YAZILIM GELİŞTİRME MODELLERİNİN GÜVENLİK AÇISINDAN ANALİZİ VE BİR GÜVENLİ YAZILIM GELİŞTİRME MODELİ ÖNERİSİ

ANALYSIS OF SOFTWARE DEVELOPMENT MODELS IN TERMS OF SECURITY AND A MODEL PROPOSAL FOR SECURE SOFTWARE DEVELOPMENT

GÜLER KOÇ

YRD. DOÇ. DR. MURAT AYDOS
Tez Danışmanı

Hacettepe Üniversitesi
Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin
Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü
YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2017

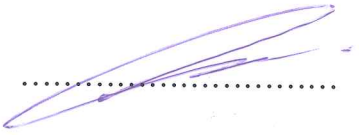
GÜLER KOÇ'un hazırladığı "**Yazılım Geliştirme Modellerinin Güvenlik Açısından Analizi ve Bir Güvenli Yazılım Geliştirme Modeli Önerisi**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Yrd. Doç. Dr. Ayça TARHAN
Başkan ..... 

Yrd. Doç. Dr. Murat AYDOS
Danışman ..... 

Doç. Dr. Mehmet TEKEREK
Üye ..... 

Yrd. Doç. Dr. Abdulkadir YALDIR
Üye ..... 

Doç. Dr. Ahmet Burak CAN
Üye ..... 

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak onaylanmıştır.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Fen Bilimleri Enstitüsü Müdürü

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

☐ **Tezimin/Raporumun tamamı dünya çapında erişime açılabilir ve bir kısmı veya tamamının fotokopisi alınabilir.**
(Bu seçenekle teziniz arama motorlarında indekslenebilecek, daha sonra tezinizin erişim statüsünün değiştirilmesini talep etseniz ve kütüphane bu talebinizi yerine getirse bile, tezinin arama motorlarının önbelleklerinde kalmaya devam edebilecektir.)

☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç Kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
(Bu sürenin sonunda uzatma için başvuruda bulunmadığım taktirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı ve ya tamamının fotokopisi alınabilir)

☑ **Tezimin/Raporumun** 11.09.2018 **tarihine kadar erişime açılmasını istemiyorum, ancak kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisinin alınmasını onaylıyorum.**

☐ **Serbest Seçenek/Yazarın Seçimi**

11 / 09 / 2017

(İmza)

Güler Koç

Öğrencinin Adı Soyadı

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

21/08/2017

.....

GÜLER KOÇ

ÖZET

YAZILIM GELİŞTİRME MODELLERİNİN GÜVENLİK AÇISINDAN ANALİZİ VE BİR GÜVENLİ YAZILIM GELİŞTİRME MODELİ ÖNERİSİ

Güler KOÇ
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Danışmanı: Yrd. Doç. Dr. Murat AYDOS
Ağustos 2017, 70 sayfa

Yazılım geliştirme süreçleri, yazılımların planlanan zamanda ve planlanan maliyetle bitirilmesi için aşamaların hangi düzende uygulanacağına odaklanmaktadır. Ancak piyasaya sürülen yazılımlardaki zafiyetlerin yaygınlığı ve çokluğu, zaman ve maliyet kriterlerini sağlamakla beraber büyük ölçüde güvenliğe odaklanan daha sıkı bir yazılım geliştirme sürecine gereksinim olduğunu göstermektedir. Bununla beraber, güvenlik odaklı uygulamaların eklendiği geleneksel yazılım geliştirme süreçlerinin güvenliği sağlamada oldukça etkili olduğunu ve çevik geliştirme süreçlerinin geleneksel yazılım geliştirme süreçlerine göre zaman ve maliyet kriterlerini sağlama konusunda daha başarılı olduğunu gösteren bazı çalışmalar mevcuttur. Yapılan bu tespitten ve yukarıda bahsedilen ihtiyaçtan yola çıkarak bu çalışmada, çevik yöntemlerle güvenli yazılım geliştirmek için güvenlik uygulamalarının Scrum çerçevesi içinde uygulanmasını amaçlayan bir güvenli yazılım geliştirme modeli (Trustworthy Scrum) önerisi yapılmıştır.

Yazılım güvenliği ile ilgili literatür çalışmalarının ve güvenli yazılım geliştirme modellerinin incelenmesi sonucu, güvenliği hedefleyen uygulamalar belirlenmiştir. Güvenlik uygulamaları Scrum çerçevesine eklenirken, geleneksel metotlardaki güvenlik yaklaşımı ile çevik yazılım geliştirme ilkelerinin çakıştığı kısımlar belirlenmeye çalışılmıştır. Hem güvenlik ilkeleri hem de çevik yöntem ilkeleri göz önünde

bulundurularak, bu güvenlik uygulamalarının Scrum çerçevesi içinde uygulanmasını amaçlayan bir güvenli yazılım geliştirme modeli önerisi yapılmıştır. Önerilen modelin uygulanabilirliğinin test edilmesi amacıyla sahada çalışan yazılım geliştiricilerin görüşlerine başvurulmuş ve uygulanabilirliği konusunda olumlu sonuçlar ortaya çıkmıştır. Bu şekilde günümüz yazılımlarında oldukça ihtiyaç duyulan güvenlik gereksiniminin Scrum ile nasıl sağlanacağına dair bir çalışma olarak literatüre katkı sağlanmıştır.

Anahtar Kelimeler: Güvenli yazılım geliştirme, çevik yöntemler, scrum.

ABSTRACT

ANALYSIS OF SOFTWARE DEVELOPMENT MODELS IN TERMS OF SECURITY AND A MODEL PROPOSAL FOR SECURE SOFTWARE DEVELOPMENT

Güler KOÇ
Master of Science, Department of Computer Engineering
Supervisor: Asst. Prof. Dr. Murat AYDOS
August 2017, 70 pages

Software development process models focus on ordering and combination of phases to develop the intended software product within time and cost estimates. However, commonness of software vulnerabilities in the fielded systems shows that there is a need for more stringent software development process that focuses on improved security demands. Meanwhile, there are some reports that demonstrate the efficiency of existing security enhanced conventional processes and success of agile projects over conventional waterfall projects. Based on this finding and the demand for secure software, we propose a security enhanced Scrum model (Trustworthy Scrum) by taking advantages of both security activities and Scrum framework which has fast adaptation and iterative cycle. While enhancing Scrum with security activities, we try to retain agile and security disciplines by considering that conventional security approach conflicts with agile methodologies. It is shown through statistical test that the proposed model increases the applicability of security activities with agile methods.

Keywords: Secure software development, agile methodologies, scrum.

TEŞEKKÜR

Bilgi ve deneyimleri ile katkıda bulunan, tez konusunun belirlenmesini sağlayan, tez metninin yazılmasına ve tez çalışmasına yardımcı olan danışman hocam Sayın Yrd. Doç. Dr. Murat Aydos'a, tez metnini inceleyerek biçim ve içerik bakımından son halini almasına yardımcı olan Sayın Yrd. Doç. Dr. Ayça Tarhan'a, Sayın Doç. Dr. Mehmet Tekerek'e, Sayın Yrd. Doç. Dr. Abdulkadir Yaldır'a, Sayın Yrd. Doç. Dr. Fuat Akal'a ve Sayın Doç. Dr. Ahmet Burak Can'a teşekkür ederim.

İÇİNDEKİLER

Sayfa

ÇİZELGELER

Sayfa

ŞEKİLLER

Sayfa

SİMGELER VE KISALTMALAR

CERT	Computer Emergency Response Team
CERT/CC	Coordination Center Computer Emergency Response Team
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
DoD	U.S. Department of Defense
FAA	Federal Aviation Administration
FAA-iCMM	FAA integrated Capability Maturity Model
ISO	International Organization for Standardization
KLOC	1000 Kaynak Kod Satır Sayısı
LOC	Kaynak Kod Satır Sayısı
NIST	National Institute of Standards and Technology
PSP	Personal Software Process
ROI	Return on Investment
SAMM	Software Assurance Maturity Model
SDI	Strategic Defense Initiatives
SDL	Microsoft Security Development Lifecycle
SDLC	Software Development Lifecycle
SEI	Software Engineering Institute
SSE-CMM	Systems Security Engineering Capability Maturity Model
T-CMM	Trusted Capability Maturity Model
TSM	Trusted Software Methodology
TSP	Team Software Process
TSP-Secure	Team Software Process for Secure Software Development

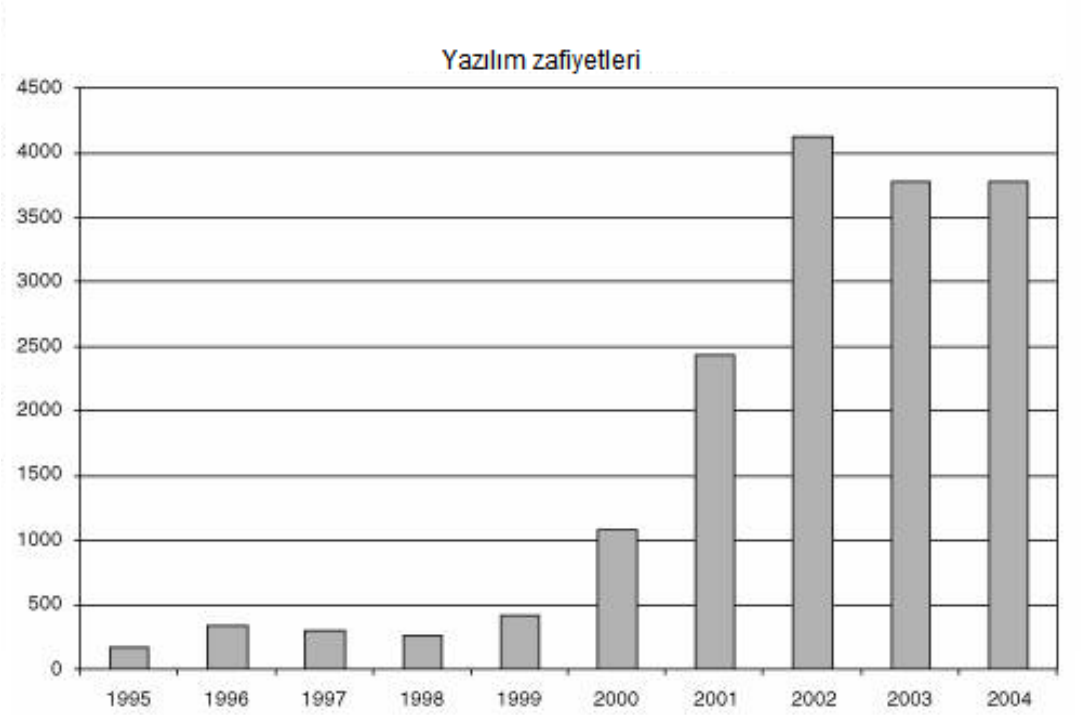
1. GİRİŞ

Geliştirilen bir yazılım projesinin planlamasından başlayarak teslimatına kadar geçirmiş olduğu bütün aşamalara ve bu aşamalardan oluşan döngüye, yazılım geliştirme yaşam döngüsü denir. Basitçe bir proje geliştirilirken projenin planlama, analiz, tasarım, gerçekleştirim ve bakım aşamaları yer almaktadır. Yaşam döngüsündeki bu aşamalara ilişkin işlevleri yerine getirmek amacıyla kullanılan yöntemler, belirtim yöntemleri olarak anılmaktadır. Süreç modelleri ise, yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlar, süreçlerin içsel ayrıntıları ya da süreçler arası ilişkilerle ilgilenmemektedir.

Geçmişten günümüze kadar verimli bir şekilde kaliteli ve olgun yazılım üretmek için farklı yazılım geliştirme süreç modelleri oluşturulmuştur. Bu modellerde daha çok planlanan zaman ve maliyet kriterlerini sağlamak için aşamaların düzen ya da sıralarının organize edilmesine odaklanılmıştır. Ancak yazılım projelerindeki başarısızlık oranlarına bakıldığında istenen sonucun elde edilemediği görülmektedir. "The Standish Group" adlı şirketin 1995 yılında yayınladığı "Chaos" raporuna [1] göre, yazılım projelerinin gerçek başarı oranı, %9-28 arası başarıyla çok düşük seviyelerdedir ve bunun temel nedeni de yönetim süreciyle ilgilidir. Bu durum, yazılım projelerini yönetmek için çevik yöntem olarak bilinen yeni yöntemlerin gerekliliğini ortaya çıkarmıştır [2]. Çevik yöntemler tekrarlı olarak ilerler, gereksinimlerin ve tasarımın kademeli olarak ortaya çıkmasına dayanır, şelale süreç modelinin yoğun yazılı dokümantasyonundan çok, doğrudan yüz yüze iletişime vurgu yapar [3]. Aynı şekilde "The Standish Group" tarafından 2015 yılında yayınlanan "Chaos" raporunda, 2011-2015 yılları arasında çevik ve şelale yöntemlerin uygulandığı projelerin başarı oranlarının karşılaştırıldığı çalışma, çevik yöntemlerin daha başarılı olduğunu göstermektedir [4].

Yukarıda bahsi geçen başarı tanımı "Chaos" raporunda [1], "projeyi başlangıçta belirlenmiş tüm özellik ve işlevlerini yerine getirerek planlanan zamanda planlanan bütçeyle bitirmek" olarak verilmiştir. Yani projenin güvenlik, hatasızlık ve sağlamlık boyutları hesaba katılmamıştır. Sahaya sürülen yazılımların güvenlik açısından incelendiği raporlar, başarılı görülen projelerin aslında birçok güvenlik sorunları ve programlama hatalarına sahip olduğu gerçeğini ortaya çıkarmaktadır. "Coordination Center Computer Emergency Response Team (CERT/CC)" tarafından yayınlanan raporlara göre çok yaygın ve gittikçe artan bir güvenlik açığı sorunu mevcuttur (Şekil 1.1). CERT/CC 2002 yılında 4,129 adet rapor edilmiş sistem açığı tanımlamıştır, bu sayı 2001 yılındaki sayının %70'i

kadar artış ve 2000 yılındaki sayının ise 3 katı kadar bir artış olduğunu göstermektedir [2], [5]. CERT/CC tarafından yapılan başka bir analize göre güvenlik açıklarının %90'dan fazlası, kodlama, tasarım ve gereksinim hataları gibi birçok hata tipinin dâhil olduğu bilinen yazılım hatalarından kaynaklanmaktadır [6]. Bununla beraber yazılımların karmaşıklığı ve genişleyebilirliği ihtiyaçları karşılamak için gittikçe artmakta ve sonuç olarak hata olma olasılığı da oldukça yükselmektedir. Bu da yazılım güvenliğine daha çok önem verilmesi gerektiğini göstermektedir. Ancak yazılım güvenliği ilk çalışmaların 2001'de başladığı çok yeni bir alandır.



Şekil 1.1. CERT/CC'ye rapor edilen yazılım zafiyetlerinin yıllara göre dağılımı

Yazılım güvenliği, yazılımı geliştirirken güvenli olacak şekilde inşa etmektir, yazılım geliştirme yaşam döngüsü boyunca uygulanması öngörülen tasarımı güvenli yapmak, güvenli kodlama ilkelerine uymak, yazılımın güvenli olduğundan emin olmak, geliştiricileri güvenlik konusunda eğitmek gibi birçok uygulamayı kapsar [7].

Buna karşılık yazılım güvenliği ile oldukça karıştırılan uygulama güvenliği ise yazılımı inşa ettikten sonra korumaya almak anlamına gelir. Uygulama güvenliği hataları daha çok hatalar baş gösterdikten sonra bulup çözmeye dayanırken, yazılım güvenliği bu hataların olabileceğini başta tespit edip bu hatalara karşı dirençli bir yazılım geliştirmeyi hedefler [7]. Bu bakımdan uygulama güvenliğine göre daha kapsamlı bir korumadır.

Yazılım endüstrisinde günümüzün oldukça gereksinim duyulan iyileştirilmiş, geliştirilmiş güvenlik talebini karşılamanın anahtarı, ölçülebilir şekilde geliştirilmiş güvenliği ortaya koyan tekrarlanabilir işlemlerin gerçekleştirildiği yazılım sürecidir [8]. Bu da yazılım sağlayıcıların, büyük ölçüde güvenliğe odaklanan daha sıkı bir yazılım geliştirme sürecine geçmelerini gerektirir. Böyle bir süreç, tasarım, kodlama ve dokümantasyon aşamalarında varolan güvenlik açıklarını minimize etmeyi, bu açıkları yazılım geliştirme yaşam döngüsünde mümkün olan en kısa sürede tespit edip ortadan kaldırmayı hedefler. Güvenliği içeren böyle bir sürece en çok, internetten alınan bilgileri işleyen, tehdit altında olan kritik sistemleri kontrol eden ve kişisel bilgileri işleyen kurum ve tüketici yazılımları ihtiyaç duymaktadır [8], [9].

Bazı yazılım sağlayıcılar, varolan geliştirme süreçlerine güvenlik uygulamaları ekleyerek, yazılım geliştirme yaşam döngüsü boyunca güvenlik açıklarını bulup minimize etmeyi amaçlamışlardır. Microsoft'un varolan standart yazılım geliştirme sürecine güvenlik uygulama ve elementlerini eklemesi sonucu oluşturduğu güvenli yazılım geliştirme süreci olan Microsoft Security Development Lifecycle (SDL) ile daha önceden 16 olan yıllık yayınlanan güvenlik bülten sayısını 3'e düşürdüğü görülmektedir [8]. Aynı şekilde Microsoft, SDL ile geliştirilen ve piyasaya sürülen yazılımlarda sistem açıklarının %50'den daha fazla oranda düştüğünü belirtmiştir [10]. Bununla beraber güvenli yazılım geliştirmek için yazılım mühendisliği konusunda disiplinli ve olgun ilkeler sunan Team Software Process (TSP) ile kalitenin %20 arttığı, öngörülen takvim planlamasından ve harcanan efordan %8 tasarruf sağlandığı rapor edilmiştir [11]. Bu raporlara benzer şekilde, güvenlik uygulamalarının güvenliği sağlama konusundaki başarısını kanıtlayan birçok çalışma mevcuttur.

Güvenli yazılım geliştirme modelleri, güvenlik uygulamalarının daha çok şelale süreç modeline eklenmesiyle oluşturulmuştur. Ancak geleneksel modellerdeki güvenlik yaklaşımının bazı noktalarda çevik ilkelerle ters düşmesinden dolayı çevik yöntemlere hitap eden çalışma sayısı oldukça azdır. Yapılan araştırmalarda, sistem tasarımının en baştan detaylı olarak belirlendiği yazılımlara güvenlik yaklaşımının çevik yöntemlerin kademeli olarak ilerleyen yinelemeli yapısına uymaması, test anlayışlarındaki farklılıklar ve güvenlik uygulamalarının dokümantasyona önem vermesi nedeniyle uyuşmazlıkların olduğu konusunda ortak bir görüş mevcuttur. Küçük iterasyonlar, geleneksel yazılım geliştirme modellerindeki güvenlik uygulamaları ile uyuşmamaktadır [3] ve küçük iterasyonlar gerekli testleri yapmak için yeterli değildir [12], [13]. Çevik modellerin,

güvenlik olaylarını gözardı etmesinin nedeni, güvenlik konusunda farkındalığın olmaması [12] veya güvenliğin yazılım geliştirmeyi aksattığı ve yavaşlattığı şeklindeki yanlış kanı da [14] olabilir.

Yukarıda bahsedilen ihtiyaçtan hareketle bu çalışmada, güvenli yazılım geliştirmek için hem çevik yöntemlerin hızlı, değişime ayak uyduran ve aşamalı ilerleyen yapısından yararlanılarak, hem de yazılımda güvenliği sağlamak için gerekli olan ve büyük şirketler tarafından uygulanarak güvenliği sağlama konusundaki başarısı kanıtlanmış olan güvenlik uygulamaları kullanılarak Scrum ile güvenli yazılım geliştirme modeli önerilmektedir.

Bu tez kapsamında yapılan çalışmalar aşağıda maddeler halinde verilmiştir:

1. Piyasaya sürülen yazılımlardaki güvenlik açıklarına dikkat çekerek, problemin neden kaynaklandığı üzerine araştırma yapılmıştır ve yazılım geliştirme süreçlerinde özellikle güvenliği amaçlayan uygulamaların olmayışına vurgu yapılmıştır.
2. Yetenek ve yazılım olgunluk modelleri ve güvenli yazılım geliştirme modelleri güvenlik açısından irdelenmiştir. Yazılım güvenliği ile ilgili literatür çalışmalarının ve güvenli yazılım geliştirme modellerinin incelenmesi sonucu, güvenliği hedefleyen uygulamalar belirlenmiştir.
3. Diğer taraftan çevik yöntemlerin, geleneksel yöntemlere göre zaman ve maliyet kriterlerini sağlama konusundaki başarısı göz önüne alınarak, güvenliğin çevik yöntemlere nasıl entegre edileceğine dair çalışmalar incelenmiştir. Güvenlik ile çevikliğin çakıştığı kısımlar belirlenmeye çalışılmıştır.
4. Hem güvenlik ilkeleri hem de çevik yöntem ilkeleri göz önünde bulundurularak, Scrum için güvenli yazılım geliştirme modeli önerisi yapılmıştır.
5. Önerilen modelin uygulanabilirliğinin test edilmesi amacıyla sahada çalışan yazılım geliştiricilerin görüşlerine başvurularak güvenlik uygulamalarının çevik yazılım geliştirme ile çakışan ve uyumlu tarafları belirlenmeye çalışılmıştır.

Bu tezin izleyen ikinci bölümünde, yazılım güvenliğinin tanımı, yazılım için referans niteliğindeki olgunluk modellerinin güvenlik açısından değerlendirmesi, varolan güvenli yazılım geliştirme modelleri (güvenliğin geleneksel yazılım geliştirme yöntemlerine eklenmesiyle oluşan modeller) ve çevik yöntemler hakkında ön bilgi verilmiştir. Üçüncü bölümde, literatürde yer alan, çevik yöntemlere güvenlik eklenmesiyle ilgili çalışmalardan bahsedilmiştir. Dördüncü bölümde, yazılım güvenliği için gerekli uygulamaların

ayrıntıları, neden gerektiği ve çevik yöntemlere uyarlanabilirliği alt başlıklar halinde verilmiştir. Beşinci bölümde, önerilen model (Trustworthy Scrum) detaylı olarak anlatılmıştır. Altıncı bölümde metodoloji ve deneyin kurulumu, yedinci bölümde deneyden elde edilen bulgular verilmiş ve son bölümde, çalışmanın sonuçları ve sonraki aşamalarda yapılabilecek olan uygulamalardan bahsedilmiştir.

2. ÖN BİLGİ

Bu bölümde sırasıyla, güvenli yazılım geliştirme, olgunluk modellerinin güvenlik açısından değerlendirmesi, güvenli yazılım geliştirme modelleri ve çevik yöntemlerle ilgili bilgilere yer verilmiştir.

