Gömülü Yazılım Geliştirme Pratikleri

Çiğdem Turan[1] Alaattin Dökmen[2] Sadık Akdağ[3] Bora Kartal[4]

[1,2,3,4] ASELSAN A.Ş., REHİS Grubu, Ankara

[1]e-posta: cturan@aselsan.com.tr [2]e-posta: adokmen@aselsan.com.tr

[3]e-posta: sakdag@aselsan.com.tr [4]e-posta: bkartal@aselsan.com.tr

Özetçe

Büyük ölçekli sistem projeleri için geliştirilen gömülü yazılımların, sistemdeki diğer yazılımlar veya donanımlar ile entegrasyonu ve ardından gerçek ortamda çalıştırılması süreci, proje takviminin sonlarına kalmakta ve genellikle geliştirme süreci tamamlanamadığı için gecikmektedir. Bu gecikme, entegrasyon ve gerçek ortam testleri sırasında oluşabilecek problemlerin çok sonra görülmesi sonucuna yol açmakta ve gelinen aşamada bulunan hataları çözmenin maliyeti yüksek olmaktadır.

Gömülü yazılımların test süreci, bilgisayar ortamında çalışan yazılımlara göre daha zor olmaktadır. Bilinen yöntemler dışında, uygulamaya özel çözümler üretmek gerekmektedir.

Ayrıca, sistem ilk defa geliştiriliyor ise, sistemin tasarımında hatalar yapılması normaldir ancak bunlar erken bir aşamada farkedilip düzeltilmezse geri dönüş çok zor olmaktadır.

Bu makalede, bu olası problemlere karşı ASELSAN, REHİS Grubu'nda uygulanan çözümler anlatılmaktadır. Bunlar, yazılım geliştirme sürecine ek faaliyetler olarak değerlendirilmelidir.

1. Giriş

Gömülü yazılım geliştirmede geleneksel yöntemlere ek olarak, uygulamaya özel yazılım geliştirme ve test faaliyetlerine ihtiyaç duyulduğu tecrübeler sonucunda anlaşılmıştır. Bu bildiride, gömülü yazılımlar ile ilgili genel bilgi verildikten sonra, gömülü yazılım geliştirmede karşılaşılan problemler ortaya konulup, bu problemleri önlemek için uyguladığımız çözümler anlatılacaktır.

1.1. Gömülü Yazılım Projeleri İle İlgili Bilgiler

Gerçek zamanlı gömülü sistem projeleri, çok genel bir ifadeyle, sensörlerden veri toplanması, bu verilerin çeşitli matematiksel algoritmalar tarafından işlenerek sonuçların üretilmesi ve bu sonuçların kullanılması şeklinde tariflenebilir. Gömülü yazılım geliştirmenin diğer yazılım geliştirme türlerine göre farklıkları ve zorlulukları şöyle özetlenebilir: [1]

- Sistemin gömülü yazılımları ile donanımları aynı zamanda geliştirilmeye başlar.
- Donanımlar genellikle gecikir ve yazılım – donanım entegrasyonu için çok az zaman kalır.
- Bir hata oluştuğunda, bunun yazılımla mı yoksa donanımla mı ilgili olduğunu anlamak çok zordur ve bunu anlama görevi yazılıma düşer.
- Gerçek zaman, eş zamanlı çalışma (multitasking) ve güvenlik ihtiyaçları vardır.
- Bilgisayar yerine özel işlemci kartları üzerinde çalıştıkları için hafıza ve işlemci gücü kısıtları vardır.

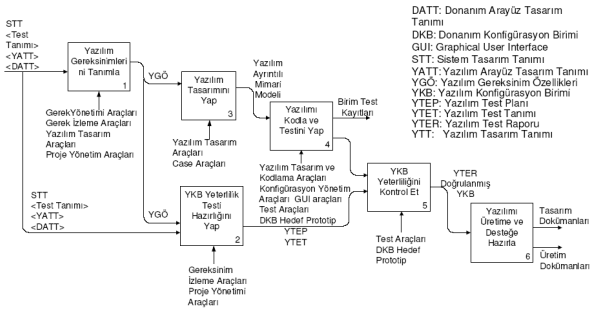
Tüm bu sebeplerden ötürü, gömülü yazılım geliştirme için farklı bir disiplin gerekir.

