Ekstrem Programlama Tekniklerinin Güvenlik-Kritik Sistemlerin Yazılım Geliştirme Süreçlerinde Uygulanabilirliği

Cevahir Turgut[1] Serkan Nizam[2]

[1,2]Aydın Yazılım ve Elektronik Sanayi A.Ş. (AYESAŞ) Silikon Blok 1.Kat No:1
Teknokent-ODTÜ 06531 ANKARA

[1]e-posta: cevahirt@ayesas.com [2]e-posta: serkann@ayesas.com

Özetçe

Bu bildiride Ekstrem Programlama (XP – Extreme Programming) pratiklerinin güvenlik-kritik sistemlere yönelik yazılım geliştirme süreçlerinde uygulanabilirliğine yönelik değerlendirmeler ve uygulanabilir bulunan Ekstrem Programlama pratiklerinin pratik uygulamalarına yönelik tespitler anlatılmaktadır. Ayrıca bu bildiri ile AYESAŞ'ın RTCA DO-178B standardı uyarınca geliştirdiği güvenlik-kritik aviyonik yazılım projelerinde bazı XP Programlama pratiklerinin kısmi uygulamaları ve bu uygulamaların sonuçları tartışılmaktadır.

XP Programlama, Çevik Yazılım Geliştirme Yöntemleri (Agile Methodologies) arasında en çok ilgi çeken yöntemlerden birisidir. Sürekli değişen yazılım gereksinimlerinin geliştirme sürecindeki olumsuz etkileri azalttığı görüşü, XP Programlamanın önemli destek toplama noktalarının başında gelmektedir. Ancak yüksek güvenlik güvencesi gerektiren yazılım projeleri için Çevik Yöntemlerden çok Şelale (Waterfall) gibi plan güdümlü yazılım süreçleri uygulamaları öne çıkmaktadır. Bu bildiride güvenlik-kritik yazılımlarda temel olarak seçilmiş yöntemle beraber XP Programlama pratiklerinin uygulanabilirliği incelenmiştir.

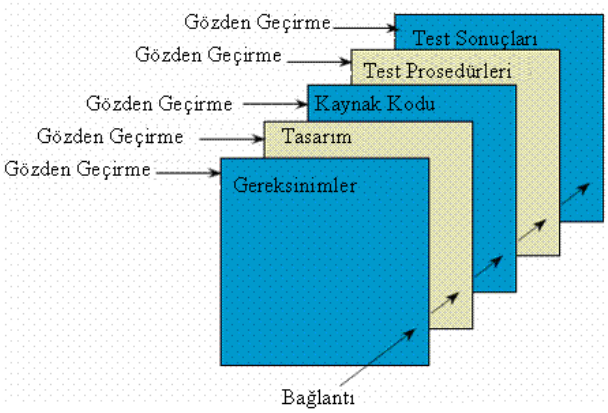
1. Giriş

Yazılım mühendisliğine yönelik ihtiyaçların giderilme çalışmaları günümüzde halen devam etmektedir. Bu kapsamda birçok farklı süreç modeli ve standart ortaya konulmuştur. Çevik Yöntemler son dönemde birçok alanda en çok ilgi çeken süreç yöntemlerindendir. Ancak gerek geliştirilen sistemin güvenlik seviyesinden, gerek kontratsal kısıtlardan, gerekse şirketlerin uyguladığı süreçlerden dolayı savunma ve havacılık sektörlerinde uygulanan yazılım geliştirme süreç yöntemleri geleneksel plan güdümlü yöntemler olarak kalmaya devam etmektedir.

Havacılık sektöründeki DO-178B standardına uygun olarak geliştirilen güvenlik kritik yazılımın yazılım gereksinimlerinin sürekli olarak değişmesi istenmemekle beraber değişen gereksinimlerin gerek zaman gerekse maliyet olarak yükü ağır olmaktadır. Havacılık sektöründeki güvenlik kritik yazılım gereksinimlerine cevap veren DO-178B [1] standardı uygulama maliyeti en yüksek standartlardan biridir. Şekil 1'de DO-178B'de doğrulama ve geçerleme aktivitelerinde yazılım birimleri arasında izlenebilirliği sağlamak amacıyla kurulması gereken izlenebilirlik verisinin yazılım ürünleri arasında nasıl kurulması gerektiği verilmiştir.

Bu izlenebilirlik verisi kullanılarak şekildeki her bir ürün için (yazılım gereksinimleri, tasarım, kaynak kod, test prosedürleri, test sonuçları) doğrulama yapılmaktadır. Bu doğrulama ve geçerleme aktivitelerinin yanında DO-178B, Gereksinim Kapsama Analizi (GKA), Yapısal Kapsama Analizi (YKA) ve Obje Kod Analizi (OKA) gibi temel analizlerin de yapılmasını şart koşmaktadır.