2.1 Güvenli Yazılım Geliştirme

Yazılım güvenliği, yazılımı geliştirirken güvenli olacak şekilde geliştirmektir, yazılım geliştirme yaşam döngüsü boyunca uygulanmayı öngören uygulamalar içermektedir [7]. Aşağıda yazılım güvenliğiyle ilgili iki önemli terim olan yazılım ve güvenlik güvencesi tanımlanmıştır:

Yazılım güvencesi: Yazılım güvencesi, yazılımın sistem açıklarından ne kadar arınmış olduğunun ve yazılımın istenilen biçimde ne kadar işlediğinin bir seviyesidir [15].

Güvenlik güvencesi: SSE-CMM, güvenlik güvencesini, ürünün güvenlik ihtiyaçlarının karşılandığının güvencesini oluşturan süreç olarak tanımlamaktadır. Genel olarak, güvenlikle ilgili özellikler ve fonksiyonlarda güven sağlayacak faaliyetler, yöntemler ve prosedürlerdir. Yazılım geliştirme yaşam döngüsünün gereksinim analizi, tasarım, gerçekleştirim, test, teslim ve bakım aşamaları için faaliyetler içerir.

NASA'ya göre minimum güvenlik güvencesi programı aşağıdaki faaliyetleri kapsamalıdır [16]:

1. Güvenlik risk analizi yapılmıştır.
2. Geliştirilmekte ve/veya bakılmakta olan yazılım ve veri için güvenlik gereksinimleri oluşturulmuştur.
3. Geliştirme ve/veya bakım aşaması için güvenlik gereksinimleri oluşturulmuştur.
4. Tüm yazılım gözden geçirme ve denetimleri güvenlik gereksinimlerinin değerlendirmesini içermelidir.
5. Konfigürasyon yönetimi ve düzeltici faaliyet süreçleri varolan yazılım için güvenliği sağlamalı ve değişim değerlendirme süreçleri güvenlik ihlallerini önlemelidir.
6. Yazılım ve veri için fiziksel güvenlik uygun olmalıdır.

Varolan süreç modelleri ve standartlar güvenli yazılım geliştirme için aşağıdaki 4 yazılım geliştirme yaşam döngüsü faaliyet alanını tanımlar [17].

1. **Güvenlik Mühendisliği Faaliyetleri:** Güvenli bir çözüm planlayıp üretmek için gerekli faaliyetleri içerir. Örnek olarak, güvenlik gereksinimlerinin belirlenmesi, güvenlik için tasarım ilkelerine dayalı güvenli tasarım yapılması, statik analiz araçlarının kullanımı, güvenlik inceleme ve denetlemeleri ve güvenlik testi verilebilir.
2. **Güvenlik Güvencesi Faaliyetleri:** Doğrulama ("verification"), geçerleme ("validation"), uzman değerlendirmesi, ürün/ara ürün gözden geçirme ve değerlendirmeleri içeren güvence faaliyetlerini içerir.
3. **Güvenlik Organizasyonel ve Proje Yönetimi Faaliyetleri:** Organizasyonel faaliyetler, organizasyonel ilkeler, üst yöntem desteği ve denetimi, organizasyonel rollerin oluşturulması ve diğer güvenliği destekleyen organizasyonel aktiviteleri içerir. Proje yönetimi faaliyetleri, güvenlik mühendisliği, güvenlik güvencesi, risk tanımlama faaliyetlerinin planlanması, yönetimi ve takibini sağlamak için kaynak tahsisi ve kullanımının planlaması ve takibini içerir.
4. **Güvenlik Riski Tanımlaması ve Yönetimi Faaliyetleri:** Güvenlik risklerinin tanımlanmasının ve yönetiminin, güvenli SDLC içerisindeki en önemli faaliyetlerden biri olduğu konusunda yaygın bir görüş birliği vardır ve bu faaliyet gerçekten de sonraki faaliyetler için bir operatör konumundadır.

2.2 Yetenek Olgunluk Modelleri (CMMs)

Yetenek Olgunluk Modelleri, ilk olarak 1986 yılında ABD Savunma Bakanlığı'nın (US Depertment of Defense – DoD) isteği ve yardımlarıyla birlikte, Yazılım Mühendisliği Enstitüsü (Software Engineering Institute - SEI) bünyesinde Carnegie Melon Üniversitesi tarafından yürütülmeye başlanmıştır. CMM belirli mühendislik dalları için olgun uygulamalar sunan referans model niteliği taşır. CMM özel süreçler (yazılım mühendisliği, sistem mühendisliği, güvenlik mühendisliği) için hedefler ve anahtar öznitelikler sağlar. Ama bu tanımlamalar işi nasıl yerine getireceğine dair operasyonel yönlendirme sağlamaz. Yani, CMM işlemlerin ne olacağını değil bu işlemlerin niteliklerini tanımlar.

Tarih boyunca, CMM modelleri, daha iyi zaman yönetimi, daha iyi kalite yönetimi ve yazılımdaki hata oranının azaltılması gibi iş amaçlarını karşılamaya yönelik işlemlerin üzerinde durmuştur. CMM genelde organizasyonel ve proje yönetimi süreçleri ve güvence sürecine değinmiştir, ancak belirli bir biçimde güvenlik mühendisliği aktiviteleri veya güvenlik riski yönetimi gibi konulara adresleme yapmamıştır.

CMM tabanlı değerlendirmeler, organizasyonların ürün değerlendirme ve sistem sertifikasyonu süreçlerinin yerine kullanılacak bir model değildir, daha çok bu süreçlerdeki zayıflıkları düzeltme çalışmalarına odaklanmayı amaçlamaktadır [6].

CMM çeşitlerinden Tümleşik Yetenek Olgunluk Modeli (CMMI), Federal Havacılık Yönetim Tümleşik Yetenek Olgunluk Modeli (FAA-iCMM) ve Sistem Güvenlik Mühendisliği Yetenek Olgunluk Modeli (SSE-CMM) geniş kullanım alanına sahiptir. Sadece SSE-CMM özellikle güvenliği vurgulamak için geliştirilmiştir. Trusted CMM ise Güvenli Yazılım Metodolojisi (TSM) modelinden türemiştir.

2.2.1 Tümleşik Yetenek Olgunluk Modeli (CMMI)

Tümleşik Yetenek Olgunluk Modeli (Capability Maturity Model Integration - CMMI), bir süreç modeli olup, organizasyonların yazılım süreçlerinin olgunluğunu değerlendirme modelidir. Organizasyonlara, uzun vadede iş performansının olgunluğunu artırmasına yardım etmektedir.

CMMI'da süreç iyileştirme ve değerlendirme dört kategoride incelenir. Bunlar Proje Yönetimi, Süreç Yönetimi, Mühendislik ve Destektir. CMMI modellerinden CMMI-DEV (CMMI for Development) ürün ve hizmet geliştirmek için kapsamlı ilkeler sunar ve yazılım geliştirmede oldukça yaygın kullanılmaktadır. Çizelge 2.1'de görüldüğü gibi CMMI-DEV proje yönetimi, organizasyonel süreç iyileştirmesi ve eğitimi, kalite güvencesi ve ölçümü ve mühendislik pratiklerine değinir. Bu süreçlerde güvenlik güvencesi, güvenlik mühendisliği, güvenlik için organizasyonel ve proje faaliyetleri, güvenlik riski yönetimi gibi güvenlik alanlarına doğrudan vurgu yapılmamaktadır. CMMI içerisinde doğrudan güvenlikle ilgili bir süreç yoktur.

Çizelge 2.1. CMMI-DEV süreç alanları

Kategori	Süreç Alanları
Süreç Yönetimi	OPF Kurumsal Süreç Odaklanma OPD Kurumsal Süreç Tanımlama OT Kurumsal Eğitim OPP Kurumsal Süreç Performansı OID Kurumsal Yaratıcılık ve Yaygınlaştırma
Proje Yönetimi	PP Proje Planlama PMC Proje İzleme ve Kontrol SAM Tedarikçi Anlaşma Yönetimi IPM Bütünleşik Proje Yönetimi RSKM Risk Yönetimi IPT Bütünleşik Takım ISM Bütünleşik Tedrikçi Yönetimi QPM Sayısal Proje Yönetimi
Mühendislik	RD Gereksinim Geliştirme RM Gereksinim Yönetimi TS Teknik Çözüm PI Ürün Tümleştirme VER Doğrulama VAL Geçerleme
Destek	CM Konfigürasyon Yönetimi PPQA Süreç ve Ürün Kalite Güvencesi MA Ölçme ve Çözümleme DAR Karar Analizi ve Çözüm CAR Nedensel Analiz ve Çözüm

2.2.2 Federal Havacılık Yönetimi Tümleşik Yetenek Olgunluk Modeli

Federal Havacılık Yönetimi Tümleşik Yetenek Olgunluk Modeli (Federal Aviation Administration integrated Capability Maturity Model - FAA-iCMM) federal havacılık yönetiminde yaygın olarak kullanılır. FAA-iCMM modeli, dış kaynak kullanımı ve kaynak yönetimini de içeren büyük yazılım sistemlerinde kullanılabilecek pratiklerden oluşan bir model sunar.

FAA-iCMM, Çizelge 2.2'de görüldüğü üzere, 3 kategori ve 23 süreç alanına ayrılmıştır. FAA-iCMM proje yönetimi, risk yönetimi, tedarikçi yönetimi, bilgi yönetimi, konfigürasyon yönetimi, tasarım ve test gibi güvenli yazılım geliştirme yaşam döngüsünü bütünleyici süreçlere değinir. FAA-iCMM de CMMI gibi doğrudan güvenliği vurgulamayan genel uygulamalar seti sunar.

Çizelge 2.2. FAA-iCMM süreç alanları

Kategori	Süreç Alanları
Yönetim Süreçleri	Bütünleşik Kurum Yönetimi Proje Yönetimi Risk Yönetimi Tedarikçi Anlaşma Yönetimi Bütünleşik Takım
Yaşam Döngüsü Süreçleri	İhtiyaçlar Gereksinimler Tasarım Gerçekleştirim Entegrasyon Kurulum ve Dağıtım İşletme ve Destek Değerlendirme
Destek Süreçler	Dış Kaynak Kullanımı Seçenek Analizi Ölçme ve Çözümleme Kalite Güvencesi ve Yönetimi Konfigürasyon Yönetimi Bilgi Yönetimi Süreç Tanımlama Süreç İyileştirme Eğitim Yenilik

Güvenlikteki açıkları kapatmak için, Federal Havacılık Yönetimi (Federal Aviation Administration - FAA) ve ABD Savunma Bakanlığı (U.S. Depertment of Defense – DoD)

içindeki bazı organizasyonlar, FAA-iCMM ile uyumlu kullanılacak şekilde en iyi güvenlik uygulamalarının belirlenmesi üzerine çalışmalara destek vermektedir. FAA-iCMM için önerilen bu Emniyet ve Güvenlik (Safety and Security) uzantısı aşağıdaki 4 hedefi kapsar:

Hedef 1: Emniyet ve güvenlik için altyapı tanımlanmalı ve yönetilmelidir.

Hedef 2: Emniyet ve güvenlik riskleri tanımlanmalı ve yönetilmelidir.

Hedef 3: Emniyet ve güvenlik gereksinimleri yerine getirilmelidir.

Hedef 4: Faaliyetler ve ürünler, emniyet ve güvenlik gereksinimlerini ve hedeflerini başarmak için yönetilmelidir.

2.2.3 Güvenli CMM/Güvenli Yazılım Metodolojisi (T-CMM/TSM)

1990'larda Strategic Defense Initiatives (SDI) tarafından Güvenli Yazılım Geliştirme Metodolojisi adıyla geliştirilen model daha sonra Güvenli Yazılım Metodolojisi (Trusted Software Methodology - TSM) olarak adlandırıldı. Bu modelde düşük seviye istenmeden yapılan yazılım zafiyetlerine karşı direnç gösteren, yüksek seviye ise kötücül ataklara karşı koruyan işlemler içeren seviye olarak tanımlanmıştır. TSM daha sonra CMM ile birleştirilmiş ve Güvenli CMM (Trusted CMM - T-CMM) ortaya çıkmıştır [18]. T-CMM/TSM günümüzde yaygın olarak kullanılmamaktadır ancak güvenli yazılım geliştirme için kaynak olabilecek bilgiler içermektedir [17].

2.2.4 Sistem Güvenlik Mühendisliği Yetenek Olgunluk Modeli

Sistem Güvenlik Mühendisliği Yetenek Olgunluk Modeli (Systems Security Engineering Capability Maturity Model - SSE-CMM) organizasyonaların güvenlik mühendisliği yeteneğini geliştirip değerlendirmeleri için kullanabilecekleri bir modeldir. SSE-CMM, güvenlik uygulamalarını, genel kabul görmüş güvenlik mühendisliği ilkelerine göre değerlendirmeye yönelik kapsamlı bir çerçeve sunar. Bu model iki geniş alana ayrılmıştır: birincisi güvenlik mühendisliği, diğeri ise proje ve organizasyon süreçleridir (Çizelge 2.3).

SSE-CMM en son 2005 yılında revize edilmiştir. Bu model 2008 yılında ISO standartı haline gelmiştir. The International Systems Security Engineering Association (ISSEA) SSE-CMM modelini sürdürmektedir [17].

Çizelge 2.3. SSE-CMM süreç alanları

Proje ve Organizasyon Süreçleri	
Kaliteyi Sağla Konfigürasyonu Yönet Proje Riskini Yönet Teknik Eforu İzle ve Kontrol Et Teknik Eforu Planla Organizasyonun Sistem Mühendisliği Sürecini Tanımla Organizasyonun Sistem Mühendisliği Sürecini İyileştir Ürün Hattı Değerlendirmesini Yönet Sistem Mühendisliği Destek Ortamını Yönet Yetenek ve Bilginin Sürekliliğini Sağla Tedarikçilerle Koordinasyon Sağla	
Güvenlik Mühendisliği Süreç Alanları	
Mühendislik	Güvenlik Gereksinimlerini Belirle Güvenlik Girdileri Sağla Güvenlik Durumunu İzle Güvenlik Kontrollerini Yönet Güvenliği Sağla
Güvence	Güvenliği Doğrula ve Geçerle Güvenlik Argümanı Oluştur
Risk	Tehditleri Belirle Sistem Açıklarını Belirle Etkilerini Tanımla Güvenlik Riskini Belirle

2.3 Yazılım Garanti Olgunluk Modeli (SAMM)

Yazılım Garanti Olgunluk Modeli (Software Assurance Maturity Model - SAMM"), organizasyonlara güvenli yazılım geliştirmek amacıyla strateji belirlemeleri konusunda yardımcı olmak için geliştirilmiş bir olgunluk modelidir. SAMM, yazılım geliştirmedeki 4 temel fonksiyona ayrılmaktadır. Bu ana başlıklar aslında normal yazılım geliştirme döngüsünün temel adımlarına karşılık gelmektedir. Her bir ana başlık altında üçer güvenlik

uygulaması yer almaktadır. Bu güvenlik uygulamaları Çizelge 2.4'deki gibi kategorize edilmiştir.

Yönetim (Governance): Organizasyonda uygulanacak yazılım güvenliği programı, güvenlik çalışmalarının performansını ölçme yöntemleri, belirlenmiş standartlara uyum sağlanması ve çalışanların yazılım güvenliği konusunda eğitilmesi gibi uygulamaları kapsamaktadır.

Yapım (Construction): Güvenli yazılım geliştirmek için yazılım gereksinimi ve tasarımı aşamalarında gerçekleştirilmesi gereken güvenlik eylemlerine değinmektedir. Yazılımın karşılaşacağı tehditlerin değerlendirilmesi, güvenlik ihtiyaçlarının belirlenmesi ve güvenli mimarinin oluşturulması gibi güvenlik uygulamalarını içerir.

Doğrulama (Verification): 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 uygulamalarıdır.

Uygulama (Deployment): Yazılımın piyasaya sürülmesi ve desteğinin verilmesi faaliyetlerini kapsamaktadır. Sistem açığı yönetimi, ortam sıkılaştırması ve operasyonel bilgi aktarımı bu aşamadaki güvenlik uygulamalarıdır.

Çizelge 2.4. SAMM yapısı

Yönetim	**Yapım**	**Doğrulama**	**Uygulama**
Strateji ve metrikler	Tehdit değerlendirmesi	Tasarım gözden geçirme	Açıklık yönetimi
Politika ve uyum	Güvenlik gereksinimleri	Kod gözden geçirme	Ortam sıkılaştırması
Eğitim ve rehberlik	Güvenli mimari	Güvenlik testi	Operasyonel bilgi aktarımı

2.4 Güvenli Yazılım Geliştirme Modelleri

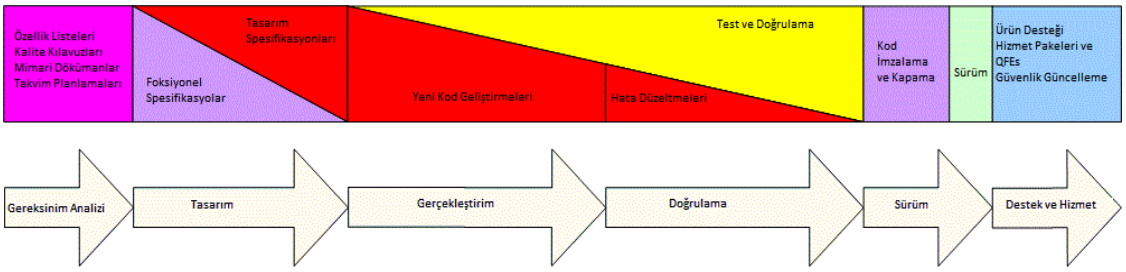
Bu bölümde geleneksel yazılım geliştirme süreç modellerine güvenliğin eklenmesiyle oluşturulan geliştirme modelleri verilecektir.

2.4.1 Microsoft Güvenli Yazılım Geliştirme Yaşam Döngüsü

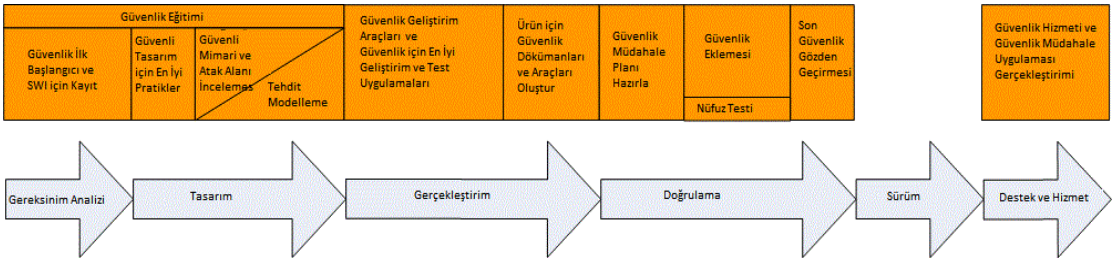
Microsoft Security Development Lifecycle (SDL), Microsoft'un güvenlik ataklarına karşı koyabilen bir yazılım geliştirilmesi için hayata geçirdiği yazılım sürecidir [8]. Bu süreç, güvenlik odaklı uygulamaların, Microsoft'un varolan yazılım geliştirme sürecinin her bir aşamasına eklenmesini içerir. Bu uygulamalar, yazılım tasarımı aşamasında tehdit modellerinin oluşturulmasını, gerçekleştirim aşamasında statik kaynak kod analizi yapan araçların kullanımını, kod gözden geçirme ve güvenlik testlerinin yapılmasını içerir.

Şekil 2.1'de sunulan Microsoft'un standart yazılım geliştirme modeli [8], şelale (waterfall) modeli gibi görünse de aslında süreç sarmal (spiral) modeldir. Gereksinimler ve tasarım, değişen piyasa ihtiyaçlarına göre sürekli olarak ziyaret edilir. Aynı zamanda geliştirme süreci, her aşamada çalışabilen bir kod olması gerektiğini vurgular. Bu yüzden süreçteki önemli dönüm noktaları, test edilebilir, kullanılabilir ve teslim edilebilir parçalara ayrılır.

Şekil 2.2'de ise Microsoft'un standart varolan yazılım geliştirme sürecine güvenlik uygulama ve elementlerinin eklenmesi sonucu ortaya çıkan model sunulmaktadır [8].



Şekil 2.1. Standart Microsoft geliştirme süreci



Şekil 2.2. SDL iyileştirmeleri eklenmiş Microsoft geliştirme süreci

SDL aşağıdaki aşamaları içerir:

a. **Eğitim ve Farkındalık:** Bazı organizasyonlar, merkezi güvenlik ekibi rolünün, danışman tarafından yürütüldüğü kanaatinde olsa da, SDL, yazılım güvenliğini iyileştirmeyi amaçlayan adımların, yazılım geliştirme sürecine entegre edilmesi gerektiğini ifade eder [8]. Bunun için tercih edilecek çözüm ise, güvenlik ekibini yazılım geliştirme organizasyonu bünyesinde oluşturmak, takımı oluşturup üyelerini eğitmek ve üyelere güvenlik konusunda farkındalık yaratmaktır.
b. **En İyi Tasarım Uygulamalarını Tanımlamak ve Takip Etmek:** Güvenlik mimarisi ve tasarım prensiplerini tanımlamak için, sisteme güvenlik açısından bakılarak yazılımın geniş kapsamlı yapısı tanımlanır, doğru işleyişi güvenlik için vazgeçilemez olan bileşenleri belirlenir. Yazılım saldırı yüzeyinin elemanları dokümante edilir ve saldırı yüzeyinin en aza indirgenmesi için tasarım teknikleri tanımlanır.
c. **Risk Analizi:** Tehdit modellemesi yürütülerek, organizasyon değerlerine zarar verecek veya zarar verme riski olan tehditler, önlem almak amacıyla önceden belirlenir. Daha sonra bileşen ekibi bu riski ortadan kaldırıcı önlemler alır.
d. **Güvenlik Dokümanları, Araçları ve Müşteriler için En iyi Uygulamaları Oluşturmak:** Hiçbir müşteri uygulama kullanıcısının güvenlik hataları yapmasını istemeyeceği için güvenlikle ilgili böyle bir doküman çok yararlı olacaktır.
e. **Güvenli Kodlama ve Test İlkeleri:** Kodlama ve test ilkelerine göre gerçekleştirim yapılır, statik analiz araçlarıyla kaynak kod taranır.
f. **Güvenlik Eklemesi (Security Push):** Güvenlik için kod gözden geçirme ve güvenlik testi çalışmalarını içerir.
g. **Son Güvenlik İncelemesi (Final Security Review – FSR):** FSR'nin amacı, "Güvenlik açısından bakıldığında, bu yazılım müşterilere sunmak için hazır mı?" sorusunu cevaplamaktır. Yani yazılımın piyasaya sürüldükten sonra saldırılara karşı koyabilirliğini sunmaktadır. Eğer FSR bir güvenlik açığı bulursa, uygun çözüm sadece hatayı düzeltmek değil, aynı zamanda yazılım geliştirme sürecinde yer alan daha önceki aşamaları da tekrar gerçekleştirip, olayın temelindeki sebebe (örneğin, eğitimi iyileştirmek, araçları geliştirmek) karşı gerekli işlemleri yapmaktır.
h. **Güvenlik Müdahale Uygulaması (Security Response Execution):** Güvenlik hataları, tipik hatalar gibi değildir, müşteriyi riske atacak acil durumlara neden

olabilirler. Bu yüzden bir an önce güvenlik hatasını yerinde gidermek için hazır bir plana sahip olmaya ihtiyaç vardır.