1.2. Bildiriye Konu Olan Proje İle İlgili Genel Bilgi

Aselsan REHİS bünyesinde geliştirilen bu proje, tipik bir gerçek zamanlı gömülü sistem projesi olmakla birlikte içinde birçok yazılım ve donanımı barındıran ve geliştirme süresi 4 yıl olan bir projedir. Yazılım Konfigürasyon Birimi (YKB) sayısı 4, algoritma modülü sayısı 10'dur.

YKB'ler, sistem senaryolarını gerçekleyen, donanım ve yazılım arayüzlerini sağlayan, gerekli algoritma modüllerini gerektiği zaman çalıştıran ve onlara veri sağlayan yazılımlar olarak değerlendirilebilir. Algoritma modülleri ise, belli bir işlevi yerine getiren, matematiksel yazılım parçaları olarak tanımlanabilir. Örneğin, yön bulma algoritması. Algoritma modülleri öncelikle Matlab veya Simulink gibi bir simulasyon ortamında geliştirilir daha sonra yazılım ortamına aktarılır.

1.3. Yazılım Geliştirme Süreci



Şekil 1: Yazılım Geliştirme Süreci [2].

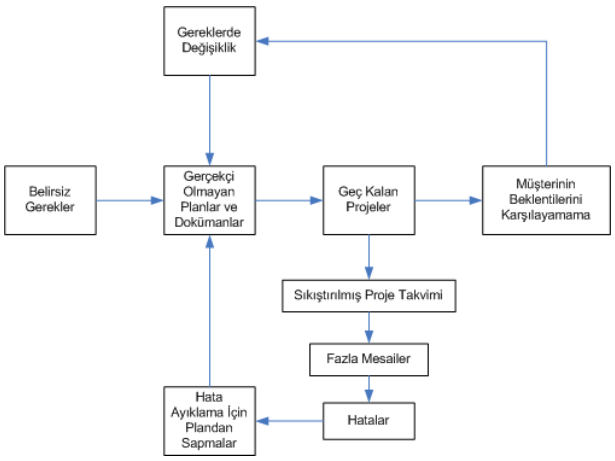
REHİS grubunda uygulanan yazılım geliştirme süreci Şekil1'de verildiği gibidir.

Yazılım geliştirme süreci, sistem tasarımının belirlenmesinin ardından, yazılım gereksinimlerinin tanımlanması ile başlar. Daha sonra mimari ve ayrıntılı tasarım yapılır; tasarımla paralel bir şekilde yazılım test hazırlıkları gerçekleştirilir.

Tasarım sonrası kodlama ile yazılım gerçekleme işlemi yapılır. Gerçekleme sırasında yazılım birim testleri de gerçekleştirilir. Yazılım gerçekleme çalışmaları tamamlandığında yeterlilik testleri test mühendisleri tarafından yapılır. Son aşamada da test edilmiş yazılımın işlevsel ve fiziksel tetkikleri yapılarak süreç tamamlanır. Süreçte her zaman her noktadan geri dönüşler ve aşamalar arasında iç içe geçmeler olabilir. [2]

2. Gömülü Yazılım Geliştirmede Temel Problemler Modeli

İlk defa geliştirilen sistemlerde, genellikle, "yazılım gereksinimlerini tanımla" adımına girdi olan Sistem Tasarım Tanımı (STT) dokümanının hazırlanması çok uzun sürer. Sistem tasarımı bir takım belirsizlikler ve bilinmezlikler içerir ve sürekli gelişmeye açık durumdadır. Sistem tasarımı devam ederken ve belirsizliğini korurken yazılım geliştirme faaliyetlerine ***başlanmaması*** şöyle bir "tehlikeli süreç" ortaya çıkarır:



Şekil 2: Tehlikeli Süreç.