DO-178B Seviye (Level) A, B, C, D ve E olmak üzere 5 faklı güvenlik seviyesi için farklı hedefler ortaya koymaktadır. Örneğin ilk üç seviye için YKA yapılmasını istemektedir. Bu tür aktivitelerin maliyetleri artan güvenlik seviyesine göre artmakta olup değişen gereksinimlerle maliyetleri daha da artmaktadır.



Şekil 1 DO-178B'de Veri İzlenebilirliği

Şekil 1'de verilen veri izlenebilirlik şemasından da anlaşılacağı üzere, yazılım gereksinimlerindeki bir değişikliğin diğer tüm yazılım ürünlerine yansıması olacaktır. Şelale Yönteminde şekildeki tüm ürünler şekilde verilen sırayla üretilmektedir. Bu, Şelale Yönteminde yaşanabilecek olan olumsuzlukların DO-178B standardını uygulayan projelerde de yaşanabileceği anlamına gelmektedir. Örneğin, test aşamasında bulunacak ciddi bir hatanın maliyeti ağır olacaktır. Her ne kadar, her aşamadan sonra hataları azaltmak için gözden geçirme aktiviteleri uygulansa da eksik bilgi ya da gereksinim analizi ve tasarım aşamasında yanlış yapılmış bir varsayım nedeniyle hata bulunamayabilir. Yine belirtilen bir durum nedeniyle kodlama ve test aktiviteleri planlanandan fazla zaman ve iş gücü gerektirebilir. Bu tür durumlarda, uygulanabilir olan XP Programlama pratiklerini kullanmak gündeme gelebilir.

2. Güvenlik-Kritik Aviyonik Sistemlerde Yazılım Geliştirme

Güvenlik kritik aviyonik yazılımlar hatalı çalışmaları nedeniyle, insanların hayatlarını kaybetmesine ya da ağır yaralanmalarına ve/veya ağır maddi kayıplar verilmesine sebebiyet verebilecek yazılımlardır. Önceki bölümde de bahsedildiği üzere, bu yazılımların aviyonik sistemler için geliştirilmesinde uygulanılması gereken süreçler DO-178B standardında 5 farklı güvenlik kritiklik seviyesinde kategorize edilmişlerdir. İlk seviye Catastrophic (felaket) olarak isimlendirilmekte ve bu tür yazılımlardaki hatalar sürekli uçuş ve iniş güvenliğini tehlikeye atmaktadır. İkinci seviye Hazardous (tehlikeli) olarak isimlendirilmekte ve bu tür yazımlardaki hatalar hava aracının uçuş yeteneklerinin ve/veya kontrol kapasitesinin büyük ölçüde azalmasına, ciddi ve potansiyel yaralanmalara sebebiyet verebilmektedir. Üçüncü seviye Major (önemli) olarak isimlendirilmekte ve bu tür yazılımlardaki hatalar hava aracının uçuş yeteneklerinin ve/veya kontrol kapasitesinin önemli ölçüde azalmasına, potansiyel yaralanmalara sebebiyet verebilmektedir. Dördüncü seviye Minor (önemsiz) olarak isimlendirilmekte ve bu tür yazılımlardaki hatalar önemli kayıplara sebebiyet vermemektedir. Son seviyeyse No Effect (etkisiz) olarak isimlendirilmekte ve hava aracının hayati derecedeki işlevsel yeteneklerine (güvenli kalkış-iniş ve uçuş görevlerine) etkisi bulunmamaktadır [1]. Yazılımın güvenlik seviyesi belirlenirken hava aracının donanımından fonksiyonel tasarımına kadar tüm etkenler dikkate alınır. Yazılım biriminin güvenlik seviyesi yazılımın hangi işlevi gerçekleştireceği, hangi donanımda çalışacağı, yazılımın üzerinde çalışacağı hava aracının tipine ve büyüklüğüne göre değişkenlik gösterir. Örneğin tek veya iki kişilik bir uçak için ikinci seviye seçilebilirken 10 kişiden fazla kişi taşıyan bir helikopter ya da büyük bir yolcu uçağı için birinci seviye seçilebilir. Yine hava aracının kontrolünde ve temel kabiliyetlerinde etkisi olmayan bir yazılım birimi için üçüncü seviye seçilebilir.