2.4.2 Güvenli Yazılım Geliştirmede Takımsal Yazılım Süreci

SEI tarafından geliştirilen TSP (Team Software Process for Secure Software Development), yazılım mühendisliği ilkelerini takımda veya bireysel olarak uygulamaya yönelik işlemler ve disiplinli yöntemler takımı sunar. TSP-Secure ise, TSP'den türeyen ve yazılımlarda güvenliğe daha çok ve doğrudan odaklanan bir geliştirme modelidir. TSP-Secure güvenli yazılım geliştirmeyi 3 yöntemle sağlar. Birincisi güvenliği sağlamak için planlama yapar. İkincisi TSP-Secure ürün geliştirme döngüsü boyunca kaliteyi yönetmeye yardım eder. Son olarak, TSP-Secure geliştiriciler için güvenlik farkındalığı eğitimleri sağlar.

Güvenlik Yöneticisi, gereksinim, tasarım, gerçekleştirim, gözden geçirme ve test aşamalarında güvenlik konusunun sağlanması için takıma rehberlik eder. Gerektiğinde dışarıdan bir güvenlik uzmanı ile çalışabilir.

TSP-Secure modelinin kalite yönetim stratejisi, yazılım geliştirme döngüsünde birden fazla hata yok etme noktasına sahip olmasıdır. Daha çok hata yok etme noktası demek, hatalar ortaya çıktıktan hemen sonra sorunu fark edip daha karmaşık bir hal almadan kolay bir şekilde çözmek ve hatanın ana nedenini saptamak demektir. Her hata yok etme aktivitesi, sistem açığına yol açabilecek hataların belli bir bölümünü yazılımdan uzaklaştıran bir filtre gibidir [11].

Yazılım geliştirme sürecinde, yazılımdaki hataları azaltmanın çok fazla maliyet getireceği üzerine yaygın bir görüş vardır. Ancak geliştirme aşamasında ne kadar az hatalı yazılım üretilirse, yazılımın onarımına da o kadar az zaman harcanır, sürümün gecikmesinden dolayı kaynaklanacak olan maliyet önlenmiş olur ve sonuçta toplam verimlilik artar. Örnek olarak, TSP projelerinde ortalama zamanlama hatası sadece %6, onarım için ortalama zamanlama ise sadece %4 olarak sonuçlanmış ve verimlilikte %78 artış sağlanmıştır [19]. Bu konuda yapılan bir başka çalışmada [20], zamanlama ve kalite arasındaki bağlantı tanımlanırken; yazılımda daha az hatanın daha az zamanlama hatası anlamına geldiği ifade edilmiştir.

TSP-Secure'de de güvenlik konusunda farkındalık ve eğitim, güvenlik risklerini ve güvenlik gereksinimlerini belirleme, güvenli tasarım, tasarım ve kod gözden geçirmeler,

statik analiz araçlarının kullanımı, birim testleri ve bulandırma testleri gibi güvenliğe odaklanan birçok uygulama vardır.

SEI tarafından yetkili TSP eğitmeni Noopur Davis, yazılım yaşam döngüsünün aşamalarında güvenlik için uygulanması gereken en iyi pratikleri Çizelge 2.5'deki gibi sunmuştur. Hangi süreç modeli olduğu önemli değildir, sarmal, artırımsal veya iteratif geliştirme için, bu pratikler ürün ilerledikçe tekrarlanacaktır [21].

Çizelge 2.5. Yazılım geliştirmede uygulanacak en iyi güvenlik uygulamaları

Gereksinim Analizi	**Tasarım**	**Gerçekleştirim**	**Test**
Güvenlik gereksinimleri	Tehdit modelleme	Kodlama standartları	Güvenlik test planları
Değerlerin tanımlanması	Güvenli tasarım ilkeleri	Kod gözden geçirme	Black-box testi
Kullanım senaryoları	Güvenlik özellikleri tasarımı	Statik analiz araçları	White-box testi
Kötüye kullanım senaryoları	Tasarım gözden geçirme	Dinamik analiz araçları	Hata testi gözden geçirmesi

2.4.3 Yapısal Doğruluk

Yapısal doğruluk (Correctness by Construction), Praxis High Integrity Systems tarafından yüksek bütünlük gerektiren yazılımları geliştirmek için oluşturulan bir metodolojidir [22]. Bu metodoloji, emniyet ve güvenlik bakımından kritik önem taşıyan sistemleri büyük bir başarıyla geliştirmek için kullanılır [23].

Correctness by Construction modeli, yazılımın güvenlik ve emniyet özelliklerini belirtmek için neredeyse her zaman resmi bir metot kullanır. Bu model aşağıdaki ilkeleri benimser:

1. Değişen gereksinimlere ayak uydurma,
2. Hem hatasız yazılım hem de doğrulama (verification) için test,
3. Test etmeden önce hataları giderme,
4. Doğrulaması kolay olacak şekilde yazılım geliştirme,
5. Aşamalı geliştirme,

6. Kullanıcı kılavuzu, iş süreçleri, tasarım dokümanları, yorumlarla desteklenmiş kaynak kod ve test durumları oluşturma.

Correctness by Construction, resmi yöntemleri geliştirme aktivitelerine dâhil eden sayılı güvenli yazılım geliştirme süreçlerinden bir tanesidir [24].

Correctness by Construction ile geliştirilen yazılımlarda daha az resmi yaklaşımlarla geliştirilen sistemlere göre hata oranı azalmıştır. Her KLOC için 0.04 hata oranıyla, endüstri ortalamasına göre çok daha iyi başarı elde etmiştir [25].

2.5 Çevik Yöntemler

Çevik modelleme, yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve dokümantasyonunu yapmaya yönelik pratiğe dayalı yöntemlere denir. Aşırı kuralcı klasik yazılım süreç modellerine karşın bu model yazılım geliştirme sürecini hızlandırmak, daha etkin kullanmak ve gerektiğinde dokümante etmek amacıyla ortaya çıkmıştır.

Çevik yazılım geliştirmenin 12 ilkesi [26] aşağıda verilmiştir:

1. En önemli önceliğimiz değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir.
2. Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır.
3. Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
4. İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdır.
5. Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
6. Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüzyüze iletişimdir.
7. Çalışan yazılım ilerlemenin birincil ölçüsüdür.
8. Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
9. Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
10. Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır.

11. En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
12. Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

Agile metodu, Sınırsal Programlama (Extreme Programming - XP), Çevik Birleştirilmiş Süreç (Agile Unified Process), Scrum, Test Güdümlü Geliştirme (Test-Driven Development), Özellik Güdümlü Geliştirme (Feature-Driven Programming) gibi aralarında farklılaşan metodolojilere sahiptir. Bu metodolojiler arasında birçok farklılıklar olsa da temelde bazı ortak ilkelere dayanır. Bu ilkeler:

1. Erken ve devamlı teslim edilebilir yazılım,
2. Değişen gereksinimlere hızlı adaptasyon,
3. Kısa geliştirme iterasyonları,
4. Ön aşama olarak minimal tasarım,
5. Gelişmekte olan tasarım ve mimari,
6. Teknik ve iyi tasarım konusunda özen,
7. Doğrudan iletişim ve minimal ya da hiç dokümantasyon (dokümantasyon kaynak kodda yapılır).

Bu uygulamalardan bazıları güvenli yazılım geliştirme süreçleri ile çelişmektedir. Örneğin, projenin başlangıç aşamasında, tehdit modelleme gibi güvenlik risklerine değinen güvenli tasarım ilkelerine dayalı detaylı tasarım çoğu güvenli yazılım geliştirme süreçlerinin gerekli bir parçasıdır. Bu yöntem çevik metotların, gelişmekte olan gereksinim ve tasarım ilkeleriyle uyuşmamaktadır.

Çevik yöntemlerlerden Scrum'la ilgili temel bilgiler aşağıda verilmiştir:

Scrum

Scrum, Scrum kılavuzunda [27] aşağıdaki gibi tanımlanmaktadır:

Scrum, 1990'ların başından beri karmaşık ürün geliştirme sürecini yönetmek için kullanılan bir süreç çerçevesidir. Scrum çerçevesi, Scrum Takımları ve takımlarla ilgili rolleri, etkinlikleri, eserleri ve kuralları kapsar. Scrumın temelinde deneysel süreç kontrol teorisi yer alır. Scrum, öngörülebilirliği en iyi seviyeye çıkarmak ve riski kontrol etmek için iterasyonlu ve artımlı bir yaklaşım kullanır.

Scrum'ı tanımak için Scrum kılavuzundan [27] alınan bazı önemli tanımlar aşağıda verilmiştir:

Scrum Takımı

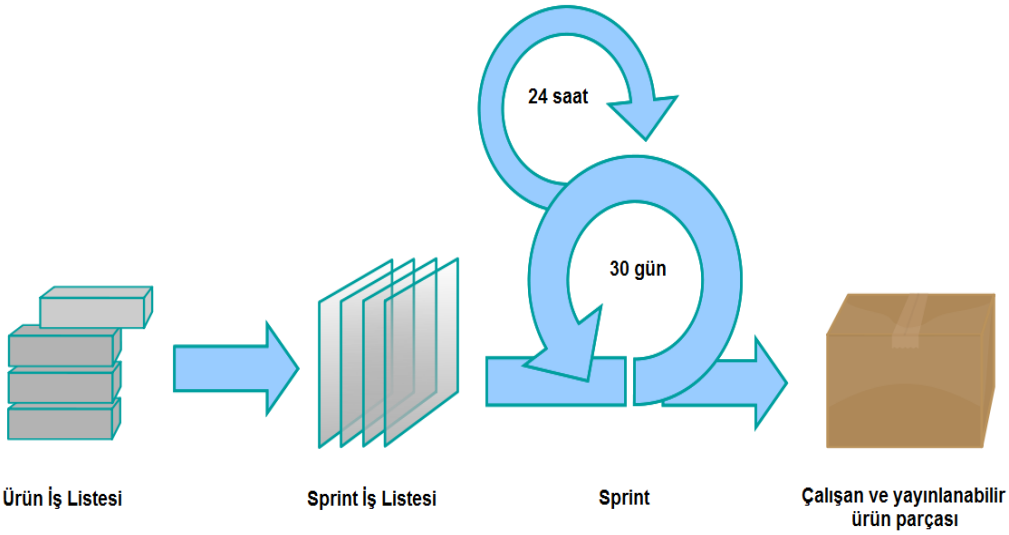
Scrum Takımı, bir Ürün Sahibi, Geliştirme Takımı ve bir de Scrum Masterdan oluşur. **Geliştirme Takımı**, her Sprint sonunda "Bitti" tanımına uyan ve yayınlanabilir ürün parçası teslim etmekle, **Ürün Sahibi**, Geliştirme Takımının işini ve ürünün değerini en üst seviyeye çıkarmakla, **Scrum Master,** Scrumın anlaşılmasını ve uygulanmasını temin etmekle sorumludur.

Scrum Etkinlikleri

Bir ay veya daha az zaman sınırı olan, içerisinde "Bitti" durumunda, kullanılabilir ve potansiyel olarak yayınlanabilir bir Ürün Parçasının oluşturulduğu **Sprint**, Scrumın kalbidir. Sprintte yapılacak iş **Sprint Planlama** toplantısında planlanır. **Sprint Değerlendirme**, her bir Sprintin sonunda Ürün Parçasını görüp kontrol etmek ve gerekiyorsa Ürün İş Listesini uyarlamak için düzenlenir. **Sprint Retrospektifi**, Scrum Takımının kendini gözlemlemesi ve sıradaki Sprintte yapacağı iyileştirmelere ilişkin bir plan oluşturması için bir fırsattır.

Scrum Eserleri

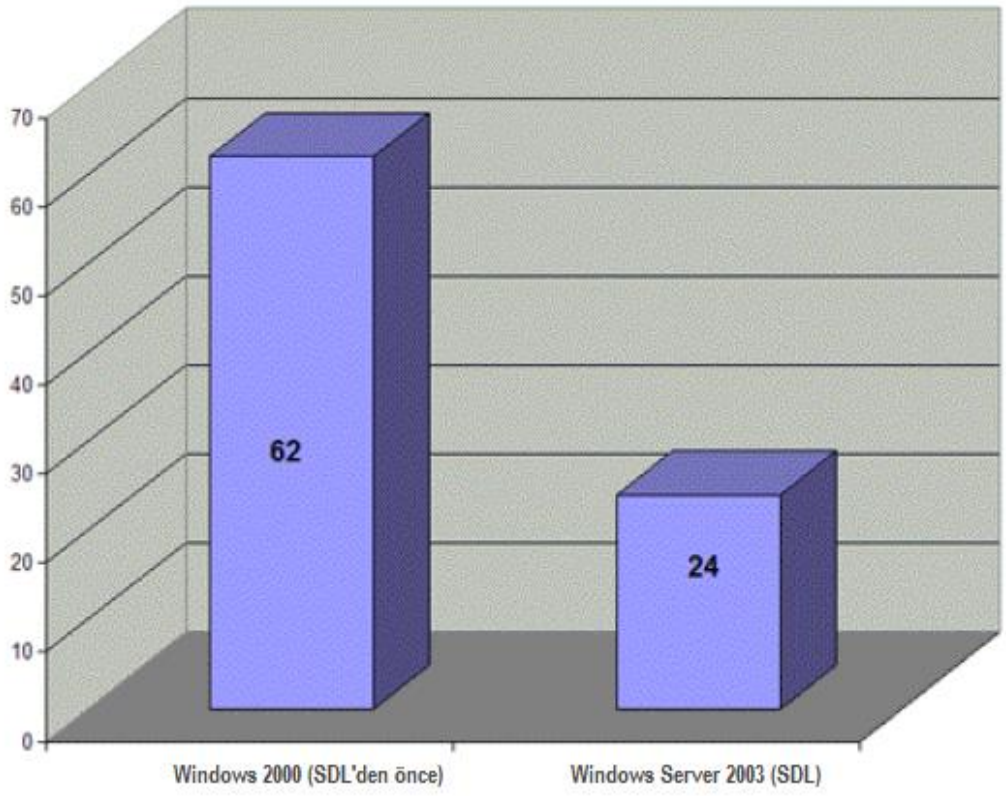
Ürün İş Listesi, üründe ihtiyaç duyulan her şeyin sıralandığı bir liste, yani gereksinimler kaynağıdır. **Sprint İş Listesi**, Sprint için seçilen Ürün İş Listesi kalemlerini ve sprint hedefine ulaşma planını içerir. **Ürün Parçası**, bir Sprint boyunca tamamlanan Ürün İş Listesi kalemlerinin ve tüm geçmiş Sprintlerin Ürün Parçalarının değerlerinin toplamıdır.



Şekil 2.3. Scrum süreci

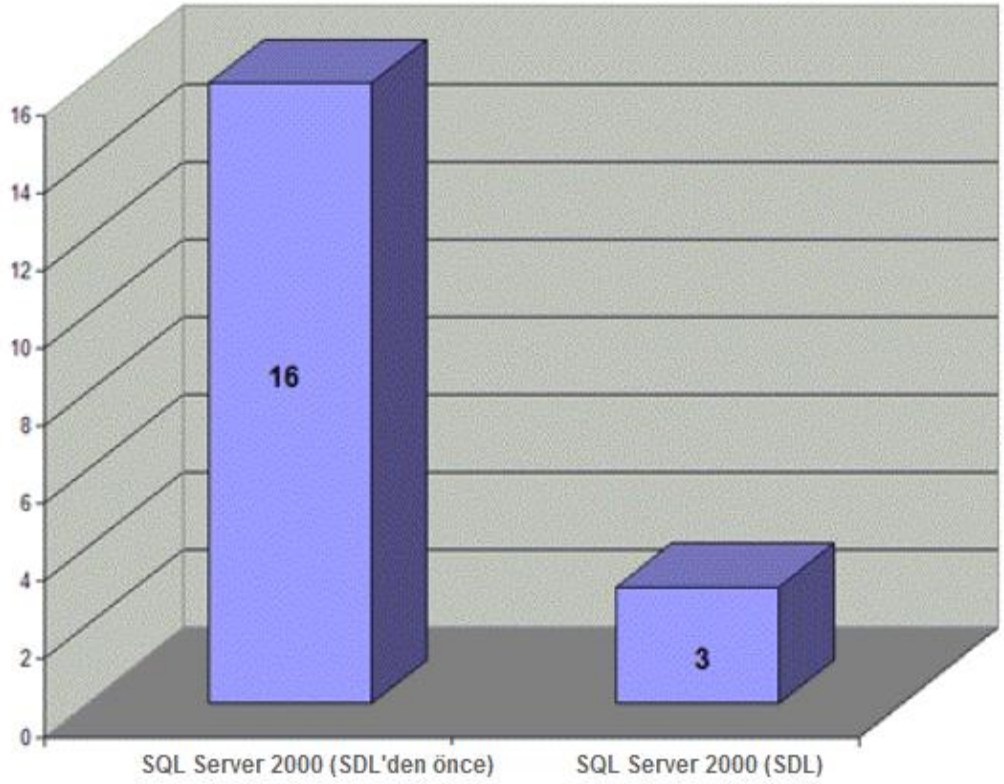
2.6 Güvenli Yazılım Geliştirme Modelinin Firmalardaki Sonuçları

Windows Server 2003, Microsoft'un güvenli yazılım geliştirme modeli olan SDL (SDL modelinin tamamı değil ancak geniş bir parçası) ile geliştirdiği ilk işletim sistemi sürümüdür [8]. Şekil 2.4 [8], Microsoft'un iki işletim sistemi olan Windows 2000 ve Windows Server 2003 sürüldükten sonraki bir yıl içerisinde ortaya çıkan güvenlik bültenleri sayısını göstermektedir. Windows Server 2003, tamamen olmasa da SDL süreçlerinin oldukça büyük bir kısmıyla geliştirilmiş, Windows 2000 SDL süreçleri ile geliştirilmemiştir.

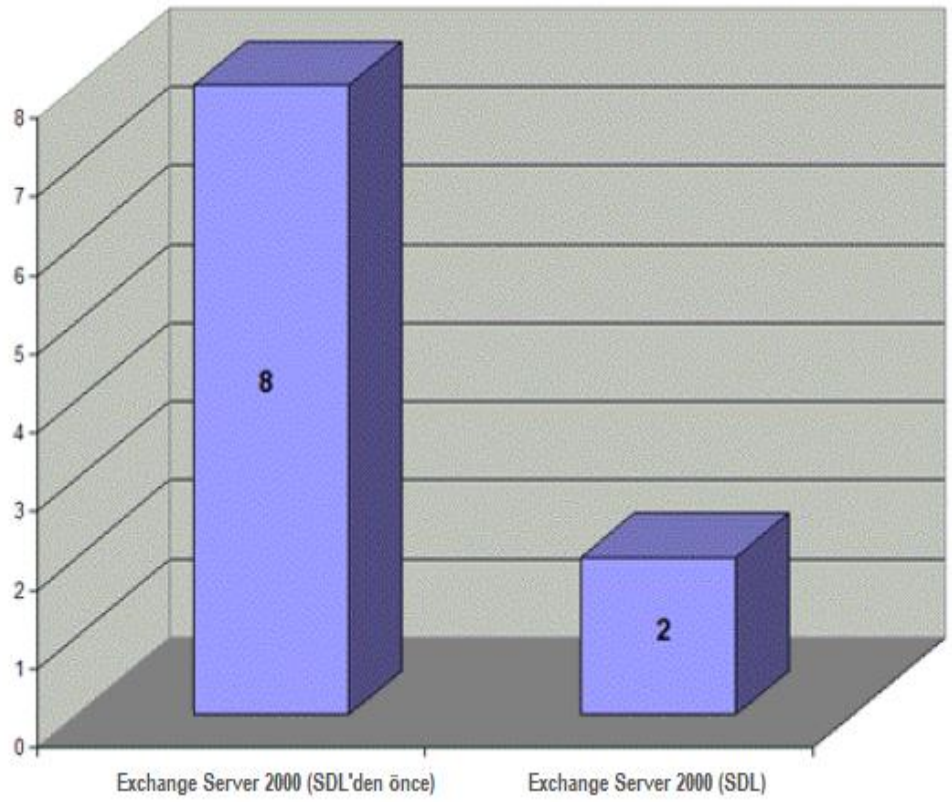


Şekil 2.4. Windows işletim sistemi için SDL'den önce ve sonraki kritik ve önemli seviyeli güvenlik bültenleri

SDL süreçleri uygulanan diğer Microsoft ürünlerine SQL Server 2000 ve Exchange 2000 Server yazılımları örnek verilebilir. SQL Server ve Exchange Server takımları, yazılımları piyasaya sürmeden önce tehdit modelleme, kod gözden geçirme ve güvenlik testi içeren güvenlik eklemesi (security push) faaliyetini uygulamıştır [8]. Şekil 2.5 ve 2.6 [8], sırasıyla SQL Server 2000 ve Exchange 2000 Server için güvenlik eklemesi yapılmadan önceki ve yapıldıktan sonraki eşit zaman aralıklarında (SQL Server 2000 için 24 aylık periyot ve Exchange 2000 Server için 18 aylık periyot) yayınlanan güvenlik bülteni sayılarını karşılaştırmaktadır.



Şekil 2.5. SQL Server 2000 için SDL'den önce ve sonraki güvenlik bültenleri



Şekil 2.6. Exchange Server 2000 için SDL'den önce ve sonraki güvenlik bültenleri

İyi yapılandırılmış yazılım mühendisliği ilkelerini desteklemek için tasarlanmış operasyonel bir süreç olan TSP'yi kullanmak için, yazılımcılar önce Personal Software Process (PSP) denilen kişisel yazılım süreci konusunda eğitim alırlar [28]. TSP kalite iyileştirmesi kapsamında 298 yazılım geliştiriciden alınan PSP eğitim verisi ve 4 şirketteki 18 projeden alınan TSP verisinden çıkan sonuçlara göre TSP projelerinde kalite değeri diğer projelerdekine göre 20 kat yükselmiştir ve planlanan efor ve takvim planından %8 tasarruf sağlanmıştır [28]. Bu verilerden çıkan TSP proje sonuçları [28] Çizelge 2.6 ve 2.7'de ayrıntılı olarak verilmiştir.