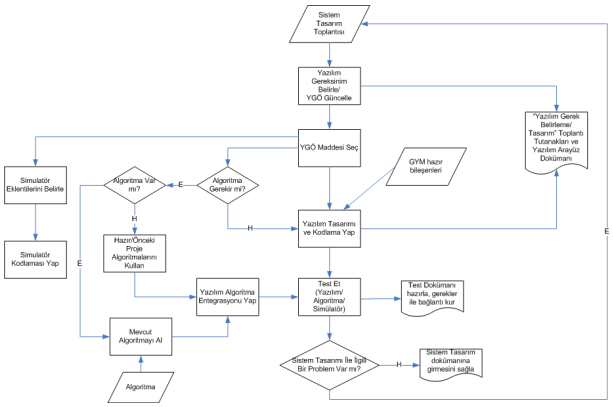
Şekil2'deki model, sistem tasarım gereklerindeki belirsizlikleri uzun süre çözemeyen ve bu süre boyunca da yazılım geliştirme sürecine başlamayan projelerin başlarına gelebilecekleri anlatmaktadır. Sistem tasarım gereklerindeki belirsizlik devam ederken yapılan planlar, yazılan tasarım dokümanları gerçekçi olmaz. Bu süreçte yazılım geliştirme faaliyetlerine başlanmazsa bir süre sonra projenin geç kalması ihtimali ortaya çıkar. Bu andan itibaren tehlikeli süreç başlar. Öncelikle sıkıştırılmış proje takvimi hazırlanır. Bu takvim fazla mesailere ihtiyaç duyar. Takvim baskısı altında çalışılırken hata yapma olasılığı artar. Hatayı ayıklama işleminin ne kadar süreceği planlanamaz. Gecikildiği için bazı yetenekler gerçeklenemez ve bu, müşteri memnuniyetsizliğini doğurur. Müşteri bu durumda gereklerde değişiklik yapma ihtiyacı duyabilir. Bu da yine tehlikeli süreci besleyen bir durumdur.

Araştırıldığında, tüm dünyada gömülü yazılım projelerinin aynı problemlerle karşı karşıya kaldığı görülmektedir. Projelerin geç kalması, hata miktarının fazla olması, maliyetinin doğru hesaplanamaması gömülü sistemler dünyasında sık karşılaşılan problemlerdendir. [1] Nitekim, bu başlıkta anlatılan tehlikeli sürece çok benzer bir model, 1 numaralı referansta da anlatılmıştır.

3. Gömülü Yazılım Geliştirme Sürecine Ek Faaliyetler

Tehlikeli süreci yaşamamak için sistem tasarım dokümanının tamamlanmasını beklemeden, sadece planlar yapmak yerine yazılım geliştirme faaliyetlerine başlamak gerekir. Gerekler net değilken yazılım geliştirmek zordur, ancak yazılım geliştirmeye geç başlamak intihar etmektir.[3]

Süreçteki, "Yazılım Gereksinimlerini Tanımla", "Yazılım Tasarımını Yap" ve "Yazılımı Kodla ve Testini Yap" adımları detaylandırıldığında ve birtakım faaliyetler eklendiğinde Şekil3'teki model ortaya çıkar:



Şekil 3: Yazılım Geliştirme Sürecine Ek.

Yazılım geliştirme faaliyetleri, sistem tasarım toplantıları sonucunda karar verilen, sistemin en temel gerekleri ile başlar. Bu toplantılarda alınan kararları gerçekleyecek olan yazılım gerekleri belirlenir. Yazılım Gereksinim Özellikleri (YGÖ) dokümanına ilgili gerekler eklenir; mevcut gerekler etkileniyorsa onlar da güncellenir. Sistem tasarımının başlarında bu süreç biraz daha yavaş işler; sistem tasarımının olgunlaşması beklenir. Ancak bu süre mümkün olduğunca kısa tutulmaya çalışılarak, yazılımların sistem tasarımını takip etmesi sağlanır.