Güvenlik seviyelerine göre maliyetteki değişimin temel unsuru güvenlik seviyesine göre yazılımın istenilen ölçütleri karşılamasıdır (dolaylı olarak yazılım kalite beklentisindeki artıştır). Örneğin üçüncü seviyede YKA faaliyetinde İfade Kapsama (Statement Coverage) aranırken birinci seviyede ifade kapsamanın yanında Değiştirilmiş Koşul/Karar Kapsama (MC/DC - Modified Condition/Decision Coverage) da aranmaktadır. Kapsama türlerinden kısaca bahsetmek gerekirse İfade Kapsamanın sağlanabilmesi için, yazılımdaki çalışabilir her satır kodun gereksinim tabanlı testler tarafından en azından bir kez çalıştırılmış olması gerekmektedir. Karar Kapsamanın sağlanabilmesi içinse bir kontrol bloğunun sonucunun hem doğru hem de yanlış olduğu durumlar için çalıştırılmış olması gerekir. Koşul Kapsamada bir karar yapısı içindeki tüm koşulların en az bir kez tüm olası değerleri alması gerekir. Karar/Koşul kapsama, karar ve koşul kapsamaların birleşimidir ve yazılımdaki tüm koşullarla tüm kararlar çalıştırılmış olmalıdır. MC/DC'deyse her bir koşulun diğer koşullara göreceli olarak testi gereklidir. Sadece İfade Kapsamayı sağlamak için yazılmış bir kod parçacığı MC/DC'yi sağlamayabilir. Örneğin Şekil 2'deki kod bloğunda İfade Kapsamayı sağlamak mümkünken MC/DC'yi sağlamak mümkün olmayacaktır. Çünkü ikinci kontrol bloğuna gelindiğinde A değişkeni sadece 2 veya 3 değerlerini alabilir ve böylece ikinci bloktaki else-if kontrolü asla yanlış sonuçlanmayacaktır. Böylece İfade Kapsama sağlanırken MC/DC sağlanamayacaktır. Yine MC/DC için geliştirilen gereksinimler ve testler İfade Kapsama için geliştirilen ürünlerden çok daha maliyetli olacaktır.

Aviyonik yazılım projelerinde, gerek geliştirilecek yazılımdaki güvenlik riskinin azaltılması gerekse yazılımın sertifikasyonu açısından DO-178B standardında anlatılan süreçlerin düzgün ve eksiksiz olarak uygulanmaları önem taşımaktadır. Bu standart herhangi bir yazılım süreç yöntemi önermemekle beraber tüm yazılım ürünleri için belirli süreçlerin uygulanmasını şart koşmaktadır. Bu süreçler, Yazılım Planlama Süreci, Yazılım Geliştirme Süreçleri, Tamamlayıcı (Integral) Süreçlerdir. Yazılım Planlama Süreci, yazılım geliştirme aktivitelerini ve proje için tamamlayıcı süreçleri tanımlar ve bu aktivitelerle süreçleri koordine eder. Yazılım Geliştirme Süreçleriyse yazılımı geliştiren yazılım gereksinim, yazılım tasarım, yazılım kodlama ve yazılım entegrasyon süreçlerinden oluşmaktadır. Son olarak Tamamlayıcı Süreçler, yazılım yaşam döngüsü süreçlerinin ve bunların ürünlerinin doğruluğunu ve güvenilirliğini sağlamaktan sorumlu olan ve yazılım yaşam döngüsü süresince yazılım geliştirme süreçleriyle beraber uygulanmak zorunda olan süreçlerdir. Bu süreçlerse yazılım doğrulama, yazılım konfigürasyon yönetimi, yazılım kalite güvencesi ve sertifikasyon irtibat (certification liaison) süreçlerinden oluşmaktadır. Standard sistem gereksinimlerinden yeni üretilecek bir yazılım birimi için Yazılım Geliştirme Süreçlerinin uygulanma sırası, yazılım gereksinim, tasarım, kodlama ve bütünleşme süreçleri olarak verilmektedir. Ancak doğrudan yazılım gereksinimlerinden kodlama, yazılımın yeniden kullanımı veya prototip geliştirme durumları için farklı bir sıralama önerilmektedir.

```
if ((A < 2) || (A > 3)) {
   exit();
}

if ( A == 2) {
...
}
else if ( A == 3 ) {
...
}
```

Şekil 2 İfade Kapsamaya Uygun MC/DC'ye Uygun Olmayan Kod Bloğu

Güvenlik kritik aviyonik yazılım geliştirirken DO-178B standardının yazılım yaşam döngüsünde toplanmasını şart koştuğu veriler vardır. Bu verilerin toplanma amacı yazılım yaşam döngüsü sürecinde aktiviteleri planlamak, yönetmek, açıklamak, tanımlamak, kaydetmek ve kanıt olarak sunmak olabilir. Bu veriler yazılımın ve yazılıma daha sonra gelebilecek olan değişikliklerin sertifikasyonu açısından önem taşımaktadır. Her bir yazılım birimi için belirtilen yazılım süreçlerini uygulamak gerekliliğinden yola çıkarak bu verilerin de her bir yazılım birimi için toplanması gerektiği sonucunu çıkarabiliriz. Bu verilere örnek olarak yazılım geliştirme, doğrulama, konfigürasyon yönetimi, kalite güvencesi, sertifikasyon planları; yazılım gereksinim, tasarım,

kodlama standartları; yazılım gereksinimleri; tasarım tanımları, kaynak kod, çalıştırılabilir obje kodu; yazılım doğrulama vaka ve prosedürleri, yazılım doğrulama sonuçları, yazılım yaşam döngüsü ortam konfigürasyon indeksi, yazılım konfigürasyon indeksi, sorun raporları; yazılım konfigürasyon yönetimi, kalite güvence kayıtları ve yazılım tamamlama özeti verilebilir.