Çizelge 2.6. TSP proje sonuçları

		Planlanan	**Gerçekleşen**
Boyut		110 LOC	89,995 LOC
Efor		16 saat	14,711 saat
Zaman		77 hafta	71 hafta
Ürün kalitesi (Her aşamada giderilen hata sayısı/KLOC)	• **Entegrasyon**	1.0	0.2
	• **Sistem testi**	0.1	0.4
	• **Saha denemesi**	0.0	0.02

Çizelge 2.7. TSP proje sonuçları

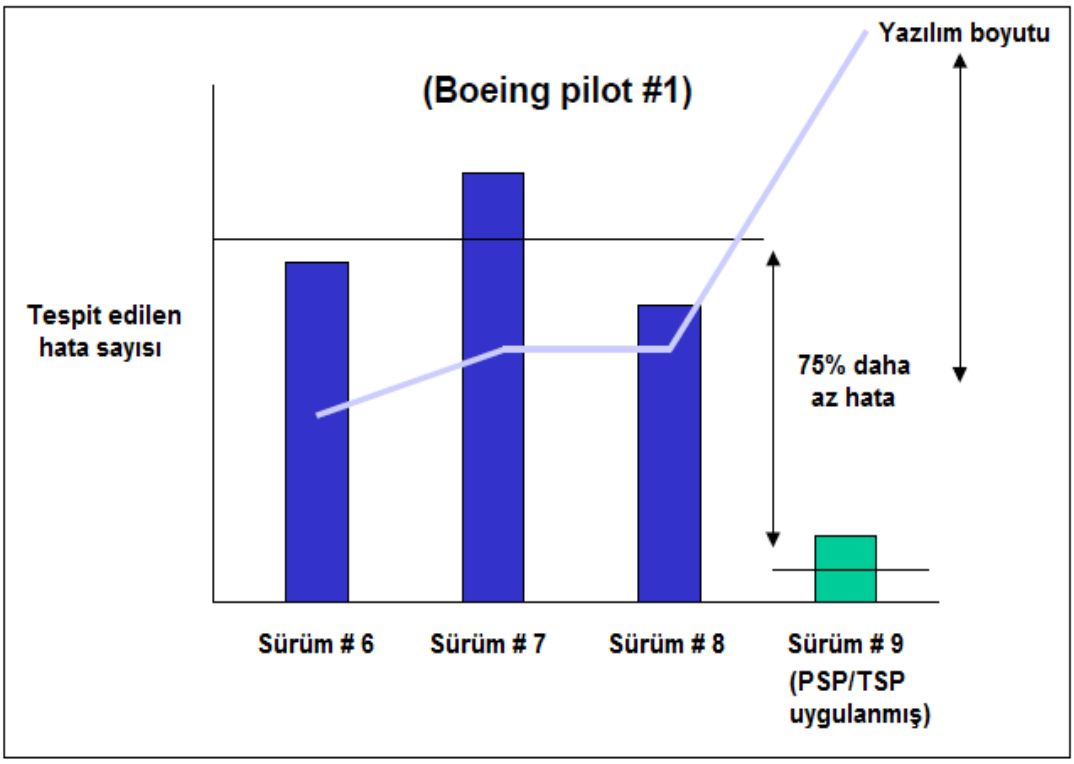
Kategori	**TSP olmadan**	**TSP varken**
Ortalama takvim sapması (aralık)	27% ile 112%	-8% ile 5%
Ortalama efor sapması (aralık)	17% ile 85%	-8% ile -4%
Kabul testi ürün kalitesi (hata/KLOC)	0.1* ile 0.7	0.02 ile 0.1
Sistem testinden kazanç (1 KLOC için)	1 ile 5 gün	0.1 ile 1 gün
Sürümden sonraki hata sayısı/KLOC	0.2 ile 1+	0 ile 0.1

* Bu veri (kabul testinde 0.1 hata/KLOC), CMM Seviye 5 uyumlu bir organizasyondan alınmıştır.

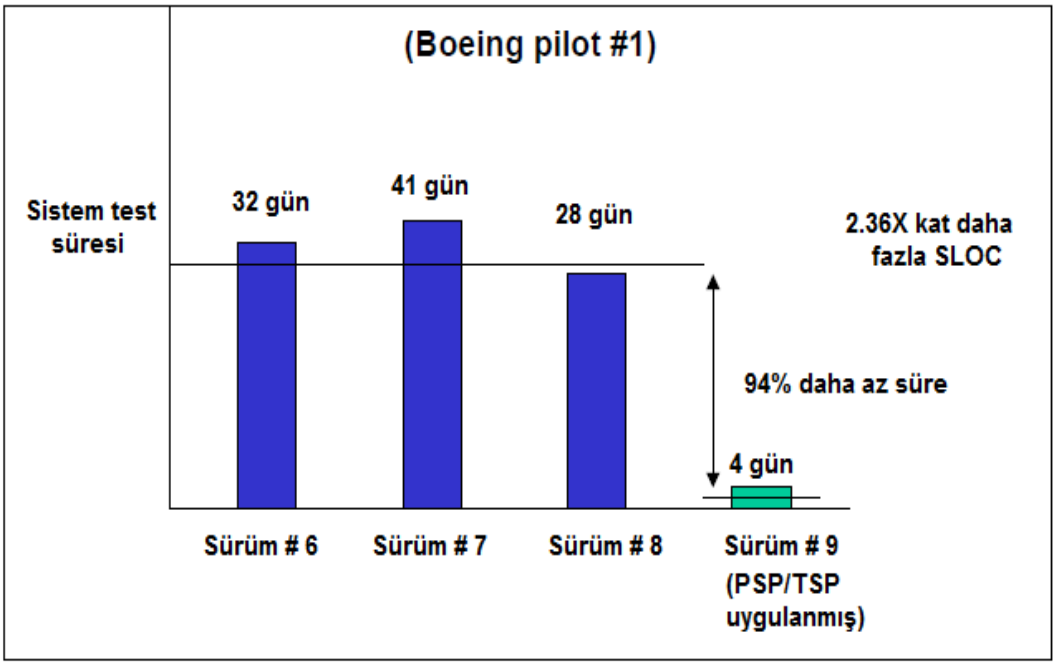
TSP ile deneyimlerini paylaşan Teradyne [11], TSP'den önce entegrasyon testi, sistem testi, saha testi ve müşteri kullanımında ortalama her 1000 satır kod için 20 hata olduğunu tespit etmiştir. İlk TSP projeleri ise bu sayıyı her 1000 satır kod için 1 hataya düşürmüştür. Her hatayı bulup çözmenin maliyeti yaklaşık olarak 12 mühendislik saati olduğundan, Teradyne her 1000 satır kod için 228 mühendislik saati tasarruf sağlamıştır. Her 1000 satır kodu kodlayıp birim testi yapmanın maliyeti yaklaşık olarak 50 mühendislik saati olduğundan, hatayı düzeltmede sağlanan tasarrufun (228 saat), ilk aşamada kodlamanın maliyetinden 4.5 kat daha fazla olduğu sonucu ortaya çıkmıştır.

Hill Air Force Base, CMM Seviye 5 alan ilk ABD devlet kurumudur [29]. Burada ilk TSP projesinde ekip verimliliği %123 iyileşmiş ve kurumsal ortalama test süresi proje takviminin %22'si iken, TSP ile bu oran %2.7'ye inmiştir [30].

Dünyanın en büyük hava-uzay şirketi olan "Boeing", TSP ile geliştirilen projelerden Şekil 2.7 ve 2.8'deki sonuçları [11] elde etmiştir. TSP ile sistem testinde %94 azalma sonucu proje takviminde önemli bir iyileşme olmuş ve şirketin yüksek kalite ürünleri planlanandan önce teslim etmesi sağlanmıştır.



Şekil 2.7. TSP hata sayıları



Şekil 2.8. TSP sistem testi süresi

3. İLİŞKİLİ ÇALIŞMALAR

Bu tez kapsamında yapılmak istenen çalışmaya benzer olarak, güvenlik uygulamalarının çevik yazılım geliştirme süreç modellerine nasıl ekleneceği veya çevik süreçlerin güvenliği destekleyecek şekilde nasıl yeniden yapılandırılacağı üzerine çalışmalar incelenmiştir.

Beznosov ve Kruchten [3], güvenlik güvencesi metotlarının çevik yöntemlere nasıl uyarlanacağı üzerine yaptığı çalışmada, güvenlik güvencesi metotlarını çevik yöntemlerle çakışma derecelerine göre sınıflandırmış ve uyumsuzluğu azaltıcı önermeler yapmıştır. Bu çalışmaya göre ikili programlama, kod ve tasarım gözden geçirme çevik geliştirmeyle doğal olarak uyuşmaktadır. Güvenli tasarım ilkeleri ve gereksinimler için kılavuzlar takip etmek, sürüm ve değişim kontrolü, uygulanan süreç modelinden bağımsız olduğu için çevik yöntemlerle uyuşmazlık yoktur. Yarı uyarlanabilir kategorisinde bulunan güvenlikle alakalı önleyici kodların statik analiz araçlarıyla test edilmesi, bilinen sistem açıklarına karşı uygulanan sistem testi ve nüfuz testi eğer otomatize edilmezse her iterasyonda tekrarlanması yüksek maliyet ve zaman gecikmelerine neden olabilir. Buna çözüm olarak, araçların otomatikleştirilmesi önerilmiştir. Ancak güvenlik testlerinin otomatikleştirilmesi, güvenlik problemlerinin sadece yarısı için bir çözüm olabilir. Her ne kadar test otomasyonu faydalı olsa da uygulamaya özgü test geliştirmesi, güvenlik uzmanı ve yazılım geliştiriciden ziyade bir güvenlikçi tarafından yapılan güvenlik yönelimli test anlayışı gerektirir. Bu durum farklı uygulama alanları ve sektörler için de geçerlidir. Örneğin, karmaşık bir bankacılık uygulamasının iyi bir şekilde tasarlanıp geliştirilmesi için, geliştiricilerin finans alanında kapsamlı bir bilgiye sahip olması gerekir. Bu açığı kapatmanın bir yolu olarak, güvenlik metot ve tekniklerinin uygulanması için gerekli olan bilgileri kullanarak bu analiz araçlarının geliştiricilerin kendileri tarafından kodlanması önerilmiştir. Güvenlik metotlarından çevik yöntemlere uymayanlar ise, sistemin kapsamlı dokümantasyonu ve harici güvenlik veya resmi doğrulama uzmanlarının gerekliliği olarak verilmiştir. Sonuç olarak, çevik yöntemlere uyarlanması çok yüksek maliyet ve zaman aşımlarına neden olacağı için bu metotlar en zor uyarlanacak olanlardır. Bunun için makalede iki tane olasılık sunulmuştur. Gerçekleştirmesi zor olan yöntem, agile yöntemlere uygun yeni güvenlik metotlarının oluşturulup var olanların yerine konmasıdır. Bu yöntemlerin ne olacağı konusunda bir anlayış sunulmamıştır ancak aşağıdaki özellikleri destekleyerek dokümantasyon, tekrarlı geliştirme, kod yeniden yapılandırma ve test anlayışları noktalarında çakışmaları ortadan kaldırması gerektiği savunulmuştur:

1. Doğrudan iletişim ve üstü kapalı bilgi,

2. Kısa ve sık iterasyonlar,
3. Paylaşımlı kod sahipliği ve aktif kod yeniden düzenleme (refactoring) ile tasarımın aşamalı ortaya çıkışı,
4. Kullanıcı hikâyeleriyle karar verilen test durumları ile oluşturulan test tabanlı geliştirme.

Önerilen diğer yönteme göre güvenlik metotları, geliştirme yaşam döngüsü boyunca en az iki defa uygulanmalıdır. İlki, projedeki ilk birkaç iterasyondan sonra, sonuncusu ise projenin sonlarına doğru, örneğin sistemin sahaya sunulmadan birkaç iterasyon öncesinde uygulanabilir. Sonuncusu nihai ürünün güvenliliğinin garantisinin sağlanması açısından gereklidir. İlki ise ana tasarım ve mimari kararların güvenlik özellikleri açısından erken garantisi için gereklidir ve projenin sonlarına doğru ortaya çıkabilecek büyük güvenlik açıklarının olasılığını azaltmaktadır. Bu birleşmenin en büyük zorluğu hala agile (çevik) karşıtı olan kapsamlı dokümantasyona neden olmasıdır.

Waryrnen [31] ise Beznosov ve Kruchten [3] tarafından önerilen güvenlik uygulamalarını çevik süreçlere uydurmanın tersi olarak, çevik yöntemlerden Extreme Programming modelinin, güvenlik mühendisliğini destekleyecek şekilde nasıl adapte edileceğini araştırmıştır. Çalışma teorik analiz üzerinedir, deneysel bir çalışma yapılmamıştır. Bu çalışmadan çıkarılan önemli sonuçlar aşağıda listelenmiştir:

1. Geliştirme takımı içerisinde, güvenlik risklerinin belirlemesi, güvenlikle ilgili kullanıcı hikâyeleri önerilmesi, ikili kodlama ile sistemin tasarım ve kodlama aşamasında gerçek zamanlı güvenlik gözden geçirmelerini yapması için bir güvenlik mühendisi olmalıdır.
2. Güvence argümanı oluşturmak için, güvenlik mühendisi ikili programlama aktivitelerini dokümente etmelidir.
3. Güvence argümanı oluşturmak için, güvenlik mimarisi dokümente edilmelidir.
4. İkili programlama statik doğrulama ve otomatik yürütmeler ile tamamlanmalıdır.

Waryrnen, yukarda bahsedilen çalışmadan [31] alınan sonuçlardan yola çıkarak, başka bir çalışmada [32], Extreme Programming modelinin Planning Game aşamasının, gereksinimlerin oluşturulmasını destekleyecek şekilde nasıl uyarlanacağını anlatmıştır. Bu çalışma, gereksinimleri belirleme sürecine kötüye kullanım senaryoları ve güvenlikle alakalı kullanım senaryolarının oluşturulmasını da dâhil etmeyi öneriyor. Bu yöntemin,

senaryolardaki tehditler ve buna karşı alınacak önlemlere karşılık olarak belirlenen gerekli kodlama ve tasarım standartlarının ilgili senaryolarla eşleşme yapmaya yardımcı olacağı öngörülüyor.

Başka bir çalışmada [33], Scrum süreci için güvenlik analizi ve tasarım uygulamalarının eklendiği güvenlik destekli Scrum versiyonu olan S-Scrum önerilmiştir. Bu yöntem güvenlik olaylarıyla ilgili analiz, tasarım ve doğrulamanın dâhil olduğu bir ekleme öneriyor. Ancak bu çalışma daha çok kötüye kullanım senaryolarıyla desteklenen güvenlik gereksinimlerini oluşturma sürecine odaklanmakta, tasarım, gerçekleştirim ve doğrulama aşamalarında ne tür uygulamalar yapılacağına dair bir yöntem sunmamaktadır. Web uygulamaları hedef alınarak, paydaşlar tarafından oluşturulan gereksinimlerden güvenlik için hikaye oluşturup eğer kritikse devam etmekte olan sprintin sonlandırılarak yeniden sprint planlaması yapılıp bu güvenlik gereksiniminin gerçekleştirimi ve testinin yapılmasını, kritik değilse sonraki sprinte kadar beklemesini öneriyor. Bu tez kapsamında önerilen Trustworthy Scrum modeli, yoğun dokümantasyon içeren kötüye kullanım senaryolarının kullanımını aza indirecek bir yöntem sunuyor. Ayrıca sadece web uygulamaları değil tüm yazılımlar için geçerli olabilecek bir yöntem sunmaktadır.

Scrum sürecine güvenliği entegre etme üzerine başka bir çalışma olan Secure-Scrum [34] ise, güvenlikle ilgili olayların S-Tag ve S-Mark denilen güvenlik işaret ve etiketleriyle gösterimine dayanmaktadır. Buradaki S-Tag güvenlik kaygılarının gösterimi için kullanılıyor, S-Mark ise S-Tag'ları ilgili iş listesindeki ilgili kaleme bağlayan ve bu kalemi diğerlerinden ayırt etmek için kullanılan bir işarettir. Bu yöntem S-Tag ve S-Mark arasında bağlantı kurup bunun devamlığını sağlamayı garanti ediyor ve belirlenmiş olan güvenlik olaylarının bu sayede kaybolmasının ve test edilememesinin önüne geçiyor. S-Tag, kullanım senaryosu, kötüye kullanım senaryosu veya güvenlik gereksinimini anlatan herhangi bir gösterim olabilir. Güvenlik olaylarını tanımlama işlemi, paydaşlar tarafından öneme sahip parçalara odaklanarak, güvenlikle alakalı kullanıcı hikâyeleri tanımlama üzerine kuruludur. Bu çalışmada, yazılımın sadece güvenliğe yönelik parçalarının tanımlanarak bu parçaların güvenliğinin sağlanması amaçlanmıştır. Ancak güvenlik tüm sistemi ilgilendiren gerekli bir niteliktir [7]. Örneğin, arabellek taşması, gizlilik (confidentiality), bütünlük (integrity), kullanılabilirlik (availability) gibi güvenlik özellikleri veya kritik bir arayüzde ortaya çıkmasından bağımsız bir güvenlik problemidir [35]. Göz önünde bulundurulması gereken diğer konu ise, yazılım güvenliğinin, riske dayalı güvenlik gereksinimlerinin belirlenip önleminin alınmasıyla beraber, yazılım

içerisinde güvenlik sorunlarına yol açacak, programlama veya tasarım hatalarından oluşabilecek içsel tehditlerin (insider) de irdelenmesini gerektirdiğidir [7], [35], [36], [37]. Sadece paydaşlarla belirlenen kullanıcı senaryolarında güvenlik olaylarına odaklanmak bunu sağlayamaz. Trustworthy Scrum güvenliğe, yazılımın sadece güvenliğe yönelik parçalarına eklenebilecek bir özellik yaklaşımıyla bakmak yerine, tüm yazılımı ilgilendiren ve geliştirme süreci boyunca hesaba katılan bir nitelik olarak bakmaktadır ve buna yönelik uygulamalar içermektedir.

İçsel tehditler kategorisinde yer alan kodlama hataları gibi istenmeden yapılan veya eğitim eksikliği nedeniyle oluşan gerçekleştirim ve sistem hataları vb. hatalar bu iki çalışmada [33], [34] hesaba katılmamıştır. Kullanım senaryosu oluşturmak gibi her iterasyonda tekrarlanması zaman alıcı ve dokümantasyon odaklı olan bu yöntemin etkisi azaltılamamıştır. Aynı zamanda güvenlik uygulamalarından olan eğitim, kodlama standartları, güvenli tasarım ilkeleri, kod ve tasarım gözden geçirmeler, statik analiz araçlarının kullanımı gibi uygulamalara değinilmemiştir. Trustworthy Scrum, bu güvenlik uygulamalarını de içeren bir modeldir.

Başka bir çalışmada [38], Scrum'a yeni scrum rolü, eseri ve etkinliği eklemeyi gerektiren, güvenlik gereksinimleri için ayrı bir güvenlik ürün iş listesi oluşturmayı öneren yöntem sunulmaktadır. Bu yöntemde güvenlik geliştirmesi ve standart geliştirme ayrı yürütülmektedir. Ancak bu yöntem yazılım güvenliğinin, yazılım geliştirme süreci boyunca güvenlik odaklı düşünüp güvenliğin her geliştirme aşamasına dâhil edilmesi ilkesini karşılamamaktadır. Güvenlik gereksinimleri ayrı bir listede tutulmakta ve diğer standart gereksinimlerle arasında bağlantı kurulamamaktadır. Trustworthy Scrum, aynı ürün iş listelerinde yer aldığından güvenlik gereksinimlerinin, standart gereksinimlerden ayırt edilecek şekilde etiketlenmesini önermektedir ve aralarında bağlantı kurulmasına imkân vermektedir.

İncelenen bu çalışmalarda genel olarak, geleneksel yaklaşımla güvenlik gereksinimlerini belirleyip güvenli tasarımın yapılması ilkesi projenin baştan sona kapsamlı tasarımını gerektirdiği için, çevik yöntemlerin aşamalı olarak ortaya çıkan tasarım yapısıyla çeliştiği üzerine ortak bir görüş mevcuttur.

Burada dikkat edilmesi gereken iki önemli nokta aşağıda verilmiştir:

1. Güvenlik uygulamalarının hepsinin çevik yöntemlerle uyumsuzluğu söz konusu değildir. Diğer taraftan her iterasyonda uygulanması maliyetli ve zaman alıcı olan

uygulamaları, her tekrar yerine proje büyüklüğü ve yapılan işlerin niteliğine göre belli sayıdaki iterasyonlar sonrasında yani dönüm noktalarında yaparak, aynı zamanda çevik yöntemin de iteratif yapısını avantaja çevirerek (risk analizine girdi sağlaması gibi) modifiye etme yöntemi seçilebilir.

ÇÖZÜM: Bundan yola çıkarak güvenlik uygulamaları her iterasyonda ve dönüm noktalarında gerçekleştirilecek uygulamalar olarak 2 kategoriye ayrılacaktır.

2. İncelenen çalışmalar, tehditlerin saldırgan vb. harici kaynaklar tarafından yapılan ataklar ve programlama hatalarının dâhil olduğu içsel tehditler olabileceği [37] ayrımı hesaba katılmadan, tüm güvenlik gereksinimlerinin olası saldırıların kötüye kullanım senaryosu şeklinde belirlenmesi ve gösterimi üzerinde yoğunlaşmıştır.

   ÇÖZÜM: Bu çalışma kapsamında önerilen modelde riske dayalı güvenlik gereksinimleri saldırgan bakış açısıyla bakılıp oluşturulacak ve dönüm noktalarında gerçekleştirilecektir. Gerçekleştirim ve tasarım hatalarından kaynaklanabilecek içsel tehditleri önlemeye yönelik gereksinimler ise, talep yönetim sistemine güvenlik anahtar kelimeleri olarak girilecektir.

Bu çalışma kapsamında önerilen Trustworthy Scrum modelinde hem bu çakışmaları hafifletecek hem de yukarıdaki ilişkili çalışmalarda bahsedilen sorunları ortadan kaldıracak bir yöntem sunulmaktadır.

4. GÜVENLİ YAZILIM GELİŞTİRME UYGULAMALARI

Bu tez kapsamında yazılım güvenliği ile ilgili literatür araştırması yapılmış, yazılım süreçlerinin olgunluğunu değerlendirme modelleri (CMMs ve SAMM) güvenlik bakış açısıyla irdelenmiş ve yazılım geliştirme süreçlerinden özellikle güvenlik konusunun hedeflendiği çalışmalar olan Microsoft Security Development Lifecycle (SDL), Team Software Process for Secure Software Development (TSP-Secure) ve Correctness by Construction gibi yazılım geliştirme süreç modelleri incelenmiştir. Yapılan bu araştırmalara göre güvenli yazılım geliştirme süreç modellerinde bulunması gereken aşamalar aşağıda verilmiştir:

1. Güvenlik ekibi yazılım geliştirme organizasyonu bünyesinde oluşturulmalı bu yüzden tüm proje ekibi üyelerine güvenlik eğitimi verilmelidir. Gerektiğinde sürece güvenlik uzmanı dâhil edilmelidir.
2. Güvenlik gereksinimleri belirlenmeli ve kötüye kullanım senaryoları oluşturulmalıdır.
3. Tasarım aşamasında gerçekleştirilmesi gereken güvenlik eylemlerine değinilerek en iyi tasarıma karar verilmelidir.
4. Risk analizi yapılmalıdır.
5. Kodlama standartlarına ve güvenli kodlama ilkelerine uyulmalı, birim testleri yazılmalıdır.
6. Statik analiz araçları kullanılmalıdır.
7. Kod ve tasarım gözden geçirme yapılmalıdır.
8. Güvenlik testi, nüfuz testi, bulandırma testi yapılmalıdır.
9. Kod yeniden yapılandırma ile güvenlik eklemeleri, düzeltmeleri yapılmalıdır.
10. Güvenlik dokümanları oluşturulmalıdır. Bu dokümanlar kullanıcının hata yapmaması için kullanıcı kılavuzu gibi güvenlik sorunlarını önlemede ve geliştirici için tasarım dokümanları, yorumlarla desteklenmiş kaynak kod ve test senaryoları gibi güvenlik geliştirmesinde etkili olabilecek dokümanlardır.