YGÖ'deki gereklerin gerçeklenmesi aşaması başlar. Sistemin bir işlevini gerçekleştirecek olan, bir grup gerek seçilir. Bu gerekler birden fazla yazılımı da ilgilendiriyor olabilir. Yazılım tasarım toplantılarında, ilgili YKB'lerin sorumluları ile birlikte yazılım tasarım kararları alınır; algoritma ve simulatör ihtiyaçları ile daha önce GYM bünyesinde gerçekleştirilmiş ve ortak bir yerde bulunan bileşenlerden, kullanılabilecek olanlar belirlenir ve alınan kararlar bir toplantı tutanağı şeklinde kaydedilir. Yazılımlar arası arayüz mesajları Yazılım Arayüz Tasarım Tanımı (YATT) dokümanına işlenir.

YKB, algoritma ve simulatör sorumluları tarafından, yazılım tasarımı, kodlaması ve testi yazılım geliştirme sürecinde anlatıldığı şekilde gerçekleştirilir.

Bu çalışmaya konu olan sistem tasarım kararlarının, eğer gerçeklemede bir problem yoksa sistem tasarım dokümanına işlenmesi veya güncellenmesi sağlanır. Bir problemle karşılaşılırsa, sistem tasarım ekibi ile yeni durum değerlendirilir.

3.1. Ön Prototipleme ile Sistem Gereklerinin Belirlenmesine Katkı

Temel yaklaşım; "sistem gereklerinin belli olduğu kadarıyla yazılım geliştirmeye başla" şeklindedir. Yazılım geliştirme sürecine başlamanın ve temel gereklerin gerçeklenmesi ile ilk sürüm yazılımların (ön prototip) ortaya çıkarılmasının sistem gereklerinin belirlenmesinde yönlendirici ve hızlandırıcı bir etkiye sahip olduğu tecrübe edilmiştir. Özellikle, bu yeteneklerin sistem tasarım ekibine bir "demo" ile gösterilmesi oldukça etkili olmaktadır.

3.2. Simulatör Geliştirme

Temel amaç, yazılımın, sistem entegrasyonu aşamasında karşılaşacağı yazılım ve donanımları; teslimat testleri aşamasında karşılaşacağı ortamı simule ederek beklenmedik bir duruma maruz kalmasını önlemektir.

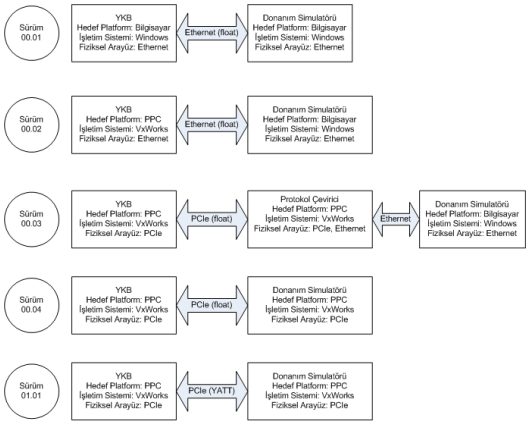
3.2.1. Donanım Simulatörü

Gömülü yazılım geliştirme süreci başladığında, ne yazılımın üzerinde çalışacağı donanım (hedef platform), ne de yazılımın haberleşeceği diğer sistem donanımları hazır olmaktadır. Ancak, bunlar hazır oluncaya kadar beklemek yerine, uygun tasarım teknikleri ve simulatörler kullanılarak bu sorun çözülür, hatta bir fırsata bile dönüştürülebilir.