3. Çevik Yöntemler ve XP Programlama

Çevik Yazılım Geliştirme, bir takım yazılım geliştirme yöntemlerine verilen genel isimdir. Ekstrem Programlama, Scrum, ASD (Adaptive Software Development), AUP (Agile Unified Process) ve Kristal Yöntemler Çevik Yöntemlere verilecek örneklerden bazılarıdır. Belirsiz veya tam olmayan ya da sürekli değişen gereksinimlerin bulunduğu; başka bir değişle ihtiyaçları sürekli değişen veya ihtiyaçlarını tam anlamıyla bilmeyen, ifade edemeyen paydaşların bulunduğu yazılım projelerinde paydaşlara yüksek kaliteli ürünler sunmayı amaçlamaktadır. Bu amacaysa paydaşların ve geliştirme ekibinin kendi içlerinde ve birbirleriyle sürekli olarak sıkı bir iletişim içerisinde olmasıyla; sürekli çalışan bir yazılım ortaya koymakla ve süreç içerisinde değişen ihtiyaçlara cevap verebilecek bir süreç yöntemiyle ulaşmayı hedeflemektedir. Çevik Yöntem destekçileri tarafından yayınlanan manifestoda da Çevik Yöntemin prensipleri (12 prensip halinde) açıklanmıştır [2].

Kuzey Amerika ve Avrupa'da birçok bilişim şirketi tarafından Çevik Yöntemlerin kullanıldığını veya yakın gelecekte kullanılmaları düşünüldüğünün işaretleri vardır [3]. Yine 13 Avrupa ülkesinden gömülü yazılım geliştiren 35 şirkette yapılan araştırmanın sonuçlarına göre Çevik Yöntem (XP Programlama ve Scrum) tekniklerinin kullanımında ve şirketleri bu yöntemlere adaptasyon sağlama sürecinde artış görülmektedir [4]. Bu yöntemlerden Ekstrem Programla günümüzde popüler Çevik Yöntemlerin başında gelmektedir. XP Programlama pratiklerini "planlama süreci, test güdümlü geliştirme, erişilebilir (on-site) müşteri, sürekli bütünleşme, tararım iyileştirme, küçük yayımlar (releases), kodlama standardı, ortak kod ve ortak (pair) programlama, basit tasarım, yeniden yapılandırma (re-factoring), metafor ve 40 saatlik mesai" başlıkları altında toplamak mümkündür [6]. Bu bildiride Ekstrem Programlama yöntemi üzerinde durulmuştur. Her ne kadar Ekstrem Programlama Çevik Yöntemlerden olsa da, her bir Çevik Yöntemin kendisini özgü özellikleri bulunmaktadır.

Genel olarak, plan tabanlı yöntemler gereksinimlerin büyük kısmının değişken olmadığı ve gereksinimlerdeki değişikliklerin tahmin edilebilir boyutta olduğu projelerde uygulanılırken; çevik yöntemler, büyük ölçüde değişken gereksinimlerin bulunduğu ve hızlı değişkenlik gösteren ihtiyaçların olduğu projelere çözüm getirmek amacıyla ortaya çıkmıştır. Bu anlamda çevik yöntemlerin plan tabanlı yöntemlerden belli başlı farkları bulunmaktadır [5].

4. XP Programlama Pratiklerinin Güvenlik-Kritik Sistem Yazılımında Kullanımı

4.1. Güvenlik Kritik Yazılım Geliştirme Sürecinde Uygulanan XP Programlama Pratikleri

Bu bölümde, en az bir kez, küçük ile orta ölçek arası bir güvenlik kritik yazılım projesinde uygulanan ve başarı elde edilen XP Programlama pratiklerinin uygulama yöntemleri yer almaktadır (XP Programlama yöntemi temel yazılım yaşam modeli olarak uygulanmamıştır). Pratiklerin uygulama yöntemleri projeye göre değişiklik göstermiştir. Bu bildiride verilen bilgiler, 30 KSLOC (kilo cinsinden mantıklı yazılım kod satırı) büyüklükteki ve yaklaşık bir yıl geliştirme süresi olan bir gömülü yazılım projesi temel alınarak verilmiştir. Sürekli bütünleşme, kodlama standardı, tasarım iyileştirme, test aktiviteleri, ortak kod pratikleri genel olarak projenin tüm birimlerinde uygulanmıştır. Ancak ortak programlama pratiğinin uygulanması ve erişebilir müşterinin faydaları anlatılacağı gibi sadece ihtiyaç duyulan birimlerde (gereksinimlerin tam olarak belirlenemediği ve değişikliklerin sıkça yaşandığı kısımlardan sorumlu birimlerde) uygulanmıştır.