Güvenli yazılım geliştirme uygulamaları alt başlıklar halinde ayrıntılı olarak anlatılmış, neden gerektiği ve çevik yöntemlerde uygulanabilirliği gibi konulara değinilmiştir.

4.1 Güvenlik Eğitimi

Güvenlik eğitimi, hangi yazılım geliştirme metodunun uygulandığından bağımsız olarak kritik öneme sahiptir ve hepsine uygulanabilir durumdadır. Basit güvenlik pratikleri uygulanmadığı sürece, hiçbir yazılım geliştirme modeli güvenli yazılım oluşturmaz. Yılda en az bir defa mühendislik güvenlik konusunda eğitilmelidir. Eğitim bir maliyet getirir ancak eğitimin getirdiği maliyet, güvenlik zafiyetinin riskleri ve maliyetine göre çok daha düşük seviyededir. Özellikle çevik yöntemlerde eğitim çok fazla kritik öneme sahiptir. Çünkü karar yetkisi daha çok ürün sahibi ve geliştirme takımındadır.

4.2 Güvenlik Gereksinimleri

Güvenlik gereksinimleri hem sistemin gizlilik, bütünlük, kullanılabilirlik gibi güvenlik özelliklerini sağlayan hem de sisteme saldırgan gözüyle bakarak olası atakları belirleyip önceden önlem almayı sağlayan gereksinimleri kapsar [7], [39], [40], [41], [42].

Güvenlik gereksinimleri genelde diğer gereksinimlerden ayrı olarak geliştirilir. Bu yüzden de spesifik güvenlik gereksinimleri ihmal edilmekte ve fonksiyonel gereksinimler güvenlik durumu yok sayılarak belirlenmektedir. Ayrıca saldırgan bakış açısı hesaba katılmadığı zaman da bu gereksinimler yetersiz olmaktadır. Bu yüzden saldırgan perspektifinden de bakmayı hesaba katan sistematik bir yaklaşım, güvenlik gereksinimlerini belirlemede yararlı olacaktır [41]. Bazı sistematik yaklaşımlara CLASP, SQUARE, temel güvenlik gereksinimleri yapıları (core security requirements artefacts) örnek verilebilir. Güvenlik gereksinimlerine resmi, şekilsiz yaklaşımlara ise REVEAL, Software Cost Reduction (SCR) ve Common Criteria örnek verilebilir.

Güvenlik gereksinimlerini belirlemede önemli bir kısım saldırgan gözüyle bakmaktır. Çünkü saldırganlar sistemin güvenlik özellikleri ile ilgilenmezler. Başarılı bir saldırı yapabilmek için daha çok sistemdeki hataları ve normal olmayan durumları ararlar. Atak modelleri (attack pattern) bu riskleri belirlemede faydalı bir yöntemdir [39]. Saldırganın perspektifini tanımlayacak diğer yöntemler kötüye kullanım senaryoları, atak ağacı ve tehdit modellemedir [43], [40], [44], [45].

Gereksinim mühendisliğinin önemi

Bazı çalışmalar, yazılımın gereksinim mühendisliği hatalarını, yazılımı sahaya sürdükten sonra düzeltmenin, gereksinim aşamasında tespit edip düzeltmeye göre 10-200 kat daha fazla maliyet getirdiğini göstermiştir [46], [47]. Çoğu yazılım projelerinde gereksinim, tasarım ve kod hatalarını düzeltmek toplam projeye harcanan eforun %40-50'sine mal

olmakta [48] ve gereksinim aşamasından kaynaklanan hataların oranı ise %50'den fazla olarak tahmin edilmektedir.

Bir başka çalışmaya göre güvenlik analizi ve güvenlik mühendisliği pratikleri, yazılım geliştirme döngüsünün erken aşamalarında devreye girerse bundan sağlanan yarar (ROI) %12-21 arasında değişmektedir [49]. NIST raporlarına göre güvenlik ve güvenilirlik bakımından hatalı yazılımların bozulma ve düzeltilme maliyetleri yıllık 59.5 milyar dolar tutmaktadır [50]. Yetersiz güvenlik gereksinimlerinin maliyeti gösteriyor ki bu alandaki küçük bir iyileştirme yüksek bir fayda sağlayacaktır. Bir yazılım, operasyonel ortamına sürüldüğünde güvenlik sorunlarını düzeltmek veya güvenliğini iyileştirmek çok zor ve çok pahalıdır.

Çevik yöntemlerde güvenlik gereksinimleri

Projenin başlangıç aşamasında güvenlik gereksinimleri belirlenmelidir. Müşteriye sorular sorularak güvenlikle ilgili isterlerin ortaya çıkmasına yardımcı olunmalıdır. Kullanım senaryoları ve kötüye kullanım senaryoları bu güvenlik gereksinimlerine dayanmalıdır.

Kötüye kullanım senaryoları üzerine düşünmek geliştiriciler için yazılımın saldırganların bakış açısıyla görülmesini sağlayarak güvenli bir yazılımın nasıl geliştirileceğinin daha iyi anlaşılmasına yardımcı olur. Bir başka amacı da yazılımın uygunsuz kullanıma nasıl tepki vereceği konusunda öncül bir bilgi oluşturup dokümante etmektir. Bunu belirlemede en kolay yöntem de bu konu üzerinde bir beyin fırtınası yapmaktır. Bunu yönetmek için de yazılım güvenlik uzmanları sistemin zayıf yönlerini belirlemek için tasarımcıya sorular sorabilir. Sonuç olarak geliştiriciler güvenlik açısından yapılmaması gereken olayları çıkarır. Örneğin; "javaScript kodu izin vermeyeceği için kullanıcı 50'den fazla karakter giremeyecek" [40].

Çevik yöntemde tasarım yapılırken ara ara küçük toplantılar yapılıp beyin fırtınası tarzında görüşmeler sonucu kararlar alınmaktadır. Bu toplantılarda kötüye kullanım senaryoları üzerine yoğunlaşılarak güvenlik gereksinimleri oluşturulabilir.

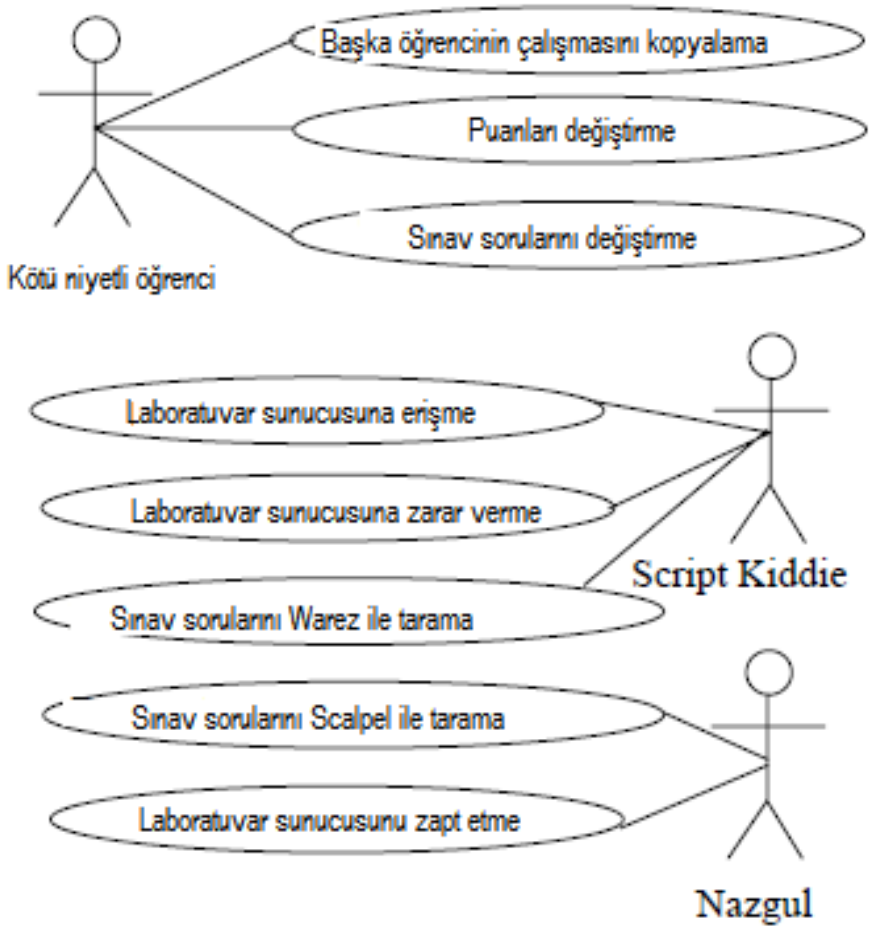
Eğer Scrum kullanılıyorsa bu durumda belirlenen kötüye kullanım senaryoları çizim yapılmadan ürün iş listesine diğer gereksinimlerden farklı bir etiketle işaretlenerek eklenebilir. Çevik yöntem ilkelerine göre sistemin tam tasarımı en baştan belli olmayacağı için başlangıçta güvenlik gereksinimleri tam anlamıyla belirlenemeyebilir. Buna çözüm olarak, Scrum'da her bir veya birkaç sprint sonrası oluşan sürümün güvenlik gereksinimleri belirlenebilir. Böylelikle aşamalı şekilde gelişen yazılımın, güvenliği de

aşamalı bir şekilde sağlanmış olacaktır. Oluşan ara sürümün tüm güvenlik gereksinimleri ürün iş listesine atıldıktan sonra bu listeden sıradaki sprint için seçilen güvenlik gereksinimlerinin gerçekleştirimi yapılabilir.

Belirtim yöntemleri

Gereksinimler sistemin neler yapması gerektiğini anlatır ve kullanım senaryoları ile gösterimi yapılabilir. Kullanım senaryoları bize herhangi bir sistemin nasıl davrandığını ve çalıştığını gösterir. Güvenlik gereksinimleri ise daha çok sistemin ne yapmaması gerektiğini gösterir ve bu yüzden kullanım senaryoları bunu ifade etmekte yetersiz olabilir. Buna çözüm olarak Guttom Sindre ve Andreas L. Opdahl [51] yanlış kullanım senaryosu (misuse case) tekniğini geliştirmiştir. Bu geliştirmeye göre yanlış kullanım senaryoları kullanım senaryolarının tersidir, sistemin izin vermemesi gereken fonksiyonları gösterir. Yanlış kullanım senaryolarındaki mis-actor, kullanım senaryolarındaki aktörün tersidir, sistemi kötüye kullanan ve sistemin desteklememesi gereken aktördür. Yanlış kullanım senaryoları mis-actorleri sınıflandırmayı ve sistemin ataktan önceki ve sonraki durumlarını göstermeyi de sağlar. Bu bakımdan yanlış kullanım senaryoları olası atakları, aktörleri, kayıpları ve atağın sistemdeki etkisini anlamada ve belirlemede oldukça kullanışlıdır

John McDermott ve Chris Fox, benzer şekilde kullanım senaryosu tekniğini güvenliği kapsayacak şekilde uyarlayarak kötüye kullanım senaryosu (abuse case) tekniği geliştirmiştir [43]. Şekil 4.1'de bu çalışmadan alınan örnek diyagram çizimi verilmiştir.



Şekil 4.1. İnternet tabanlı "Bilgi Güvenliği" dersinin laboratuvarı için abuse case diyagramı

4.3 Risk Analizi

Risk analizi sistemdeki riskleri değerlendirip analiz etmeye yönelik bir aktivitedir. Risk analizi yapılırken yazılımın spesifikasyonları, tasarımı, mimari dokümanları, veri hassasiyeti gibi konuların üzerine yeterli araştırma yapılıp bilgi sahibi olunmalıdır. Yazılımın karşılaşabileceği güvenlik olayları ve yazılımdaki kusurlar (flaw) belirlenir. Organizasyon değerlerini tehlikeye düşürebilecek potansiyel tehditler ve bunlara ilişkin riskler analiz edilir. Tehlikenin gerçekleşmesi durumunda ne tür etkilere neden olacağına dair analiz yapılır. Riskler etkilerine göre önceliklendirilir. Riski azaltmak için alınan önlemleri içeren bir hafifletme stratejisi geliştirilir. Yazılım, bu riski hafifletme çalışmalarının etkisini tespit etmek için yeniden değerlendirilir. Tüm araştırmalar, bulgular ve değerlendirmeler rapor edilir.

Risk analizi yapılırken özet olarak organizasyon değerleri, bu değerlere karşı olabilecek tehditler, sistem açıkları, riskler ve olasılıkları belirlenir. Risk analizi planlı, olaya dayalı veya ihtiyaca dayalı esaslarla yapılabilir.

Risk analizindeki bazı kavramlar aşağıda verilmiştir:

Değer: Organizasyonun korunan değerleri, veri bileşeni veya tüm sistem.

Risk: Bir değerin olumsuz bir etkiden değer kaybına uğrayacağının olasılığı. Birçok faktör bunu belirleyebilir: uygulamanın kolaylığı, saldırganın motivasyonu ve kaynakları, sistemin varolan açıkları.

Tehdit: Zararlı etkenin neden olacağı tehlike.

Sistem açığı: Hata, zayıflık.

Etki: Riskin gerçekleşmesi durumunda organizasyona etkisi. Parasal veya itibarla ilgili olabilir veya kanun, düzenleme, sözleşme ihlalleri ile sonuçlanabilir.

Olasılık: Verilen olayın gerçekleşme olabilirliği.

Tehditler sistemin ve değerlerin korunmasını ve güvenlik ilkelerini ihlal eden tehlikedir. Tehditler kod kırıcıları ve casuslar tarafından kaynaklanan bilinçli tehditler olabileceği gibi veri giriş hataları, programlama hataları veya eğitim yetersizliğinden kaynaklanan istenmeden yapılan hatalar da olabilir [37]. Tüm tehditler yazılım hatalarını kötüye kullanmayabilir. CERT ve US Secret Secure tarafından yapılan araştırmaya göre içsel tehdit denilen eğitim yetersizliği, ihmalkârlık, kötücül niyetli çalışanlar vb. nedenlerden kaynaklanan tehditlerin %57'si sistem açıklarını kötüye kullanmış veya denemiştir [52]. İstenmeden yapılan programlama hataları da bu içsel tehditlere dâhildir.

Risk yönetimi yazılım yaşam döngüsü boyunca riskleri yeniden değerlendiren sürekli bir işlemdir. NIST Special Publications Computer Security (NIST SP800-34) risk yönetimi aktivitelerini söyle tanımlar [53]:

1. Proje başlangıç aşamasında, değerler tanımlanır ve bu değerlerin zarar görmesi sonucu etkileri belirlenir. Riskler sistem gereksinimleri ve güvenlik operasyon konsepti bakımından değerlendirilir.
2. Geliştirme aşamasında, belirlenen riskler yazılımın güvenlik analizine destek sağlar.
3. Gerçekleştirme aşamasında, risk yönetimi, sistemin operasyonel ortamda gereksinimlere göre gerçekleştiriminin değerlendirilmesine destek sağlar.

4. Bakım ve operasyon aşamasında, periyodik olarak yapılan sistemin yeniden yetkilendirilmesi (veya onaylanması) veya yazılımda büyük değişiklikler yapılması durumlarında yeniden güvenliği değerlendirip sağlamak adına risk yönetimi aktiviteleri yapılır.

Risk analizindeki çoğu yaklaşımlar ölçülebilen değer konseptine dayanır. Örnek olarak bazı yaklaşımlar finansal kayıp, matematiksel risk derecesi veya niteliksel bir değerlendirme şeklinde olabilir. Klasik bir risk hesabı aşağıdaki denkleme dayanan bir finansal kayıp veya yıllık kayıp beklentisi olabilir [54]:

- **ALE** = SLE x ARO
- **ALE:** Yıllık kayıp beklentisi
- **SLE:** Tek kayıp beklentisi
- **ARO:** Yıllık beklenen kayıp sayısı

Örnek olarak, yetkisiz erişim izni veren bir açığı olan internet tabanlı alışveriş sitesi için bir adet güvenlik olayının maliyeti 150 dolar ise ve bu olay yılda 100 defa tekrarlıyorsa, yıllık kayıp beklentisi 15,000 dolardır. Bu büyük kayıp, bir miktar parayı hatayı düzeltmeye yatırmak için karar vermede oldukça etkilidir [54].

Çevik yöntemlerde risk analizi

Güvenlikle ilgili bir açığı düzeltme maliyetinin bu açığın verdiği zarardan daha fazla olabileceği durumlarda risk analizi yapmak doğru karar vermede oldukça etkili bir yöntem olacaktır. Bu yöntem de çevik yazılıma katkı sağlar.

4.4 Güvenli Tasarım

Tasarım aşaması güvenliği sağlamak konusunda oldukça önemli bir aşamadır. Çünkü güvenlik problemlerinin büyük bir kısmı tasarım aşamasındaki hatalardan kaynaklanmaktadır. Michael Howard ve David LeBlanc tarafından yapılan analize göre güvenlik hatalarının %50'si tasarım aşamasında bulunmaktadır [55].

Değerler, tehditler ve olası sistem açıkları belirlendikten sonra yazılımın atağa açık alanlarını en aza indirgeyecek şekilde tasarım yapılır. Howard ve Lipner'e göre atak alanı, kod parçası, arayüz, servis, protokol (iletişim kuralı) ve özellikle yetkisiz kullanıcılar olmak üzere tüm kullanıcılara açık olan uygulamalar olarak tanımlanmıştır [56]. Bu alanı minimize etmek de yazılımda oluşabilecek olası atakları azaltmak anlamına gelir.

Çoğu yazılım geliştiriciler, yazılım güvenliği konusunda bir güvenlik uzmanın tecrübesi ve aldığı dersler kadar tecrübe edinmemiş olabilir. Ancak bu boşluğu kapatmak için yazılımda güvenliği sağlama adına gerçek deneyimlerden yola çıkılarak oluşturulan pratikleri sunan ilkeler, kılavuzlar ve kodlama kuralları gibi kaynaklardan yararlanılabilir. Örnek olarak Jerome Saltzer ve Michael Schroeder güvenli tasarım ilkelerini, koruma mekanizmaları kapsamında ilişkilendirip gruplandırmıştır [57].

Çevik yöntemlerde güvenli tasarım

Çevik yöntemlerde tasarım aşamalı olarak geliştirildiği için bu aşamalarda kazanılan deneyim ve alınan dersler, yapılan hataların bir daha yapılmaması adına ileride kılavuz olarak kullanılabilir.

Çevik yöntemlerde erken aşamalarda ciddi tasarım hataları olabilir. Amaç bu hataları anlamak ve sonraki iterasyon için artırımsal değişiklikler yaparak hataları düzeltmektedir. Ayrıca çevik yöntemlerin bir ilkesi de basit tasarımdır ve tasarımın gereksinimleri karşılaması yeterlidir [26]. Güvenli yazılım geliştirmenin "Kompleks yazılım zordur ve güvenli hale getirmek mümkün olmayabilir" [9] ilkesi de bunu destekler niteliktedir.

4.5 Güvenli Kodlama ve Test

Güvenli kodlama ilkelerine göre geliştirme yapılmalı, statik analiz araçları ile kod taranmalı, kod gözden geçirmelerle araçların bulamadığı hatalar bulunmalıdır. Teste oldukça çok yer verilmeli, hata bulundukça bunun için test yapılmalıdır. Bunlar güvenlik için de geçerlidir. Örneğin tam sayı taşması olursa bunu tetikleyen güvenlik testi yazılmalıdır. Test, kodun her derlenmesinde çalıştırılarak hatanın giderildiğinden emin olunmalıdır. Tüm hatalar ve güvenlik hataları için testler her çalıştırıldığında başarılı olmalıdır.

Sistem açıkları gerçekleştirim seviyesindeki ve tasarım seviyesindeki hatalar olmak üzere iki kategoriye ayrılabilir. Gerçekleştirim seviyesindeki hatalar için kaynak kod analizi yapılır. Tasarım seviyesindekiler için ise güvenli tasarım ilkeleriyle ilgili kaynaklardan ve organizasyonun rapor ettiği tecrübelerden yararlanılır. Yazılımlarda ciddi güvenlik açıklarına neden olan ve oldukça yaygın olan gerçekleştirim hataları aşağıda verilmiştir:

Yarış Durumu (Race Condition): Doğru senkronize edilmemiş işler (thread) arasındaki zamanlama problemlerinden ortaya çıkan, aynı zaman diliminde bir kaynağa erişme mücadelesi olarak tanımlanabilir. Herhangi bir zamanaşımı kontrolü olmadan yazılımının hiç bırakılmayacak bir kaynağı beklemesi (deadlock), paylaşılan kaynaklara eşzamanlı

erişim hataları (resource collision), programın sonlanmasına izin vermeyen mantıksal veya kontrol akışlar, sonsuz döngüler (infinite loops) örnek verilebilir.

Girdi Geçerleme (Input Validation): Kullanıcı veya parametre girdilerine güvenmek sorunlara yol açabilir. Semantik veya SQL enjeksiyonlar gizlilik (confidentially) ve bütünlük (integrity) için risk oluşturur. SQL enjeksiyon veritabanındaki hassas bilgiler için tehlike oluşturur.

Olağandışı Durumlar (Exceptions): Kodun normal akışını bozan olaylardır. Exception handling programın anormal olayları atlatmasını sağlar.

Bellek Taşması (Buffer Overflow): C, C++ kodlarının, dizin (array) ve işaretçi (pointer) yapılarında sınır kontrolleri olmadığı için emniyetsiz oluşundan dolayı geliştirici bu kontrolleri yapmalıdır. Bu açıktan yararlanan saldırganlar uzaktan zararlı kod enjeksiyonu ile yazılımı kötüye kullanabilir.

Yığın Taşması (Stack Overflow): Bellek taşmasının bir formu olan yığın taşması, programın çalışma zamanı için ayrılan belleğin dikkatsiz kullanımıdır. Saldırgan kontrolün zararlı koda geçmesini sağlayabilir.

Tamsayı Taşması (Integer Overflow): Tamsayıyı ikili gösterime yetmeyecek bir alana yerleştirme girişiminde ortaya çıkar.