Yazılımın kontrol ya da veri transferi amacıyla haberleşeceği sistem donanımlarının olmadığı durumda, yazılım geliştirme devam eder; bir donanıma ihtiyaç duyulduğu an o donanımın iç yapısından ve tasarım detaylarından bizi soyutlayan, sadece arayüz gereklerini gerçekleyen bir simulatör yazılır. Donanımın iç yapısından soyutlandığımız ve bizim istediğimiz şeyleri istediğimiz zaman gerçekleyen, hata yapmayan bir donanımımız olduğu için bunu yazılım geliştirmede bir avantaj olarak değerlendirebiliriz.

Simulatörler sistemin diğer yazılımları ile birlikte gelişirler. Yazılıma yeni gerekler eklendiğinde, bu gerekleri test etme amacıyla simulatörlere de yeni gerekler eklenir. Gerçeklenen ilk simulatörün donanımla birebir aynı olması beklenmez, simulatörler evrilerek gelişir.

Şekil4'te örnek bir simulatör için YKB ile birlikte gelişim evreleri tariflenmiştir:



Şekil 4: Simulatörlerin ve Yazılımların Gelişimi.

Sürüm 00.01'de, hem YKB hem de donanım simulatörü bilgisayar ortamında geliştirilir. Çünkü henüz yazılımın hangi hedef ortamda çalışacağı, donanımla hangi fiziksel arayüz üzerinden konuşacağı belli değildir. Ancak, sistemin bazı görevlerini yerine getiren ve donanımla bilinen bir arayüz üzerinden haberleşen bir sürüm çıkarılmıştır. Bu aşamada arayüz mesajlarının tipleri konusunda herhangi bir çalışma yapılmaz, "float" olduğu varsayılarak devam edilir.

Geliştirme süreci boyunca bilgisayar ortamında da çalışabilen YKB'ye sahip olmak test açısından birçok kolaylık sağlayabilir. Örneğin, "dinamik kapsama" testlerini bilgisayar ortamında yapmak, sınırlı hafızaya sahip özel işlemci kartları üzerinde yapmaktan daha kolay olacaktır.

Sürüm 00.02'de, YKB'nin hangi platformda çalışacağı belli olmuştur ve bu platform temin edilmiştir. Ancak henüz donanımla fiziksel arayüzün ne olacağı belli değildir. Bilinen bir arayüzle geliştirme devam eder. Bu sürümde de arayüz mesajlarının tip bilgileri konusunda bir çalışma yapılmaz.

Sürüm 00.03'te, fiziksel arayüze karar verilmiştir. YKB bu arayüz üzerinden haberleşecek şekilde kodlanır. Ancak bilgisayar ortamındaki donanım simulatörü ilk aşamada PowerPC (PPC) ortamına aktarılmaz; bir protokol çevirici yazılarak donanım simulatörü PPC ortamına aktarılıncaya kadar bu şekilde çalışılır.

Sürüm 00.04'te, donanım simulatörü de PPC ortamına aktarılmıştır. Tek eksik, arayüz mesajlarının nihai forma dönüşmesidir.

Sürüm 00.05'te, artık gerçek donanım ile entegre olabilmek için hiçbir engel kalmamıştır.

3.2.2. Ortam Simulatörü

Karşılaşılan bir diğer problem, entegrasyon ortamında düzgün bir şekilde çalışan yazılımların arazi koşullarında çalışmaması ya da birtakım hataların oluşmasıdır. Bu duruma sebep olabilecek, akla ilk gelen durum, arazi koşullarında yazılımın yoğun veri miktarı ile çalışmak zorunda kalması olabilir. Bunun gibi birçok sebep sayılabilir.

Geliştirilen proje hangi ortamda çalışacaksa, yazılımın o ortamda maruz kalacağı veri akışını, geliştirme sırasında sağlayabilmek büyük bir avantaj sağlar.