İlk olarak "Ortak Kod" pratiğini ele alalım. Her bir yazılım parçacığına geliştirme sürecinde temel sorumlu atanıyor. Normal şartlarda, geliştirme sürecinde bu sorumluya ilgili yazılım parçacığına yapılacak değişiklik hakkında temel bilgi verilerek değişiklik yapılıyor. Ancak değişen durumlara (örnek: Sorumlunun o an uygun olmaması, sorumlu kişinin yöneticisini bilgilendirmek, çok acil yapılması gereken bir değişikliğin gündemde olması gibi) göre sorumluya bilgi verilmeden de değişiklik yapılabilir ya da sorumlu değişiklikten sonra da bilgilendirilebilir. İlgili yazılım parçacığı yayımlandıktan sonra gerçek anlamda o projede görev alan herkesin malı olarak görülür.

"Ortak Programlama" pratiği XP Programlamadan farklı olarak gerekli görüldüğü koşullarda uygulanmıştır. Özellikle deneyimi az olan yazılım geliştiriciye ortak olarak daha deneyimli bir geliştirici verilmek suretiyle, daha az deneyimli olanın sorumlu olduğu kısımların temel veya karmaşık kısımlarının gerçekleştirilmesi aşamasında beraber çalışmaları sağlanmıştır. Ayrıca herhangi bir yazılım geliştiricinin sorumlu olduğu kısımlardaki konu hakkında daha fazla bilgi sahibi olan ve zaman olarak uygun olan kişinin, asıl sorumlu kişi ile beraber çalışması da mümkün olmuştur.

"Sürekli Entegrasyon", ücretsiz bir sürekli entegrasyon aracının şirkette halihazırda kullanılan konfigürasyon yönetimi aracı ile birlikte kullanılarak sağlanmış ve geliştirilen tüm kaynak kod, koddaki her değişikliğin ardından otomatik olarak yeniden derlenmiştir. Derlemeler farklı derleyicilerde ve değişik derleme seçeneklerine göre yapılmıştır (örnek: iki farklı işletim sistemi için derleme yapılmıştır).

"Yeniden Yapılandırma" proje takvimindeki zaman kısıdının elverdiği ölçüde ve ihtiyaç duyuldukça gerçekleştirilmiştir. Koddaki karmaşıklık düzeyinin gözden geçirmelerde harcanacak zaman kaybına ve bakım maliyetine olan etkisi değerlendirilip uygun görüldükçe yapılmıştır. Elbette bu değerlendirmede ilgili kod parçacığını yeniden yapılandıracak kişinin harcayacağı zaman da dikkate alınmıştır. Örnek olarak iki bin satırı geçen bir dosya kohezyonu artırmak için farklı işlevlere göre birkaç farklı dosyaya bölünmüştür.

"Tasarım İyileştirme" faaliyetinin uygulama şekli "Yeniden Yapılandırma" pratiğinin uygulama şekliyle benzerlik

göstermektedir. Ek olarak mevcut tasarımla devam etmenin maliyeti, değiştirince çıkacak tasarımla devam etmenin maliyetini geçtiğinde tasarım iyileştirilmesi yapılmıştır. Ayrıca değişen, yeni eklenen ve çıkarılan gereksinimlerin tasarıma olan etkisi analiz edilmiştir ve tasarıma etkileri olduğunda, tasarımdaki gerekli değişiklik yapılmıştır. Tasarımdaki her değişiklikten sonra tasarımın mevcut yazılım gereksinimlerine ve kontratsal kısıtlara uygunluğu kontrol edilmiştir. Tasarım onayları uygunsuzluklar giderildikten sonra gerçekleştirilmiştir.

"Kodlama Standardı" DO178B'nin yazılım yaşam döngüsü verileri arasında da yer almaktadır. AYESAŞ'ın CMMI–3 olgunluk düzeyinde olmasının getirisi olarak hali hazırda var olan kodlama standardı metotlarından ilgili olan metot, projede kullanılan programlama diline uygun olarak seçilmiştir. Seçilen kodlama standardı projenin gerektirdiği şekilde değiştirilerek projede uyulması zorunlu hale getirilmiştir. COTS bir yazılım ile geliştirilen kodun projenin kodlama standardına uygunluğu kontrol edilmiş ve kontrol sırasında bulunan uyumsuzluklar düzeltilmiştir.