Statik analiz araçları kaynak kodu tarar ve derleyicinin yakalayamadığı daha sonra problem yaratabilecek hataları bulur. Elle yapılan denetimler tek başına çok zaman alıcı olabileceği için statik analiz araçları hızlı bir şekilde kodu tarayarak belirli hataları bulmada oldukça etkilidir.

Kod gözden geçirme, mantıksal hataları ve statik analiz araçlarının bulamadığı hataları bulmada etkilidir. Risk analizi ile beraber kaynak kod gözden geçirmesi en iyi yazılım güvenlik pratikleri arasında yüksek öneme sahiptir [64]. Usulüne uygun gerçekleştirilen tasarım ve kod gözden geçirme, yazılım güvenliğinde önemli ölçüde iyileştirmeler sağlar. Gözden geçiriciler kodlama standartlarını dikkate alarak kodu inceler, birim test planı yapar, gereksinimlere uygunluğu kontrol eder, işlevselliklerin doğru gerçekleştirildiğini kontrol ederler. Yorum, belgeleme, birim test planı, gereksinimlere uygunluk gibi özelliklerin listelendiği kod gözden geçirme kontrol listelerini kullanırlar.

Özet olarak güvenli kodlama ve test ilkeleri aşağıda maddeler halinde verilmiştir:

1. Kodlama ve test standartlarını uygula: Kodlama standartları yazılımcının güvenlik açıklarına neden olacak hatalar yapmasını önler. Örneğin, güvenli bir dizgi ele alma (string handling) ve arabellek işletmesi (buffer manipulation), arabellek taşması (buffer overrun) zafiyetlerinin meydana gelmesini önler. Test standartları sadece yazılımın doğru işleyişine yoğunlaşmaktan ziyade, testin potansiyel güvenlik açıklarını bulmasına da odaklanır.
2. Bulandırma araçlarını (fuzzing tools) içeren güvenlik testi araçlarını kullan: Fuzzing yazılıma geçersiz girdiler verip test edilmesini sağlayarak, zafiyete yol açacak hataların bulunmasını azami seviyeye çıkarır.
3. Statik kod analizi yapan araçlar kullan: Bu araçlar, arabellek taşmaları (buffer overruns), tamsayı taşmaları (integer overruns) ve ilk değer verilmemiş değişkenler (uninitialized variables) gibi zafiyetlere neden olabilecek kodlama hatalarını tespit eder.
4. Kod gözden geçirme uygula: Kod gözden geçirme, yazılımcıların kaynak kodu incelemesi ve potansiyel güvenlik açıklıklarını bulup yok etmesini sağlayarak otomatik araçlara ve testlere destek olur. Mantıksal hataları ve otomatik araçlarının bulamadığı hataları bulmada etkilidir.

4.6 Yazılım Güvenlik Testi

Güvenlik testi, hem güvenlik özellikleri fonksiyonlarının gerçekleştirildiğinin hem de saldırganın bakış açısıyla sisteme bakıp olası sistem açıklarının hesaba katılarak yapıldığının testini gerektirir [35]. Güvenliğin sadece güvenlik mekanizması ve özelliklerinden ibaret olduğu konusundaki yanlış kanı, güvenlik testinde de yaygındır. Güvenlik, tüm sistemi ilgilendiren gerekli bir niteliktir. Örneğin bellek taşması, güvenlik özelliklerinde veya kritik bir arayüzde ortaya çıkmasından bağımsız olan bir güvenlik problemidir [35]. Bu yüzden güvenlik testi aşağıdaki iki yaklaşımı içermelidir [35]:

1. Güvenlik mekanizması fonksiyonlarının doğru gerçekleştirildiğinin testi,
2. Saldırganın bakış açısıyla bakıp sistem açıklarını göz önünde bulundurarak yapılan riske dayalı güvenlik testi.

Güvenlik mekanizması fonksiyonelliği, geleneksel yaklaşım kullanan standart test organizasyonları tarafından yapılabilir. Örneğin erişim denetim mekanizmasının beklendiği gibi çalıştığının testi standart bir işlemdir. Geleneksel kalite güvence ekibi risk tabanlı güvenlik testi için zorlanabilir. Çünkü bunun için tasarımcı saldırgan gibi düşünebilmelidir. Ayrıca güvenlik testleri direkt olarak güvenlik istismarı yaratmayabilir ve gözlemlenmesi

gerekebilir. Güvenlik testi, testçinin daha sonra kapsamlı, detaylı analiz yapmasını gerektirecek öngörülemeyen sonuçlar verebilir. Kısacası risk tabanlı güvenlik testi uzmanlık ve deneyim gerektirir [35].

Gary McGraw tarafından yapılan araştırmaya göre Cigital [35] birkaç yıl önce JavaCard platformunda çalışan akıllı kartları otomatik test etmek için bir yapı geliştirmeye başlamıştır. Bu yapı, minimum insan müdahalesi gerektiren ve risk analizlerine dayanan bir yapıdır. İlk test seti fonksiyonel güvenlik test setidir ve sınıf kodları ve yorumları kripto fonksiyonelliğini test etmek üstüne kurulmuştur. Çoğu kartlar bu fonksiyonel güvenlik testlerinin hepsini geçmiştir. Ancak risk tabanlı test yaklaşımına göre saldırgan uygulama takımı ile yapılan testlerde tüm kartlar başarısızlık sergilemiştir.

4.7 Kod Yeniden Yapılandırma (Refactoring)

Kod yeniden yapılandırma, kodu daha temiz okuması ve devam ettirmesi daha kolay, daha kaliteli ve hatta daha güvenli yapmak için iç gösterimini iyileştirmektir.

Güvenlik gereksinimi olarak, sürekli olarak sistematik bir şekilde eski kodlar gözden geçirilir, güvenlik hataları aranır, eğer bulunursa düzeltilir. Kod yeniden yapılandırma yapılırken kodun arayüzü veya davranışları değiştirilmeden, kodun kendisinde yeniden yapılandırma ve değişimler yapılır. Güvenlik hataları, kod yeniden yapılandırma işleminin bir parçası olarak görülmelidir. Örnek olarak, güvenli kod yeniden yapılandırma aşağıdaki işlemleri içermelidir [9]:

1. Yasaklanan API'leri güvenli API'ler ile değiştir. Örneğin, "strcpy" metodunu "StringCchCopy" veya "strcpy_s" ile değiştir.
2. Zayıf kripto algoritmalarını güncel ve daha güvenli versiyonlarıyla değiştir.
3. Gömülü tanımlanmış algoritma isimleri, anahtar boyutları ve diğer kriptografik değişkenleri kaldırarak kriptografik kodu daha çevik hale getir.
4. Bellek ayrılması ve dizin endekslemede kullanılan tamsayı aritmetiğini daha güvenli kod ile değiştir.

Eğer mevcut kod hassas ve kişisel veri işliyor ve bu veri internete açılıyorsa, tüm kod gözden geçirilmeli, tüm kod yeniden analiz edilip yeni testler oluşturulmalı ve tüm hatalar giderilinceye kadar yeniden yapılandırma işlemleri yapılmalıdır.

Sahaya sürülmeden önce kodun güvenlik açısından gözden geçirilip yeniden yapılandırılması güvenliği sağlama konusunda oldukça fayda sağlar.

Ayrıca 3 yıl ya da daha uzun süre dokunulmamış kod aşağıdaki nedenlerden dolayı güvenlik hatalarına sahiptir [9];

1. Güvenlik alanı, hem olumlu yönden hem de olumsuz yönden sürekli olarak önemli derecede gelişmektedir, daha çok da olumsuz yönden ilerlemektedir.
2. Güvenlik araçları, olumlu ve olumsuz yönden çok çabuk gelişmektedir.
3. İnsanlar, güvenlik hatalarını bulmada daha iyiye gitmektedir.

5. SCRUM İLE GÜVENLİ YAZILIM GELİŞTİRME: TRUSTWORTHY SCRUM

Geleneksel güvenli yazılım geliştirme modellerindeki güvenlik yaklaşımı projenin başlangıç aşamasında:

1. Detaylı baştan sona güvenlik mimarisi,
2. Detaylı tehdit modelleme ve risk analizi,
3. Güvenlik gerçekleştirimi için talimatların ve gereksinimlerin belirlenmesini gerektirdiği için çevik yöntemlerle çakışmaktadır.

Buna karşılık olarak çevik yöntemlerde güvenlik uygulamaları aşağıdaki ilkeleri benimsemelidir:

1. Kademeli olarak belirlenen güvenlik gereksinimleri,
2. Başlangıçta minimal güvenlik tasarımı ve gereksinimleri,
3. Gelişmekte olan güvenlik tasarımı, yaşam döngüsü boyunca birçok tasarım kararı,
4. Güvenlik gereksinimlerinin, tasarımı ve gerçekleştirimi sık ve kısa iterasyonlarla yapılabilecek şekilde belirlenmesi,
5. Güvenlikle ilgili geliştirmeler sonucu teslim edilebilir ürün ortaya konması.

Bu tez kapsamında önerilen modelde çakışmayı azaltmak için:

- Yukarıda verilen geleneksel yöntemlerdeki üç yaklaşım, çevik yöntemlerde güvenlik uygulamaları ilkeleri olarak verilen beş yaklaşımla değiştirilecektir.
- Bu güvenlik uygulamalarının çevik modellerin her iterasyonunda değil birkaç sayıdaki iterasyonlar sonrasında yapılması önerilecektir.

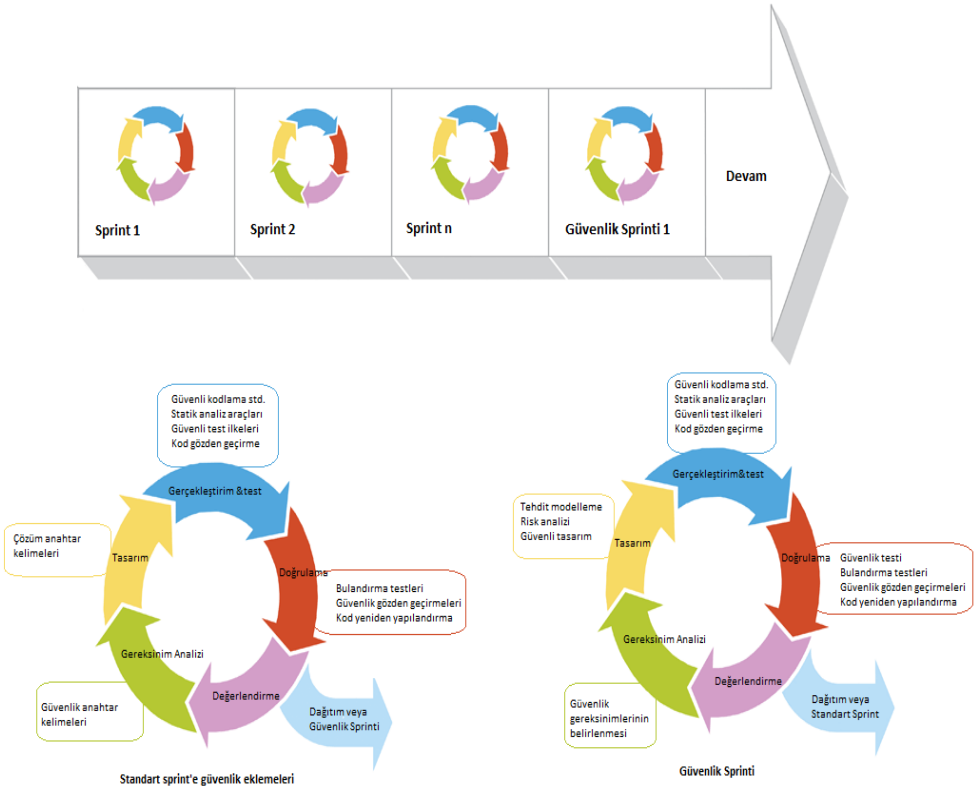
"Güvenlik Sprinti" başlığı altında birkaç sayıdaki iterasyon sonrasında (dönüm noktaları) yapılması amaçlanan bu uygulamalar için bir yöntem önerilecektir. Scrum etkinliklerine bu iterasyon, Güvenlik Sprinti olarak eklenecektir. Güvenlik Sprintindeki uygulamalar, her iterasyondan ziyade projenin başında ve proje büyüklüğü ve iterasyonlarda/sürümlerde gerçekleştirimi yapılan işlevler ve özelliklere göre her seferinde belli sayıda iterasyon sonrasında yapılacak olan güvenlik uygulamalarıdır. Biraz daha fazla zaman ve dikkat gerektiren bu uygulamaların her iterasyonda değil de proje büyüklüğü ve iterasyonlarda gerçekleştirimi yapılan işlevler ve özelliklere göre belli sayıdaki iterasyonlar arasında uygulanması ile hem fazla maliyetin önüne geçilecek hem de çevik yöntemin aşamalı ilerleyen artırımlı yapısından güvenlik açısından da yararlanılmış olacaktır. Kademeli

olarak verilen hızlı kararlar, gerçekleştirimler ve hızlı geri dönüşler sayesinde alınan derslerden ve tecrübelerden tasarım gittikçe iyileştirilecektir.

Çevik yöntem ile güvenliğin çakıştığı noktalardan bir tanesi de güvenlik gereksinimlerini belirleme aşamasında dışarıdan bir güvenlik uzmanına ihtiyaç duyulması ve her iterasyonda uzmanın sürece dâhil edilmesinin maliyetli olmasıdır. Ancak önerilen bu yöntemde, güvenlik uzmanına gerek duyulacak güvenlik gereksinimlerinin dönüm noktalarında yapılması bu sorunu hafifletecektir. Bununla beraber içeriden güvenlik uzmanı yetiştirmek de değerlendirilebilir.

İstenmeden yapılan gerçekleştirim hataları gibi içsel tehditleri önlemeye yönelik olan uygulamalar ise Scrum'ın var olan standart sprintlerine eklenecek ve "Her Sprint için Eklenen Güvenlik Uygulamaları" başlığı altında verilecektir.

Bu çalışma kapsamında önerilen modelin (Trustworthy Scrum) genel resmi Şekil 6.1'de verilmiştir.



Şekil 5.1. Trustworthy Scrum süreci

5.1 Her Sprint için Eklenen Güvenlik Uygulamaları

İstenmeden yapılan programlama hataları ve tasarımsal hatalar gibi içsel tehditleri önlemeye yönelik olan bu uygulamalar Scrum'ın var olan standart sprintlerine eklenecektir. Bu iterasyonlarda, güvenlik gereksinimleri için yanlış kullanım veya kötüye kullanım senaryoları belirleyip, gösterimini yapmak yerine "Güvenlik Anahtar Kelimeleri" kullanılacaktır. Güvenlik anahtar kelimeleri, sprint iş listesindeki bir kalemi geliştirme aşamasına gelince belirlenecek olup, tasarımı belirleme, kodlama, test ve gözden geçirme aşamalarında güvenlik açısından dikkat edilmesi ve uyulması gereken hususları belirleyecektir. Benzer şekilde kalemin gerçekleştirmesini yapan geliştirici bu anahtar kelimeleri odak noktasında tutarak güvenli kodlama standartları ve tasarım ilkelerinden yararlanarak bulduğu çözümü "Çözüm Anahtar Kelimeleri" olarak girecektir. Bu şekilde organizasyonun güvenlik olayları ve bu olayları nasıl elden geçirdiğine dair organizasyonel seviyede tasarım ve kodlama kılavuzu ortaya çıkacaktır. Bu bölüm Scrum'ın "Geliştirme Takımı" için uygulamalar sunmaktadır.

5.1.1 Güvenlik Anahtar Kelimeleri

Ürün İş Listesindeki kalemler için, olası gerçekleştirim hatalarını ve güvenlikle ilgili dikkat edilmesi gereken hususları ifade eden "Güvenlik Anahtar Kelimeleri" girilir. Bu kelimeler tasarım ve kodlama aşamasında dikkat edilecek güvenlik ilkelerini belirler. Örneğin; "Bellek taşması".

5.1.2 Çözüm Anahtar Kelimeleri

"Güvenlik Anahtar Kelimesi"yle ifade edilen açığın gerçekleşmemesi için bulunan yöntem veya alınacak önlemler "Çözüm Anahtar Kelimeleri" olarak girilir. Örneğin; "Bellek taşması" güvenlik anahtar kelimesine sahip olan bir kalem için çözüm anahtar kelimesi olarak "tüm dizin sınırlarının kontrolü" ve "tüm allocation (ayırma) işlemlerini deallocation (ayırmayı kaldırma) işlemleriyle eşleştirme" gibi güvenlik ilkelerini tarif eden çözümler girilir.

5.1.3 Güvenli Kodlama ve Test İlkeleri

Güvenli kodlama standartları ve güvenli tasarım ilkeleriyle ilgili kaynaklarda "Güvenlik ve/veya Çözüm Anahtar Kelimeleri"ne değinen ilkelere göre geliştirme yapılır ve testleri yazılır. Statik analiz araçları ile kaynak kod taranarak hatalar giderilir. Kod gözden geçirmeler bu güvenlik ve çözüm anahtar kelimeleri odak noktasında tutularak yapılır. Doğrulama aşamasında bulandırma testleri ile yazılım test edilir.

Örneğin, veritabanından sorgu yapılmasını gerektiren bir gereksinim için “SQL Enjeksiyon” kelimesi girilerek, kodlama yaparken SQL enjeksiyona imkân vermeyecek gerçekleştirim yapılmasına dikkat edilir ve testleri yapılır. Kod gözden geçirmelerde SQL enjeksiyona neden olabilecek hatalar aranır.

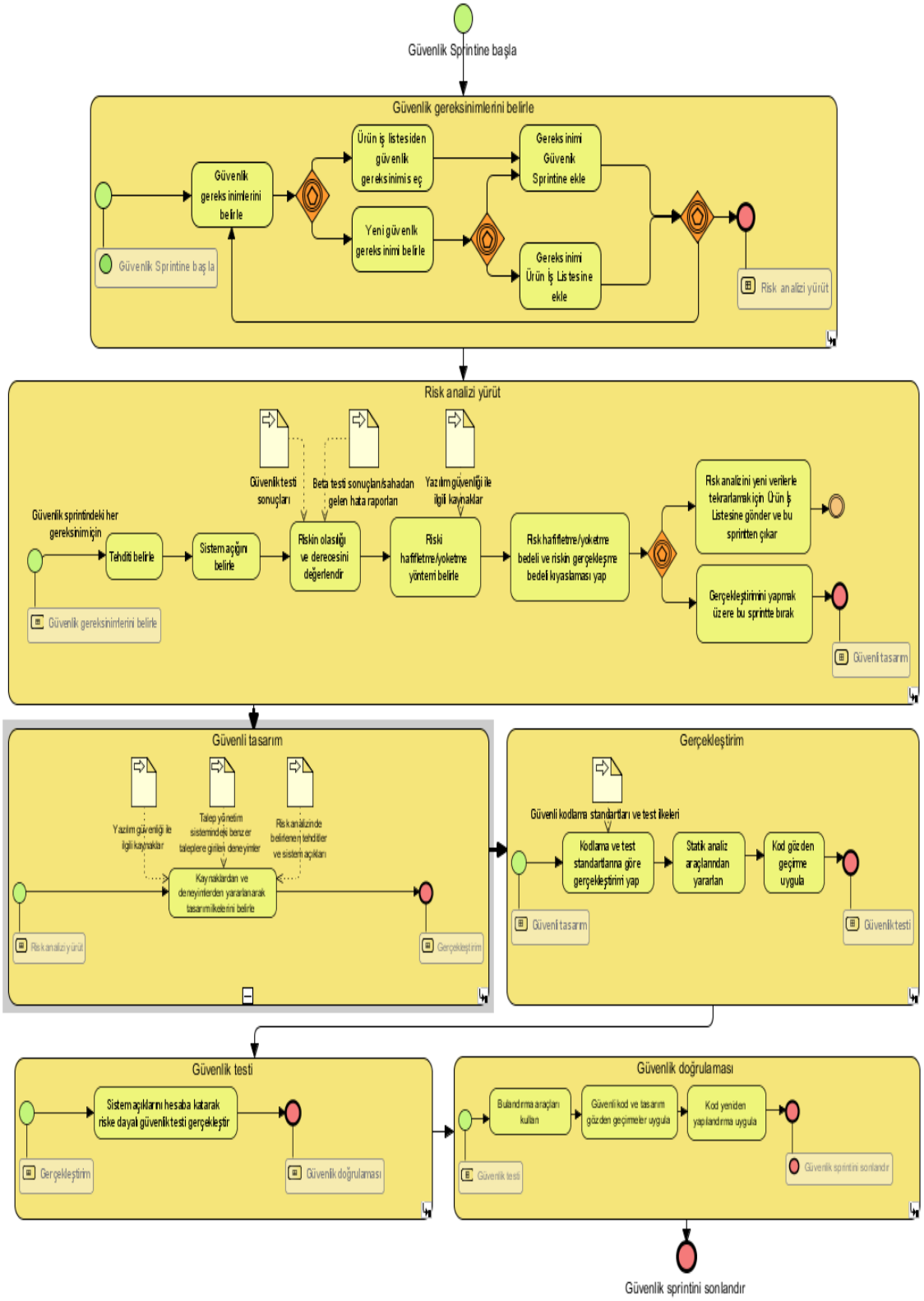
Başka bir örnek olarak, uzun kod yazımının önüne geçmek, sorgu cümlelerini hatalı yazma olasılığını azaltmak ve bakımı kolaylaştırmak adına verileri saklama, güncelleme ve silme gibi işlemlerin gerçekleştiriminde “hibernate teknolojisi ve DAO tasarım deseni” seçilmesi verilebilir.

5.1.4 Bitti Tanımı

Kalemi “Bitti” konumuna getirebilmek için hem gerçekleştirim, hem test, hem de gözden geçirme aşamalarında “Güvenlik ve/veya Çözüm Anahtar Kelimeleri”nin gerekliliklerinin yapıldığı onaylanmalı, uygulanabilir olmayanlar “Uygulanabilir Değil” diye işaretlenmelidir. Bunun dışında zaman yetersizliği vb. nedenlerle geliştirmesi yapılmayan kalem “Ertelendi” işaretiyle tekrar Ürün İş Listesine tarihçesiyle birlikte gönderilir. Bu şekilde güvenlik gereksinimleri tamamlanmayan yani güvenliği sağlanmayan kalemlerin kapanmasına izin verilmez, sonraki sprintlerde kaldığı aşamadan devam edilmesi sağlanır.

5.2 Güvenlik Sprinti

Güvenlik sprinti, hem güvenlik gereksinimlerinin belirlenip Ürün İş Listesine eklenmesini, hem de bu sprint için seçilen güvenlik gereksinimlerinin gerçekleştirilmesini kapsar. Güvenlik sprinti, diğer standart sprint gibi birbirini biter bitmez takip etmez. Projenin başlangıç aşamasına ve gerek duyuldukça belli sayıdaki iterasyonlar arasına Güvenlik Sprinti konabilir. Şekil 6.2'de Güvenlik Sprinti akış şeması verilmiştir.