Tüm bu amaçlar düşünülerek üretilen yazılıma "Ortam Simulatörü" adı verilmektedir. Bu simulatör simule edilmek istenen arazi koşullarına göre oldukça kapsamlı olabilir. Görsel birtakım öğelerle ortamı simule etmek ve ona göre veri üretmek gerekir. Bir kendini koruma sisteminin maruz kalabileceği ortamdaki tehditleri simule etmek ortam simulatörüne bir örnektir. Tamamen uygulamaya özel yazılımlardır.

3.3. Test Sürecine Katkı

Kara kutu ve beyaz kutu testlerinin gömülü yazılımlar için yeterli olmadığı durumlarla karşılaşılabilmektedir. Hem geliştirme sırasında hem de entegrasyon ve arazi testleri sırasında, yazılımda oluşabilecek hataları engellemeye ya da

oluştuysa sebebini kolayca bulmaya yarayan pratikler bu bölümde anlatılacaktır.

3.3.1. Kolay test edilebilir ve güvenilir bir yapı kurma

Burada temel yaklaşım, bir projenin birden fazla YKB'sinde ya da daha önceki projelerde ortak olan yazılım parçalarının her seferinde tekrar geliştirilmesine engel olacak, ortak bir yerden kullanılmalarını sağlayacak bir yapı kurmaktır. Bizim geliştirdiğimiz sistemlerde bu işleme olanak sağlayan modüller, algoritma modülleri ile yazılım ve donanımlarla arayüzü sağlayan haberleşme sınıfları olmaktadır. Burada mimari açıdan bu işlemlerin nasıl yapıldığı kabaca anlatılacak, detaylar verilmeyecektir; daha çok, sağladığı yararlar açısından değerlendirilecektir.

Algoritma modülleri, kendi geliştirme süreçlerinde geliştirilip testleri yapıldıktan sonra kütüphane haline getirilirler. Bu kütüphaneyi kullanacak olan YKB'de bu algoritma modülünün nasıl kullanılacağı ile ilgili arayüz sınıfı yazılır. Bu sınıf da, aynı algoritmaya ihtiyaç duyan YKB'ler tarafından kullanılmak üzere, ortak bir yerde, referans tasarım üzerinde geliştirilir. Bir algoritma üreteci yapısı ile YKB'nin ihtiyaç duyduğu algoritma modülleri çalıştırılır.

Bu sayede, bir algoritma modülü birden fazla YKB'de aynı şekilde kullanılabilmektedir. Kendi başına test edilmiş ve muhtemelen daha önce başka bir YKB'de kullanılmış algoritma modülleri ile çalışıldığı için YKB'nin güvenilirliğine katkı sağlanmış olur.

Bu yöntem, aynı amaçlı farklı algoritma modülleri arasında kolaylıkla geçiş yapmayı da sağlar. Sistemin nihai halinde olması gereken algoritma modülü geliştirilmeye devam ederken, aynı amaca hizmet eden, örneğin hazır alınmış bir algoritma modülü, gerçeğinin yerine kullanılabilir. Bu şekilde elimizde sürekli olarak çalışan bir sistem olur. Gerçek algoritma tamamlandığında da, iyileştirilmiş bir algoritmaya sahip bir sistemimiz olur.

Ortak kullanıma elverişli bir diğer modül türü ise yazılım ve donanımlarla arayüzü sağlamak amacıyla, fiziksel haberleşmeyi gerçekleyen sınıflardır. TCP/IP haberleşmesini gerçekleyen haberleşme protokolü sınıfı buna bir örnektir. Bu sınıflar da bir referans tasarım üzerinde gerçeklenir ve ihtiyacı olan YKB tarafından kolaylıkla kullanılabilir yapıdadır.

3.3.2. Test Altyapıları

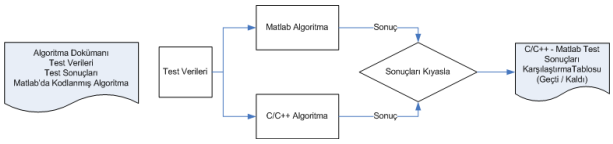
Simulatörler ile öngörülebilen her türlü ortam oluşturulmasına ve yazılımların kara kutu ve beyaz kutu testlerinin yapılmış olmasına rağmen gerçeklenemeyen durumlar olması ihtimaline karşı bir takım "hata yakalama" ve "hata durumunda kendini toparlama" yapıları kurulmaktadır.