Müşteri iletişim anlamında ne kadar erişilebilirse ve teknik anlamda geliştiricilerin olası sorularını cevaplandırabilme anlamında ne kadar başarılıysa "Erişilebilir (On-site) Müşteri" pratiği o kadar uygulanabilirdir. Çalıştığımız müşterileriyle ihtiyaç duyulduğu takdirde gerek müşteriyi bilgilendirmek amacıyla gerekse müşteriden bilgi almak amacıyla etkili bir şekilde iletişim içerisinde olunmuştur. Böylece projedeki belirsiz kısımlar için geliştirilen çözümlerde daha yüksek düzeyde başarı sağlanmıştır.

Projenin gereksinim analizi aşamasından itibaren projedeki işlevler, donanımsal birimler ve özel veri yapıları için hem geliştiricilerin kendi aralarında hem de müşteri ile iletişimlerinde kullanabilecekleri özel tanımlar yapılmış ve bu tanımların kısaltmaları belirlenmiştir. "Metafor" pratiğine benzer bir uygulama söz konusudur. Örneğin yazılımın üzerinde çalışacağı kartın işlevleri isimlendirilmiş ve ilgili işlevlerden bahsedilirken ilgili isimler kullanılmıştır.

4.2. Uygulanması Tercih Edilmeyen XP Programlama Pratikleri ve Tercih Edilmeme Nedenleri

XP Programlama yönteminin önerdiği "Planlama Süreci" ve "Küçük Yayımlar (Releases)" pratiklerini orta ve büyük ölçekli yazılım projelerinde başarıyla uygulamak mümkün olmakla beraber, bazı koşulların sağlanması gerekmektedir. Öncelikle, eğer yazılım projesi şirket içinde yürütülen bir proje değilse kontratsal olarak müşteriye bağımlılık vardır. Çünkü kilometre taşları tam olarak belli değildir. Yine projede beraber çalışılacak müşterinin teknik ve iletişim becerilerinin yüksek olması beklenmektedir. Ayrıca projede görev alacak yazılım geliştirme ekibinin de XP Programlamadaki şekilde yapılan planlara uyabilmesi ve zamanında yayımları yapabilmeleri beklenmektedir. "Basit Tasarım" pratiğinin XP Programlama yöntemindeki planlama ve küçük yayımlar pratikleriyle sıkı ilişkisi vardır. Her yayımda seçilmiş gereksinimler geliştirildiğinden dolayı mevcut tasarımda da sadece mevcut yayımda geliştirilecek gereksinimleri kapsayacak şekilde tasarım yapılmaktadır. Bahsedilen koşullar bu pratik için de geçerlidir. Ancak örnek olarak verilen projede kontrastsal yükümlülüklerden dolayı şelale temel yazılım yaşam döngüsü modeli olarak seçilmiştir.

"40 Saatlik Mesai" yazılım dünyasındaki tüm çalışanların karşı çıkmayacağı bir olgudur. Ancak proje takvimine göre belli aşamalarda fazla mesai kaçınılmaz hale gelmiştir. Her ne kadar fazla mesai ile daha hızlı ürün çıkarmak mümkün olsa da artan fazla mesailerle beraber çalışanların hata yapma potansiyelleri artmaktadır. Bu anlamda olası fazla mesailerde, proje yönetiminin etkin yürütülmesiyle (örnek: proje takvimine göre erken önlem almak) ve fazla mesaileri proje takvimine yaymak suretiyle, proje boyunca gerekli adam saate göre mesai saatleri yönetilmiştir.

Güvenlik kritik yazılım projelerinde test çok önem kazanmaktadır. Örneğin; DO-178B A seviyede geliştirilen bir yazılım projesinde yüksek seviyede yapısal kapsama sağlamak için geliştirilen kod için ciddi anlamda test aktivitesi uygulamak gereklidir. XP Programlamanın "Test Güdümlü Geliştirme" pratiğinde, fonksiyonel testlerin müşteri tarafından yazılması ve yazılım geliştiricinin birim testi haricinde test yazmaması öngörülmektedir. Bu pratiği DO-178B standardına uygun yazılım projelerinde olduğu gibi uygulamak çok uygulanabilir olmayacaktır. Bizim modelimizde de yazılım geliştirici birim (fonksiyonel) testlerini gerçekleştirir ancak ayrı bir ekip gereksinim tabanlı (kalite bazlı) testleri geliştirmekten ve koşmaktan sorumludur.