Şekil 5.2. Güvenlik Sprinti akış şeması

Güvenlik sprinti, Scrum'ın standart sprintinde yapılan "Scrum Etkinlikleri"nin güvenlik bakış açısıyla yapılmasını ve "Scrum Takımı"na güvenlik uzmanı dâhil edilmesini önerir. Bu bölüm tüm "Scrum Takımı"na uygulamalar sunar. Güvenlik uzmanı, "Scrum Master"

ve “Ürün Sahibi”ne ürünün güvenlik gereksinimlerini belirlemesine yardımcı olur ve “Geliştirme Takımı”na güvenlik geliştirmesi konusunda destek sağlar. Güvenlik sprinti, standart sprint gibi düzenli olarak biribirini takip etmediğinden ne zaman yapılacağına “Scrum Master” tarafından “Sprint Değerlendirme” sonuçlarına göre karar verilebilir.

5.2.1 Güvenlik Gereksinimlerinin Belirlenmesi

Burada riske dayalı güvenlik gereksinimleri odak noktasıdır. Dış tehditler ve saldırgan bakış açısıyla bakılarak olası tehditleri ve gereksinimleri belirlemeyi gerektirir. Aktör, tehdit, güvenlik gereksinimi, risk derecesi gibi gereksinimi belirleme ve risk analizine yardımcı olabilecek ayrıntılar kaleme eklenir ve Ürün İş Listesine atılır. Bu tanımlar gereksinimleri belirleme aşamasında beyin fırtınası yapmayı kolaylaştırır. Diğer gereksinimlerden ayırt etmek için kalemin sınıfı “Güvenlik Gereksinimi” yapılabilir veya etiketle işaretlenebilir. Gösterim olarak kötüye kullanım senaryolarından da yararlanılabilir.

5.2.2 Risk Analizi

Risk analizi planlı, isteğe bağlı veya olaya bağlı olarak gerçekleştirilebilir. Risk analizi aşağıdaki adımlardan oluşur:

1. Risk analizi için gerekli kavramlar belirlenir ve talep yönetim sisteminde ilgili kaleme girdi sağlanır. Temel olarak tehdit, sistem açığı, tehditin olma olasılığı belirlenir.
2. Tehlikenin gerçekleşmesi durumunda ne tür etkileri neden olacağına dair analiz yapılır ve gereksinime riskin derecesine göre “Rank” değeri atanabilir.
3. Hafifletme stratejisi belirlenir.
4. Kaybın maliyeti ve önleyici aktivitenin maliyeti arasında kıyas yapılarak yani risk analizi yapılarak;
    a. Geliştirmenin bu sprintte yapılacağına karar verilirse bu güvenlik sprintinin iş listesine atılır veya
    b. Riskin derecesi yüksek görünmüyorsa daha sonra yapılmak üzere veya yeni verilerle yeniden risk analizi yapılmak üzere Ürün İş Listesine gönderilebilir.
5. Yeniden yapılan her risk analizinde çıkan sonuca göre gereksinimin rankı artırılır/azaltılır veya riskin derecesi güncellenir.

Risk analizi için farklı yaklaşımlar tercih edilebilir. Çoğu yaklaşımlar ölçülebilen değer konseptine dayanır. Basit bir güvenlik risk analizi işlemi Çizelge 6.1'de verilmiştir [58]:

Çizelge 5.1. Örnek risk analizi

Güvenlik gereksinimi	ATM'de banka görevlisi tarafından para bölümünü doldurma işleminin red olunamazlığı ("non-repudiation").
Tehdit	Görevli para bölmesini tam doldurup, bir kısmını geri alabilir.
Sistem açığı	Sistem tam doldurduğunu kontrol ediyor ancak sonradan geri çekilen parayı hesabı katmıyor.
Olasılık ve Risk Derecesi Değerlendirmesi	Yüksek olasılıklı ve yüksek riskli değerlendirmesi yapıldı.
Risk hafifletme/yok etme yöntemi	Loglama mekanizması geliştirme

Ürün İş Listesinden seçilen güvenlik gereksinimlerinin ve/veya bu sprintte belirlenen güvenlik gereksinimlerinin risk analizi yapılarak güvenlik sprintinin iş listesi belirlenir.

5.2.3 Güvenli Tasarım

Değerler, tehditler ve olası sistem açıkları belirlendikten sonra yazılımın atağa açık alanlarını en aza indirgeyecek şekilde tasarım yapılır. Howard ve Lipner'e göre atak alanı kod parçası, arayüz, servis, protokol (iletişim kuralı) ve özellikle yetkisiz kullanıcılar olmak üzere tüm kullanıcılara açık olan uygulamalar olarak tanımlanmıştır [56]. Çoğu yazılım geliştiriciler, yazılım güvenliği konusunda bir uzmanın tecrübesine sahip olmayabilir ancak bu boşluğu kapatmak için yazılımda güvenliği sağlama adına gerçek deneyimlerden yola çıkılarak oluşturulan pratikleri sunan ilkeler, kılavuzlar ve kodlama kuralları gibi kaynaklardan ve Scrum takımına eşlik eden güvenlik uzmanından yararlanılabilir. Ayrıca, talep yönetim sistemine girilmiş ve kapanmış olan daha önceki benzer güvenlik gereksinimlerinden yararlanılabilir. Bkz. "4.4 Güvenli Tasarım".

5.2.4 Güvenli Gerçekleştirim

Güvenli kodlama ve test ilkelerine göre gerçekleştirim yapılır. Statik analiz araçları ile kod taranır. Kod gözden geçirmelerle, araçların bulamadığı hatalar bulunur. Bkz. "4.5 Güvenli Kodlama ve Test".

5.2.5 Güvenlik Testi

Yazılım güvenlik testi gerçekleştirilir. Bkz. “4.6 Yazılım Güvenlik Testi”. Güvenlik testi aşağıdaki iki yaklaşımı içermelidir [35]:

1) Güvenlik mekanizması fonksiyonlarının doğru gerçekleştirildiğinin testi,

2) Saldırganın bakış açısıyla bakıp sistem açıklarını göz önünde bulundurarak yapılan riske dayalı güvenlik testi.

5.2.6 Güvenlik Doğrulaması

Bulandırma araçları ile yazılıma geçersiz girdiler verilip test edilir. Kod ve tasarım güvenlik gözden geçirmeleri gerçekleştirilir.

NOT: Sprint boyunca yeni gereksinimler oluştukça Ürün İş Listesine eklenir. Burada tüm güvenlik gereksinimlerinin ayrı bir güvenlik iş listesi yerine varolan Ürün İş Listesine eklenmesi önerilmiştir. Çünkü bazı güvenlik gereksinimlerinde alınan tasarım kararları diğer standart taleplerin geliştirimini etkileyebilir. Bu durumda güvenlik gereksiniminin etkilediği standart taleplere bağlantı verilebilir.

5.3 Kod Yeniden Yapılandırma

Kodu okuması ve devam ettirmesi daha kolay, daha kaliteli ve hatta daha güvenli yapmak için iç gösterimini iyileştirmektir. Güvenlik gereksinimi olarak, eski kodlar gözden geçirilir, güvenlik hataları aranır, eğer bulunursa düzeltilir. Kod yeniden yapılandırma işlemi, “Scrum Master” veya “Geliştirme Takımı” tarafından güvenlik sprintine veya standart sprintlere gerek duyuldukça kalem olarak eklenebilir. Sahaya sürülmeden önce kodun güvenlik açısından gözden geçirilip yeniden yapılandırılması güvenliği sağlama konusunda oldukça fayda sağlar.

5.4 Kod ve Tasarım Güvenlik Gözden Geçirmeleri

Ürünün sahaya sürülmeye hazır olduğunun onaylanması açısından, kodun ve tasarımın güvenlik açısından gözden geçirilmesi gerekir. Gerektiğinde güvenlik uzmanı dâhil edilebilir ve kod yeniden yapılandırma ile birlikte yürütülebilir. “Sprint Değerlendirmesi” sonucu gerek duyuldukça “Scrum Master” veya “Geliştirme Takımı” tarafından “Sprint Planlamaları”nda güvenlik sprintine veya standart sprintlere kalem olarak eklenebilir.

5.5 Güvenlik Eğitimi

Belli bir plana göre geliştirme takımına uzman tarafından güvenlik eğitimi ve/veya yazılım güvenliğiyle ilgili kitap, cd vb. materyal sağlanır. “Scrum Retrospektifi”nde yapılan değerlendirme ve gözlemlerden yararlanılarak, “Scrum Takımı”nın ihtiyaç duyduğu konulara göre “Scrum Master” tarafından planlaması yapılabilir. Bkz. “4.1 Güvenlik Eğitimi”.

6. MATERYAL ve YÖNTEM

Bu tez çalışması için uygulanan yöntem 5 adımdan oluşur.

İlk adımda, yazılım güvenliği tanımı ve kapsamını anlamak amaçlanmıştır. Yazılımda güvenliği hedefleyen uygulamalar üzerine literatür araştırması yapılmıştır.

İkinci adımda, yazılımlar için referans model niteliği taşıyan olgunluk modelleri ve güvenli yazılım geliştirme modelleri güvenlik bakış açısıyla incelenmiştir. Bu amaçla, yazılım için olgunluk modellerinden CMMI, FAA-ICMM, SSE-CMM ve SAMM modeli incelenmiştir. Bu modellerden sadece SSE-CMM ve SAMM modellerinin doğrudan güvenliği vurgulayan uygulamalar seti sunduğu görülmüştür. Güvenli yazılım geliştirme modellerinden Microsoft SDL, TSP ve Correctness by Construction incelenmiştir. Bu modellerle geliştirilen projelerin başarılı sonuçlar verdiği görülmüştür. Bunu yapmaktaki amaç standartların yazılım güvenliğine bakışını anlamak, sunduğu uygulamaları analiz etmek ve güvenli yazılım geliştirme modellerinin süreç içerisinde bu uygulamaları nasıl faaliyete geçirdiğini anlamaktır.

Üçüncü adımda, Scrum çerçevesi ve bu çerçevenin benimsediği çevik yaklaşım üzerine araştırma yapılmıştır. Güvenlik uygulamalarının çevik yaklaşımın ilkeleriyle uymayan kısımları analiz edilmiştir.

Dördüncü adımda Scrumın temelindeki ilkeler ve yazılım güvenliği ilkeleri korunarak, güvenliği hedefleyen uygulamaların, Scrum çerçevesine entegrasyonuna çalışılmıştır.

Son adımda önerilen modelin uygulanabilirliğinin test edilmesi amacıyla sahada çalışan yazılım geliştiricilerin görüşlerine başvurularak güvenlik uygulamalarının çevik yazılım geliştirme ile çakışan ve uyumlu tarafları belirlenmeye çalışılmıştır.

Verilerin Toplanması

Yazılım güvenliği uygulamalarının, çevik yöntemlerde beraber uygulanabilirliğini ve önerilen modelin uygulamalarının uygulanabilirliğinin değişimini analiz etmek amacıyla anket çalışması yapılmıştır. İlk anket, literatür araştırması sonucu belirlenen 11 güvenlik uygulamasının 7 çevik ilkeyle olan uyumluğunu, ikinci anket ise bu tez kapsamında önerilen Trustworthy Scrum modeline göre, bu 11 güvenlik uygulamasının 7 çevik ilkeyle olan uyumluluğunu ölçmeye yönelik hazırlanmıştır. Anketlerde katılımcılardan her bir güvenlik uygulamasının 7 çevik ilkeyle olan uygulanabilirliğinin 3 seviye (0: uyumsuz, 1: kısmen uyumlu, 2: uyumlu) ile derecelendirilmesi istenmiştir.

Veriler, bilgisayar yazılımı sağlayan özel bir şirketten çoğunluğunun yazılım geliştirici ve yazılım mühendisi pozisyonuna sahip olduğu 10 gönüllü katılımcıdan sağlanmıştır. Katılımcılardan güvenlik uygulamaları konusunda bilgi sahibi olanların sayısı 8 ve çevik yöntem konusunda tecrübe sahibi olanların sayısı 9'dur. Katılımcıların, demografik özellikleri çizelge 6.1 de verilmiştir.

Çizelge 6.1. Katılımcı demografik bilgi

Yaş aralığı	**Eğitim**	**Üniversite Bölüm**	**Mesleki Deneyim (Yıl)**
• 21-30 (6 kişi) • 31-40 (4 kişi)	• Lisans (4 kişi) • Yüksek lisans (6 kişi)	• Bilgisayar Mühendisliği (8 kişi) • Bilgisayar Teknolojisi ve Bilişim Sistemleri (2 kişi)	• 1-5 yıl (4 kişi) • 6-10 yıl (4 kişi) • 11-15 yıl (2 kişi)

Ankette yer alan 11 güvenlik uygulaması aşağıda verilmiştir:

- G1: Güvenlik Eğitimi
- G2: Güvenlik Uzmanı
- G3: Güvenlik Gereksinimleri
- G4: Risk Analizi
- G5: Güvenli Tasarım
- G6: Güvenli Kodlama Standartları
- G7: Statik Analiz Araçları
- G8: Kod ve Tasarım Gözden Geçirme
- G9: Kod Yeniden Yapılandırma
- G10: Güvenlik Testi
- G11: Güvenlik Dokümanları

Ankette yer alan 7 çevik yöntem ilkesi aşağıda verilmiştir:

- C1: Erken ve devamlı teslim edilebilir yazılım
- C2: Değişen gereksinimlere hızlı adaptasyon
- C3: Kısa geliştirme iterasyonları
- C4: Ön aşama olarak minimal olan ve artırımsal gelişen tasarım

- C5: Doğrudan iletişim ve minimal dokümantasyon
- C6: Sadelik, yapılmasına gerek olmayan işlerin yapılmaması
- C7: Teknik ve iyi tasarım konusunda özen

Her bir güvenlik uygulaması için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmıştır. Wilcoxon T testi, eşleştirilmiş gruplara ilişkin farklılıkların boyutlarını da dikkate alarak iki değişkene ait dağılımın aynı olup olmadığını test etmek amacıyla geliştirilmiş bir analiz yöntemidir [59].

Ön test, Trustworthy Scrum model önerisi yapılmadan önce yukarıdaki 11 güvenlik uygulamasının, çevik yöntemlerin 7 ilkesiyle olan uyumluluğunun, Son test ise Trustworthy Scrum modeline göre 11 güvenlik uygulamasının 7 çevik yöntem ilkesiyle olan uyumluluğunun analizini ifade etmektedir.

7. BULGULAR

Güvenlik eğitimi (G1) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.1'de gösterilmiştir.

Çizelge 7.1. G1 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G1C1	10	1,10	0,88	0	2	G1C1	10	1,50	0,53	1	2	-1.414	,157
G1C2	10	1,00	0,67	0	2	G1C2	10	1,50	0,71	0	2	-1.667	,096
G1C3	10	1,20	0,63	0	2	G1C3	10	1,50	0,53	1	2	-1.342	,180
G1C4	10	1,10	0,88	0	2	G1C4	10	1,50	0,53	1	2	-1.633	,102
G1C5	10	,40	0,70	0	2	G1C5	10	1,30	0,95	0	2	-2.251	**,024**
G1C6	10	1,00	0,67	0	2	G1C6	10	1,30	0,48	1	2	-1.732	,083
G1C7	10	1,60	0,70	0	2	G1C7	10	1,80	0,42	1	2	-.816	,414

Çizelge 7.1 incelendiğinde, güvenlik uygulamalarından Güvenlik eğitiminin (G1), C5 için son test skorlarının ön teste göre artığı ($Z=-2.251$, $p<0.05$), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Güvenlik eğitimi uygulamasının çakışmaya neden olduğu söylenebilir.

Güvenlik uzmanı (G2) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.2'de gösterilmiştir.

Çizelge 7.2. G2 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G2C1	10	,70	0,67	0	2	G2C1	10	1,30	0,48	1	2	-1.857	,063
G2C2	10	,50	0,71	0	2	G2C2	10	1,50	0,85	0	2	-2.271	**,023**
G2C3	10	,90	0,74	0	2	G2C3	10	1,50	0,71	0	2	-2.121	**,034**
G2C4	10	,60	0,70	0	2	G2C4	10	1,30	0,67	0	2	-2.070	**,038**
G2C5	10	1,00	0,82	0	2	G2C5	10	1,40	0,70	0	2	-1.414	,157
G2C6	10	,60	0,70	0	2	G2C6	10	1,20	0,79	0	2	-2.121	**,034**
G2C7	10	1,20	0,79	0	2	G2C7	10	1,90	0,32	1	2	-2.333	**,020**

Çizelge 7.2 incelendiğinde, güvenlik uygulamalarından Güvenlik uzmanının, C2, C3, C4, C6 ve C7 için son test skorlarının ön teste göre artığı (C2 için Z=-2.271, $p<0.05$; C3 için Z=-2.121, $p<0.05$; C4 için Z=-2.070, $p<0.05$; C6 için Z=-2.121, $p<0.05$ ve C7 için Z=-2.333, $p<0.05$) ortaya çıkmıştır. Güvenlik uzmanı, çevik yazılım geliştirme ilkelerinin çoğunluğuyla uyumlu olduğu için, Çevik yazılım geliştirmede bu uygulamanın çakışmaya neden olmadığı söylenebilir.

Güvenlik gereksinimleri (G3) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.3'de gösterilmiştir.

Çizelge 7.3. G3 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G3C1	10	1,00	0,67	0	2	G3C1	10	1,50	0,53	1	2	-2.236	**,025**
G3C2	10	1,00	0,82	0	2	G3C2	10	1,30	0,82	0	2	-1.342	,180
G3C3	10	1,10	0,57	0	2	G3C3	10	1,40	0,70	0	2	-1.342	,180
G3C4	10	,80	0,79	0	2	G3C4	10	1,60	0,70	0	2	-2.530	**,011**
G3C5	10	,80	0,79	0	2	G3C5	10	1,70	0,48	1	2	-2.251	**,024**
G3C6	10	,90	0,88	0	2	G3C6	10	1,20	0,63	0	2	-1.342	,180
G3C7	10	1,10	0,74	0	2	G3C7	10	1,70	0,48	1	2	-2.121	**,034**

Çizelge 7.3 incelendiğinde, güvenlik uygulamalarından Güvenlik gereksiniminin, C1, C4, C5 ve C7 için son test skorlarının ön teste göre artığı (C1 için Z=-2.236, p<0.05; C4 için Z=-2.530, p<0.05; C5 için Z=-2.251 ve C7 için Z=-2.121, p<0.05) ortaya çıkmıştır. Güvenlik gereksinimi, çevik yazılım geliştirme ilkelerinin çoğunluğuyla uyumlu olduğu için, Çevik yazılım geliştirmede bu uygulamanın çakışmaya neden olmadığı söylenebilir.

Risk analizi (G4) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.4'de gösterilmiştir.

Çizelge 7.4. G4 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G4C1	10	1,30	0,67	0	2	G4C1	10	1,40	0,70	0	2	-.447	,655
G4C2	10	,90	0,74	0	2	G4C2	10	1,40	0,70	0	2	-2.236	**,025**
G4C3	10	,90	0,88	0	2	G4C3	10	1,00	0,67	0	2	-.333	,739
G4C4	10	1,10	0,74	0	2	G4C4	10	1,70	0,48	1	2	-2.449	**,014**
G4C5	10	,80	0,79	0	2	G4C5	10	1,10	0,74	0	2	-.756	,450
G4C6	10	1,00	0,94	0	2	G4C6	10	1,30	0,82	0	2	-1.342	,180
G4C7	10	1,30	0,82	0	2	G4C7	10	1,70	0,48	1	2	-1.414	,157

Çizelge 7.4 incelendiğinde, güvenlik uygulamalarından Risk analizinin, C2 ve C4 için son test skorlarının ön teste göre artığı (C2 için Z=-2.236, p<0.05 ve C4 için Z=-2.449, p<0.05), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Risk analizi uygulamasının çakışmaya neden olabileceği söylenebilir.

Güvenli tasarım (G5) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.5'de gösterilmiştir.

Çizelge 7.5. G5 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G5C1	10	1,30	,67	0	2	G5C1	10	1,40	,70	0	2	-.577	,564
G5C2	10	1,10	,74	0	2	G5C2	10	1,20	,79	0	2	-.577	,564
G5C3	10	1,20	,79	0	2	G5C3	10	1,60	,52	1	2	-1.414	,157
G5C4	10	1,00	,94	0	2	G5C4	10	1,50	,71	0	2	-1.890	,059
G5C5	10	1,10	,88	0	2	G5C5	10	1,30	,82	0	2	-.557	,577
G5C6	10	1,10	,88	0	2	G5C6	10	1,60	,52	1	2	-2.236	**,025**
G5C7	10	1,50	,71	0	2	G5C7	10	1,60	,52	1	2	-.577	,564

Çizelge 7.5 incelendiğinde, güvenlik uygulamalarından Güvenli tasarımın, C6 için son test skorlarının ön teste göre artığı ($Z=-2.236$, $p<0.05$), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Güvenli tasarım uygulamasının çakışmaya neden olduğu söylenebilir.

Güvenli kodlama standartları (G6) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.6'da gösterilmiştir.

Çizelge 7.6. G6 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G6C1	10	1,30	0,67	0	2	G6C1	10	1,70	0,48	1	2	-2.000	**,046**
G6C2	10	1,40	0,84	0	2	G6C2	10	1,70	0,67	0	2	-1.342	,180
G6C3	10	1,30	0,67	0	2	G6C3	10	1,80	0,42	1	2	-2.236	**,025**
G6C4	10	,90	0,88	0	2	G6C4	10	1,50	0,71	0	2	-1.857	,063
G6C5	10	1,20	0,79	0	2	G6C5	10	1,40	0,70	0	2	-1.414	,157
G6C6	10	1,10	0,74	0	2	G6C6	10	1,30	0,82	0	2	-.632	,527
G6C7	10	1,40	0,70	0	2	G6C7	10	1,60	0,70	0	2	-1.000	,317

Çizelge 7.6 incelendiğinde, güvenlik uygulamalarından Güvenli kodlama standartlarının, C1 ve C3 için son test skorlarının ön teste göre artığı (C1 için Z=-2.000, p<0.05 ve C3 için Z=-2.236, p<0.05), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Güvenli kodlama standartları uygulamasının çakışmaya neden olabileceği söylenebilir.

Statik analiz araçları (G7) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.7'de gösterilmiştir.

