYonetimi/Bolum-02.ppt. (Erişim tarihi: 2011)

[22] http://tbd.wmv.gen.tr/Bilisim08/Bildiriler/MERT%20BI%C7AKCI.doc. (Erişim tarihi: 2009)

[23] http://www.bilgiguvenligi.gov.tr/bt-guv.-standartlari/guvenli-yazilim-gelistirme-modelleri-ve-ortak-kriterler-standardi.html. (Erişim tarihi: 2011)

[24] ALPARSLAN, Erdem, "Güvenli Yazılım Geliştirme Modelleri", TÜBİTAK-UEKAE, http://www.bilgiguvenligi.gov.tr/teknik-yazilar-kategorisi/guvenli-yazilim-gelistirme-modelleri.html. (Erişim tarihi: Aralık 2009)

[25] http://www.bilgiguvenligi.gov.tr/yazilim-guvenligi/yazilim-gelistirme-surecleri-ve-iso-27001-bilgi-guvenligi-yonetim-sistemi.html. (Erişim tarihi: 2010)

[26] http://www.ceturk.com/muhendislik/yazilim_muhendisligi/agilecevik-modelleme-ve-cevik-yazilim-gelistirme.html. (Erişim tarihi: 2010)

[27] http://en.wikipedia.org/wiki/Agile_Modeling. (Erişim tarihi: 2011)

[28] http://www.genisbilgi.com/yazilim-muhendisligi/2554-yazilim-gelistirme-yasam-dongusu.html. (Erişim tarihi: 2010)

[29] http://ftp.cs.hacettepe.edu.tr/pub/dersler/BIL3XX/BIL346_SGM/06-07/SGvM2007YazGelSureci.doc. (Erişim tarihi: 2009)

[30] http://www.bilgisayarkavramlari.com/2008/11/29/selale-modeli-waterfall-model.

[31] http://www.kurumsaljava.com/2009/02/26/yazilimda-selale-waterfall-yontemi.

[32] http://www.programlama.com/sys/c2html/view.php?DocID=7199. (Erişim tarihi: 2009)

[33] http://www.bilgiyonetimi.org/cm/pages/mkl_gos.php?nt=55. (Erişim tarihi: 2009)

[34] ftp.cs.hacettepe.edu.tr/pub/dersler/BSS6XX/BSS651/08-09/BBS651YazMuh2.doc (Erişim tarihi: 2009)

ÖZGEÇMİŞ

Şeyda OCAK, 13.03.1984'te İstanbul'da doğdu. İlköğrenimini Kartal Merkez Eczacıbaşı İlköğretim Okulu'nda, orta öğrenimini İstek Özel Uluğbey Lisesi'nde, lise öğrenimini de Burak Bora Anadolu Lisesi'nde tamamladı. 2002 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2006 yılında mezun oldu. 2006 Ekim – 2011 Mart yılları arasında Gebze Organize Sanayi Bölgesi-Teknopark bünyesinde bulunan Kusman Teknoloji adlı şirkette Yazılım Geliştirme Uzmanı olarak çalıştı. 2011 Haziran ayından itibaren de bir sigorta şirketinde İş Zekâsı Uzmanı olarak çalışmaktadır.