5. Tartışma

Çevik yöntemleri güvenlik-kritik sistemlere yönelik yazılım projelerinde temel yazılım yaşam döngüsü modeli olarak uygulamak hem kontratsal kıstaslara uyum, hem projenin yönetimi hem de projenin boyutuna göre çalıştırılacak yazılımcılar ve bu yazılımcıların kalitesi açısından zorluklar getirecektir. Bu tür yazılım projelerinde, plan tabanlı yöntemleri temel yazılım yaşam döngüsü modeli olarak seçip, proje sürecinde yürütülecek aktivitelerde uygun XP Programlama yöntem pratiklerini, projenin gerektirdiği risk yönetim modeli dâhilinde uygulamak fayda sağlayacaktır. Bu hedefe ulaşmak için gerekli koşulların sağlanması (örnek: Yazılım geliştiricilerin yeterince kaliteli, müşterinin yeterince bilgili ve hızlı iletişim kurabilen, gibi...) ve projede görev alan yönetim kadrosunun ilgili pratikleri projenin ilgili aşamalarında uygulayabilir bilgi ve birikime sahip olmaları gerekmektedir.

XP Programlama yönteminin temelini oluşturan "Planlama Süreci" ve "Küçük Yayımlar (Releases)" pratiklerinin güvenlik kritik yazılım projelerinde uygulanması için belli ön koşullar mevcuttur. Bu pratikleri uygulamak için ciddi bir yazılım proje yönetimi, projede yer alacak geliştiricilerin yeterince kaliteli olmaları, projenin özelliklerinin (yazılımın boyutu ve türü (şirket içi araştırma - geliştirme ve ya alt yüklenici projesi gibi), müşterinin kabiliyetleri, kontrastsal kısıtlar) uygun olması gibi birçok koşulun sağlanması gereklidir. Böyle bir durumda XP Programlama gibi herhangi bir çevik yöntemim, birçok belirsiz gereksinimlerin olduğu güvenlik kritik bir yazılım projesinde temel yazılım yaşam döngüsü modeli olarak seçilmesi gündeme gelebilir.

6. Sonuçlar

Ortak Kod ve Ortak Programlama pratiklerinin uygulanmasıyla acil güncellemeler rahatlıkla yapılabiliyor, bir yazılım parçası hakkında birden fazla sayıda bilgi sahibi yazılımcı olması sağlanmıştır. Ortak Programlama pratiğinin uygulanması ile daha az deneyimli geliştiricilerin bilgi birikimlerindeki artış hızında artış sağlanmıştır. Ayrıca ilgili kişinin sorumlu olduğu kısımların gerçeklenmesi daha kısa sürmüştür.

Sürekli Entegrasyon pratiği sayesinde kodda yapılan herhangi bir değişikliğin, kodun entegrasyonunu bozup bozmadığı anında anlaşılır hale gelmiştir. Derlemeyi bozan bir değişikliğin anında düzeltilmesi sağlanmıştır. Hatta AYESAŞ'ın geliştirdiği bir test otomasyon ürününün sürekli entegrasyon aracıyla kullanılmak suretiyle, otomasyonu sağlanmış testlerin gerek Windows ortamında gerekse gerçek hedef platformlar üzerinde otomatik koşmaları sağlanmıştır. Böylece kısıtlı olan hedef platformların etkili kullanımı ve herhangi bir değişiklik nedeniyle başarısız olan test adımlarının tespiti sağlanmıştır.

Yeniden Yapılandırma pratiğinin uygulanabilmesi için proje ekibindeki yazılım geliştiricilerin yeterince bilgi ve deneyime sahip olması çok önemlidir. Eğer yeterince bilgi ve deneyime sahip geliştirici yoksa yeniden yapılandırmaların maliyeti artmaktadır. Bu pratik gerek duyuldukça Ortak Programlama pratiği ile beraber uygulanmıştır ve kodla ilgisi olan geliştirici sayısı arttığından yeniden yapılandırmada görev alabilecek kişi sayısı da artmıştır.

DO-178B standardına göre, tasarıma etkisi olan bir değişiklikte tasarımın da güncellenmesi gerekmektedir. Ancak projenin test, entegrasyon gibi aşamalarında yapılacak tasarım güncelleme aktivitesinin maliyeti oldukça fazla olacaktır. Çünkü tasarım değişikliğinin koda, testlere ve destek yazılım ürünlerine (örnek: Yazılım Kullanıcı El Kitabı) etkisi olacaktır. Kodlama standardı yazılım geliştiriciler için vazgeçilmez bir olgu haline gelmektedir. Kodlama standardı ile geliştirilen kodun gözden geçirmelerinde harcanan zamanda azalma, anlaşılırlığında artış, bakımında kolaylık sağlanmıştır. "Erişilebilir (On-site) Müşteri" ile çalışmak size bağlı değildir, proje ve müşteriye göre değişir. Erişilebilir müşteriyle sürekli iletişim içerisinde çalışmak projedeki belirsiz gereksinimlere geliştiricilerin daha hızlı ve daha doğru çözümler getirmesini sağlamıştır.