Çizelge 7.7. G7 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G7C1	10	1,00	0,82	0	2	G7C1	10	1,30	0,67	0	2	-1.732	,083
G7C2	10	1,00	0,82	0	2	G7C2	10	1,70	0,67	0	2	-2.070	**,038**
G7C3	10	1,20	0,63	0	2	G7C3	10	1,30	0,82	0	2	-.378	,705
G7C4	10	,90	0,88	0	2	G7C4	10	1,20	0,79	0	2	-1.342	,180
G7C5	10	1,30	0,48	1	2	G7C5	10	1,40	0,70	0	2	-.577	,564
G7C6	10	1,10	0,99	0	2	G7C6	10	1,90	0,32	1	2	-2.070	**,038**
G7C7	10	1,30	0,82	0	2	G7C7	10	1,60	0,70	0	2	-1.732	,083

Çizelge 7.7 incelendiğinde, güvenlik uygulamalarından Statik analiz araçlarının, C2 ve C6 için son test skorlarının ön teste göre artığı (C2 için Z=-2.070, p<0.05 ve C6 için Z=-2.070, p<0.05), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Statik analiz araçları uygulamasının çakışmaya neden olabileceği söylenebilir.

Kod ve tasarım gözden geçirme (G8) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.8'de gösterilmiştir.

Çizelge 7.8. G8 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G8C1	10	1,00	0,82	0	2	G8C1	10	1,60	0,52	1	2	-2.121	**,034**
G8C2	10	1,00	0,82	0	2	G8C2	10	1,20	0,79	0	2	-1.414	,157
G8C3	10	1,40	0,70	0	2	G8C3	10	1,60	0,70	0	2	-.541	,589
G8C4	10	,50	0,71	0	2	G8C4	10	1,30	0,67	0	2	-2.271	**,023**
G8C5	10	1,10	0,88	0	2	G8C5	10	1,30	0,82	0	2	-.557	,577
G8C6	10	1,10	0,88	0	2	G8C6	10	1,40	0,70	0	2	-1.000	,317
G8C7	10	1,20	0,79	0	2	G8C7	10	1,50	0,71	0	2	-1.342	,180

Çizelge 7.8 incelendiğinde, güvenlik uygulamalarından Kod ve tasarım gözden geçirmenin, C1 ve C4 için son test skorlarının ön teste göre artığı (C1 için $Z=-2.121$, $p<0.05$ ve C4 için $Z=-2.271$, $p<0.05$), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Kod ve tasarım gözden geçirme uygulamasının çakışmaya neden olabileceği söylenebilir.

Kod yeniden yapılandırma (G9) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.9'da gösterilmiştir.

Çizelge 7.9. G9 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G9C1	10	,70	0,67	0	2	G9C1	10	1,40	0,70	0	2	-1.890	,059
G9C2	10	,90	0,88	0	2	G9C2	10	1,40	0,70	0	2	-1.633	,102
G9C3	10	,80	0,63	0	2	G9C3	10	1,40	0,52	1	2	-1.897	,058
G9C4	10	,60	0,84	0	2	G9C4	10	1,50	0,71	0	2	-2.460	**,014**
G9C5	10	,80	0,79	0	2	G9C5	10	1,40	0,84	0	2	-1.387	,165
G9C6	10	,70	0,67	0	2	G9C6	10	1,70	0,48	1	2	-2.428	**,015**
G9C7	10	1,10	0,74	0	2	G9C7	10	1,60	0,52	1	2	-1.667	,096

Çizelge 7.9 incelendiğinde, güvenlik uygulamalarından Kod yeniden yapılandırmanın, C4 ve C6 için son test skorlarının ön teste göre artığı (C4 için Z=-2.460, p<0.05 ve C6 için Z=-2.428, p<0.05), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Kod yeniden yapılandırma uygulamasının çakışmaya neden olabileceği söylenebilir.

Güvenlik testi (G10) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.10'da gösterilmiştir.

Çizelge 7.10. G10 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G10C1	10	1,10	0,57	0	2	G10C1	10	1,50	0,71	0	2	-1.633	,102
G10C2	10	,80	0,63	0	2	G10C2	10	1,20	0,92	0	2	-1.190	,234
G10C3	10	,40	0,70	0	2	G10C3	10	1,30	0,82	0	2	-2.251	**,024**
G10C4	10	,90	0,74	0	2	G10C4	10	1,20	0,79	0	2	-1.342	,180
G10C5	10	,60	0,84	0	2	G10C5	10	1,30	0,82	0	2	-1.933	,053
G10C6	10	1,00	0,67	0	2	G10C6	10	1,40	0,52	1	2	-2.000	**,046**
G10C7	10	1,20	0,63	0	2	G10C7	10	1,80	0,42	1	2	-2.121	**,034**

Çizelge 7.10 incelendiğinde, güvenlik uygulamalarından Güvenlik testinin, C3, C6 ve C7 için son test skorlarının ön teste göre artığı (C3 için Z=-2.251, p<0.05; C6 için Z=-2.000, p<0.05 ve C7 için Z=-2.121, p<0.05), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Güvenlik testi uygulamasının çakışma derecesinin önemli derecede azaldığı söylenebilir.

Güvenlik dokümanları (G11) için Çevik yazılım geliştirme uyumluluğunun nasıl değiştiğini göstermek için Wilcoxon T testi uygulanmış ve sonuçları Çizelge 7.11'de gösterilmiştir.

Çizelge 7.11. G11 için Çevik yazılım geliştirme uyumluluğu, Wilcoxon T testi

Ön Test						Son Test						Z	Asymp. Sig. (2-tailed)
Maddeler	N	Ort	SS	Min	Max	Maddeler	N	Ort	SS	Min	Max		
G11C1	10	1,00	0,67	0	2	G11C1	10	1,40	0,70	0	2	-1.414	,157
G11C2	10	,70	0,82	0	2	G11C2	10	1,10	0,88	0	2	-1.633	,102
G11C3	10	,60	0,70	0	2	G11C3	10	1,50	0,85	0	2	-2.165	**,030**
G11C4	10	1,20	0,63	0	2	G11C4	10	1,50	0,71	0	2	-1.134	,257
G11C5	10	,90	0,88	0	2	G11C5	10	1,30	0,82	0	2	-.966	,334
G11C6	10	,70	0,82	0	2	G11C6	10	1,40	0,70	0	2	-2.333	**,020**
G11C7	10	1,10	0,88	0	2	G11C7	10	1,60	0,52	1	2	-1.890	,059

Çizelge 7.11 incelendiğinde, güvenlik uygulamalarından Güvenlik dokümanlarının, C3 ve C6 için son test skorlarının ön teste göre artığı (C3 için $Z=-2.165$, $p<0.05$ ve C6 için $Z=-2.333$, $p<0.05$), ancak diğer çevik yazılım geliştirme maddeleri için istatistiksel olarak anlamlı bir şekilde artış oluşmadığı ortaya çıkmıştır. Buradan Çevik yazılım geliştirmede Güvenlik dokümanları uygulamasının çakışmaya neden olabileceği söylenebilir.

8. TARTIŞMA VE SONUÇLAR

Anket sonuçlarına göre model önerisinden önceki ön test ve Trustworthy Scrum önerisinden sonraki son teste göre güvenlik uygulamaları ile çevik ilkelerin uyumluluk durumları çizelge 8.1 ve çizelge 8.2'de verilmiştir.

Çizelge 8.1. Güvenlik uygulamaları ile çevik ilkelerin uyumluluğu

	C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
G1	1,10	1,00	1,20	1,10	,40	1,00	1,60
G2	,70	,50	,90	,60	1,00	,60	1,20
G3	1,00	1,00	1,10	,80	,80	,90	1,10
G4	1,30	,90	,90	1,10	,80	1,00	1,30
G5	1,30	1,10	1,20	1,00	1,10	1,10	1,50
G6	1,30	1,40	1,30	,90	1,20	1,10	1,40
G7	1,00	1,00	1,20	,90	1,30	1,10	1,30
G8	1,00	1,00	1,40	,50	1,10	1,10	1,20
G9	,70	,90	,80	,60	,80	,70	1,10
G10	1,10	,80	,40	,90	,60	1,00	1,20
G11	1,00	,70	,60	1,20	,90	,70	1,10

Çizelge 8.2. Trustworthy Scrum modeline göre güvenlik uygulamaları ile çevik ilkelerin uyumluluğu

	C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**Uyumluluk %**
G1	1,50	1,50	1,50	1,50	**1,30**	1,30	1,80	%15
G2	1,30	**1,50**	**1,50**	**1,30**	1,40	**1,20**	**1,90**	%70
G3	**1,50**	1,30	1,40	**1,60**	**1,70**	1,20	**1,70**	%60
G4	1,40	**1,40**	1,00	**1,70**	1,10	1,30	1,70	%30
G5	1,40	1,20	1,60	1,50	1,30	**1,60**	1,60	%15
G6	**1,70**	1,70	**1,80**	1,50	1,40	1,30	1,60	%30
G7	1,30	**1,70**	1,30	1,20	1,40	**1,90**	1,60	%30
G8	**1,60**	1,20	1,60	**1,30**	1,30	1,40	1,50	%30
G9	1,40	1,40	1,40	**1,50**	1,40	**1,70**	1,60	%30
G10	1,50	1,20	**1,30**	1,20	1,30	**1,40**	**1,80**	%45
G11	1,40	1,10	**1,50**	1,50	1,30	**1,40**	1,60	%30

Çizelge 8.2 incelendiğinde, önerilen modelin (Trustworthy Scrum) uyumluluk dereceleri tüm hücrelerde artmakla beraber, koyu olarak gösterilenler istatistiksel olarak anlamlı bir şekilde artışın gözlendiği kısımlardır. Uyumluluk sütunu ise, her güvenlik uygulaması için tüm hücrelerde Trustworthy Scrum lehine bir artış söz konusuyken, sadece istatiksel anlamda artış olan çevik ilkelerin oranı hesaplanarak bulunmuştur. Diğer bir deyişle, istatistiki sonuçlara göre Trustworthy Scrum modelinde her bir güvenlik uygulamasının çevik yöntemlerle uyum derecesini göstermektedir.

Yazılımlarda güvenlik açıklarının çok yaygın olmasına rağmen, yazılım geliştirme modelleri güvenliği sağlamak için gerekli olan uygulamaları içermemektedir. Güvenlik konusunu irdeleyen az sayıda yazılım geliştirme modeli olmakla beraber, bu modeller daha

çok geleneksel güvenlik tekniklerinin şelale süreçlere eklenmesiyle oluşturulmuştur. Çevik modellerde ise güvenliğin yazılım geliştirmeyi aksattığı ve yavaşlattığı için çevik ilkelere uymayacağı kanısıyla gözardı edilmektedir. Bu çalışmada yazılım güvenliği ile ilgili literatür çalışmalarının ve güvenli yazılım geliştirme modellerinin incelenmesi sonucu, güvenliği hedefleyen uygulamalar belirlenmiş ve hem güvenlik ilkeleri hem de çevik yöntem ilkeleri göz önünde bulundurularak, bu güvenlik uygulamalarının Scrum çerçevesi içinde uygulanmasını amaçlayan bir güvenli yazılım geliştirme modeli önerisi yapılmıştır. Önerilen modelin uygulanabilirliğinin test edilmesi amacıyla sahada çalışan yazılım geliştiricilerin görüşlerine başvurularak güvenlik uygulamalarının çevik yazılım geliştirme ile çakışan ve uyumlu tarafları belirlenmeye çalışılmıştır ve önerilen modelde bu çakışmalarda azalma meydana geldiği görülmüştür. Gelecek çalışma olarak, önerilen model, geliştirilen yazılımın kritiklik seviyesine göre içermesi gereken uygulamalar ve dereceleri bakımından seviyelere ayrılabilir veya belli bir sektör hedeflenerek daha ayrıntılı ve sektöre özgü uygulamalar sunulabilir.

KAYNAKLAR

[1] Standish Group, *The Chaos Report*, West Yarmouth, MA: The Standish Group, **1995**.

[2] Viega, J., McGraw, G. R., *Building Secure Software: How to Avoid Security Problems the Right Way*, Portable Documents, Pearson Education, **2001**.

[3] Beznosov, K., Kruchten, P., Towards agile security assurance, In *Proceedings of the 2004 workshop on New security paradigms* (pp. 47-54), ACM, **2004**.

[4] The Standish Group Report, CHAOS Report **2015**, http://www.standishgroup.com (Kasım, **2016**).

[5] McGraw, G., From the ground up: The DIMACS software security workshop, *IEEE Security & Privacy*, 99(2), 59-66, **2003**.

[6] Redwine, T. S., Davis, N., Processes to produce secure software, *National Cyber Security Summit-USA*, **2004**.

[7] McGraw, G., Software security, *IEEE Security & Privacy*, 2(2), 80-83. **2004.**

[8] Microsoft MSDN, The Trustworthy Computing Security Development Lifecycle, https://msdn.microsoft.com/en-us/library/ms995349.aspx (Kasım, **2016**).

[9] Howard, M., Lipner, S., *The Security Development Lifecycle: A Process for Developing Demonstrably More Secure Software*, Microsoft Press, **2006**.

[10] SDL for Agile Development, Microsoft MSDN, https://msdn.microsoft.com/en-us/library/windows/desktop/ee790617.aspx (Kasım, **2016**).

[11] Humphrey, W. S., *The Team Software Process (TSP), Carnegie Mellon University*, Software Engineering Institute, **2000**.

[12] Bartsch, S., Practitioners' perspectives on security in agile development, In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference* on (pp. 479-484), IEEE, **2011**.

[13] *Securosis*, Secure Agile Development, **2014** https://securosis.com/assets/library/reports/SecureAgileDevelopment_Nov2014_FINAL.pdf.

[14] Zurko, M. E., Simon, R. T., User-centered security, In *Proceedings of the 1996 workshop on New security paradigms* (pp. 27-33), ACM, **1996**.

[15] Committee on National Security Systems, *National Information Assurance Glossary (CNSS Instruction No. 4009)*, June **2006**.

[16] NASA Software Assurance Technology Center, *Software Assurance Guidebook, NASA-GB-A201*, **1990**, https://ntrs.nasa.gov/search.jsp?R=19980228457.

[17] Davis, N., *Secure software development life cycle processes: A technology scouting report* (No. CMU/SEI-2005-TN-024), Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, **2005**.

[18] Kitson, D. H., A Tailoring of the CMM for the Trusted Software Domain, In *Proceedings of the Seventh Annual Software Technology Conference* (pp. 9-14), **1995**.

[19] Davis, N., Mullaney, J. L., *The Team Software Process in Practice: A Summary of Recent Results*, Technical Report CMU/SEI-2003-TR-014, September **2003**.

[20] Jones, C., *Software assessments, benchmarks, and best practices,* Addison-Wesley Longman Publishing Co., Inc., **2000**.

[21] Davis, N., Developing secure software, *Software Tech News*, 8(2), 3-7, **2005**.

[22] Hall, A., Chapman, R., Correctness by construction: Developing a commercial secure system, *IEEE software*, 19(1), 18-25, **2002**.

[23] Ross, Philip E., The Exterminators: A Small British Firm Shows That Software Bugs Aren't Inevitable, *IEEE Spectrum* 42, 9, 36-41, **2005**.

[24] US-CERT Build Security In, Correctness by Construction, https://www.us-cert.gov/bsi/articles/knowledge/sdlc-process/correctness-by-construction (Aralık, **2016**)

[25] Pfleeger, S. L., Hatton, L., Investigating the influence of formal methods, *Computer*, 30(2), 33-43, **1997**.

[26] Agile Alliance, Manifesto for Agile Software Development, **2005**, https://www.agilealliance.org.

[27] Schwaber, K., Sutherland, J., The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game, **2016** Scrum.Org and ScrumInc. http://www.scrumguides.org.

[28] Over, J. W., Team Software Process for Secure Systems Development, SEI CMU Pittsburgh, PA, **2002**.

[29] Webb, D. R., Managing Risk With TSP, *Crosstalk The Journal of Defense Software Engineering*, June **2000**.

[30] Paulk, M., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Reading, MA: Addison Wesley, 1995.

[31] Wäyrynen, J., Bodén, M., Boström, G., Security engineering and eXtreme programming: An impossible marriage?, In *Conference on Extreme Programming and Agile Methods* (pp. 117-128), Springer Berlin Heidelberg, **2004**.

[32] Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., Kruchten, P., Extending XP practices to support security requirements engineering, In *Proceedings of the 2006 international workshop on Software engineering for secure systems* (pp. 11-18), ACM, **2006**.

[33] Mougouei, D., Sani, N. F. M., Almasi, M. M., S-scrum: a secure methodology for agile development of web services, *World of Computer Science and Information Technology Journal*, 3(1), 15-19, **2013**.

[34] Pohl, C., Hof, H. J., Secure Scrum: Development of Secure Software with Scrum, *arXiv preprint*, **2015**.

[35] Potter, B., McGraw, G., Software Security Testing, *IEEE Security and Privacy*, 2(5), 81-85.B, **2004**.

[36] G. Hoglund and G. McGraw, *Exploiting Software: How to Break Code, Addison-Wesley*, **2004**.

[37] Hope, P., Lavenhar, S., Peterson, G., Architectural Risk Analysis 38, *Build Security In*, **2005**.

[38] Azham, Z., Ghani, I., & Ithnin, N., Security backlog in Scrum security practices, In *Software Engineering (MySEC), 2011 5th Malaysian Conference* in (pp. 414-417), IEEE, **2011**.

[39] McGraw, G., Allen, J. H., Mead, N., Ellison, R. J., Barnum, S., *Software Security Engineering: A Guide for Project Managers*, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, **2013**.

[40] Hope, P., McGraw, G., Antón, A. I., Misuse and abuse cases: Getting past the positive, *IEEE Security & Privacy*, 2(3), 90-92, **2004**.

[41] Mead, N. R., Security requirements engineering, Carnegie Mellon University, **2008**.

[42] Mead, N. R., Allen, J. H., Identifying Software Security Requirements Early, Not After the Fact (audio), *InformIT*, **2008**, http://www.informit.com/.

[43] McDermott, J., Fox, C., Using abuse case models for security requirements analysis, *In Computer Security Applications Conference, (ACSAC'99) Proceedings, 15th Annual* (pp. 55-64), IEEE, **1999**.

[44] Ellison, Robert J. Moore, Andrew. P., *Trustworthy Refinement Through Intrusion-Aware Design* (CMU/SEI2003TR002, ADA414865). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, **2003**.

[45] Schneier, B., *Secrets and Lies: Digital Security in a Networked World*, John Wiley & Sons, **2000**.

[46] Boehm, B. W., Understanding and controlling software costs, *Journal of Parametrics*, 8(1), 32-68, **1988**.

[47] McConnel, S., From the Editor–An Ounce of Prevention, *IEEE Software*, 18(3), 5-7, **2001**.

[48] Jones, C., *Tutorial Programming Productivity: Issues for the Eighties, 2nd Ed. Los Angeles*, IEEE Computer Society Press, **1986**.

[49] Berinato, S., Finally, a real return on security spending, *Cio*, 15(9), 42-50, **2002.**

[50] Newman, M., Software errors cost US economy $59.5 billion annually, *NIST Assesses Technical Needs of Industry to Improve Software-Testing*, **2002**.

[51] Sindre, G., Opdahl, A. L., Eliciting security requirements with misuse cases, *Requirements engineering*, 10(1), 34-44, **2005**.

[52] Keeney, M., Kowalski, E., Cappelli, D., Moore, A., Shimeall, T., Rogers, S., *Insider threat study: Computer system sabotage in critical infrastructure sectors*, National Threat Assessment Ctr Washington Dc, **2005**.

[53] Swanson, M., Wohl, A., Pope, L., Grance, T., Hash, J., Thomas, R., *Contingency Planning Guide for Information Technology Systems (NIST Special Publication 800-34)*, **2002**.

[54] Verdon, D., McGraw, G., Risk analysis in software design, *IEEE Security & Privacy*, 2(4), 79-84, **2004**.

[55] Howard, M., LeBlanc, D., *Writing Secure Code*, Second Edition, Microsoft Press, **2002**.

[56] Howard, M., Attack Surface: Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users, *MSDN Magazine*, November **2004**, https://msdn.microsoft.com/magazine/msdn-magazine-issues.

[57] Saltzer, J. H., Schroeder, M. D., The Protection of Information in Computer Systems, *Proceedings of the IEEE*, 63(9), 1278-1308, **1975**.

[58] Breu, R., Burger, K., Hafner, M., Jürjens, J., Popp, G., Wimmel, G., Lotz, V., Key issues of a formally based process model for security engineering. In *International Conference on Software and Systems Engineering and their Applications*, **2003**.

[59] Ott, R. Lyman, and Micheal T. Longnecker, *An introduction to statistical methods and data analysis*, Nelson Education, **2015.**

[60] Mead, N. R., Viswanathan, V., Zhan, J., Incorporating security requirements engineering into standard lifecycle processes, *International Journal of Security and Its Applications*, 2(4), 67-79, **2008**.

[61] CMU Software Engineering Institute Vulnerability Notes Database, http://www.kb.cert.org/CERT_WEB/services/vul-notes.nsf (Kasım, **2016**).

[62] NIST, G. S., Goguen, A., Fringa, A., Risk Management Guide for Information Technology Systems (NIST 800-30), *Recommendations of the National Institute of Standards and Technology*, **2002,** http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf.

[63] McGraw, G., *Software security: building security in* (Vol. 1), Addison-Wesley Professional, **2006**.

[64] US-CERT Build Security In, Code Analysis, https://www.us-cert.gov/bsi/articles/best-practices/code-analysis/code-analysis (Aralık, **2016**).

ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı : Güler Koç

Doğum Yeri : Ankara

Medeni Hali : Bekar

E-posta : glrkoc@gmail.com

Adresi :

Eğitim

Lise : Çankaya Milli Piyango Anadolu Lisesi

Lisans : Ankara Üniversitesi, Bilgisayar Mühendisliği

Yüksek Lisans :

Doktora :

Yabancı Dil ve Düzeyi

İngilizce YDS 80

İş Deneyimi

2013 - 2014 Softtech, İstanbul Merkez Ofis

2014 - Halen TUBITAK SAGE

Deneyim Alanları

Tezden Üretilmiş Projeler ve Bütçesi

Tezden Üretilmiş Yayınlar

Tezden Üretilmiş Tebliğ ve/veya Poster Sunumu ile Katıldığı Toplantılar

Koç, G., Aydos, M., Trustworthy Scrum: Scrum ile Güvenli Yazılım Geliştirme, *Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı (UBMK 2017)* (bildiri olarak kabul edildi, 17.08.2017)

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS/DOKTORA TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 25/08/2017

Tez Başlığı / Konusu: YAZILIM GELİŞTİRME MODELLERİNİN GÜVENLİK AÇISINDAN ANALİZİ VE BİR GÜVENLİ YAZILIM GELİŞTİRME MODELİ ÖNERİSİ

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam 70 sayfalık kısmına ilişkin, 25/08/2017 tarihinde tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 7'dir.

Uygulanan filtrelemeler:
1- Kaynakça hariç
2- Alıntılar dâhil
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü Tez Çalışması Orjinallik Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

Tarih ve İmza

25.08.2017

Adı Soyadı:	GÜLER KOÇ
Öğrenci No:	N14225062
Anabilim Dalı:	BİLGİSAYAR MÜHENDİSLİĞİ
Programı:	BİLGİSAYAR MÜHENDİSLİĞİ
Statüsü:	☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Yrd. Doç. Dr. Murat AYDOS

(Unvan, Ad Soyad, İmza)