Metaforlarla anlatılmak istenen olguların tüm takım tarafından anlaşılması için belli bir öğrenme süresi gerekmektedir. Ayrıca bazı olgular tam anlaşılmadan ya da yanlış anlaşılarak kullanılabilmektedir. Bu yüzden doğru öğrenilmelerinde fayda vardır. Örnek olarak, özel bir veri kanalından alınan veriyi görüntülemek için kullanılan özel bir görüntüleme şekli için seçilen bir metafor, aynı veriyi farklı bir yöntemle görüntülemek için kullanılan metotla karıştırılmıştır. Ancak metaforların doğru kullanıldığı durumlarda hem geliştiricilerin birbirleriyle olan iletişimlerinde hem de müşteriyle olan iletişimlerinde harcanan zamanlar azaltmıştır.

Uygulanan pratiklerin projedeki etkilerine bir bütün olarak bakmak gerekirse projedeki üretkenlikte artış sağlanmıştır. Takım içi iletişimdeki artış ve müşteriyle olan etkin iletişim, projede bulunan belirsizliklerin çözümünü hızlandırmış ve bu noktaların risk yönetimini etkinleştirmiştir (Örnek vermek gerekirse; proje boyunca kullanılan donanımda değişiklikler meydana gelmiştir. Buna bağlı olarak donanım değişikliğinin en fazla etkilediği yazılım biriminde üç defa temel tasarım değişikliğine gidilmiştir. Yine ilgili birimin (yaklaşık 5 KSLOC) yazılım gereksinimlerinde on defadan fazla, birimin bütününü etkileyen temel değişiklikler olmuştur). Ancak temelde plan tabanlı bir yazılım yaşam döngüsü modeli kullanıldığından dolayı belirsiz gereksinimlerin yol açtığı güncellemeler için belli bir iş yükü (rework) ortaya çıkması engellenememiştir. Ama bu güncellemelerden kaynaklı maliyetlerin projeye olan yükünün kullanılan pratikler ve uygulanan risk yönetimi sayesinde düşük seviyede kalması sağlanmıştır. XP Programlamayı güvenlik kritik yazılımlarda temel yazılım yaşam döngüsü modeli olarak uygulamak için birçok koşulun sağlanması gerekirken, uygulanmasında fayda elde edileceği düşünülen XP Programlama pratiklerini uygun görüldükçe projenin uygun safhasında uygulamak için herhangi bir kısıt yoktur. Ayrıca XP Programlama, projenin temel yazılım yaşam döngüsü modeli olmasa da, projede geliştirilmesi hedeflenen belli konfigürasyon parçaları için, örneğin değişken ve belirsiz gereksinimlerin bulunduğu kısımlar için bir bütün olarak uygulanabilir.

7. Teşekkür

Yazılım mühendisliğinde bilgi paylaşımı için gerekli ortamı bizlere sağladığı için UYMS 2009 organizasyon komitesine, XP Programlama tekniklerini AYESAŞ'ta profesyonel anlamda uygulama ortamı sağlayan Alparslan Arslan ve Orhan Uğurlu'ya teşekkürlerimizi sunarız.

8. Kaynakça

[1] RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification", 1992.

[2] http://www.agilemanifesto.org/principles.html'den ulaşılabilinir, Mayıs 2009'da erişildi

[3] C. Schwaber, R. Fichera, "Corporate IT leads the second wave of agile adoption", Forrester Research Inc., 2005

[4] O. Salo and P. Abrahamsson, "Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum", IET Softw., Vol. 2, No. 1, February 2008

[5] Barry Boehm, "Get Ready for Agile Methods, with Care", IEEE Computer, 2002

[6] R. Jeffries, A. Anderson, and C. Hendrickson, "Extreme Programming Installed", Addison Wesley, 2001

[7] Joshua Kerievsky, Senior Consultant, Cutter Consortium, "Industrial XP: Making XP Work in Large Organizations", Agile Project Management Vol. 6, No. 2

[8] K. Beck, "Extreme programming explained: Embrace change. Reading", MA: Addison-Wesley, 1999

[9] S. Fraser, K. Beck, W. Cunningham, R. Crocker, M. Fowler, L. Rising, L. Williams,"Hacker or hero? - extreme programming today (panel session)", Conference on Object Oriented Programming Systems Languages and Applications, Minneapolis, Minnesota, United States, 2000

[10] L. Williams, R. R. Kessler, W. Cunningham, "Strengthening the case for pair programming", Software, IEEE 17 (4), 2000