1. Hafıza Bölümlemesi

Aynı hafıza alanını kullanan yazılım modüllerinden birinde oluşan hata (exception) diğer modüllerde, hatta işletim sisteminde de hata oluşmasına yol açabilir. Bunun oluşmasını engellemek için, işletim sistemleri hafıza bölümlemeye imkân tanımaktadırlar. Bu bölümleme, hem işletim sistemini yazılım modüllerinden, hem de yazılım modüllerini birbirinden ayırabilmektedir. Ancak bu işin maliyeti performans kaybıdır. Bu yüzden, performans kısıtı olmayan algoritmalar için kullanılmaktadır. Yapılan ölçümlerde, algoritmaların mahiyetine göre değişmekle beraber ortalama %25'lik bir performans kaybı göze alınmalıdır.

2. Algoritma Modülleri İçin Otomatik Test

Algoritma modülleri öncelikle bir simulasyon ortamında geliştirilir ve hazırlanan test girdileri ile testleri yapılır. Daha sonra C veya C++ dilinde kodlanarak gerçek ortama aktarılması sağlanır. Bu duruma uymayan algoritma modülleri de olabilir. Onlar simulasyon ortamında geliştirilmeden, bir algoritma dokümanı referans alınarak yazılıma aktarılır. Öncelikle bir simulasyon ortamında geliştirilen algoritma modülleri için şöyle bir test tekniği uygulanmaktadır:



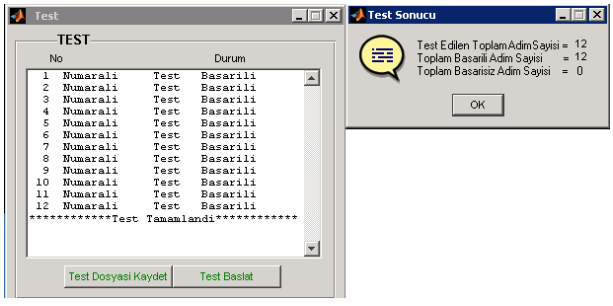
Şekil 5: Algoritma Otomatik Testi.

Algoritma dokümanı, algoritmanın Matlab'da kodlanmış hali, bu kodun test edildiği test verileri ve sonuçları, algoritma geliştiricilerden temin edilir.

Matlab'daki koda, algoritmanın belli aşamalarındaki sonuçları saklayan eklemeler yapılır. C/C++'da geliştirilen koda da aynı aşamalardaki sonuçları saklayan eklemeler yapılır. Bu sonuçları karşılaştıran bir yazılım yardımıyla, her aşama için Geçti/Kaldı bilgisi üretilir.

Sonuçları kıyaslama yöntemi algoritma yazılımının yeni sürümleri çıktığında da kullanılır. Aynı test girdileri için eski sürümde alınan sonuçlar ile yeni sürümde alınan sonuçlar karşılaştırılır ve yeni sürümdeki ekleme veya değişikliklerin eski gerekleri bozmadığı test edilir. Amaç, yazılımların yeni sürümleri çıktıkça, daha önceki sürümlerde yapılan testleri kısa sürede, otomatik olarak tekrarlayabilmektir.

Yazılımın yeni sürümü aynı test girdileriyle çalıştırılarak bahsi geçen kayıtlar tekrar alınır. Bir test yazılımı aracılığı ile bu sonuçlar olması gereken sonuçlarla otomatik olarak karşılaştırılır ve yazılımın yeni sürümü için test adımlarına ait başarılı/başarısız bilgisi üretilir.



Şekil 6: Otomatik Test.

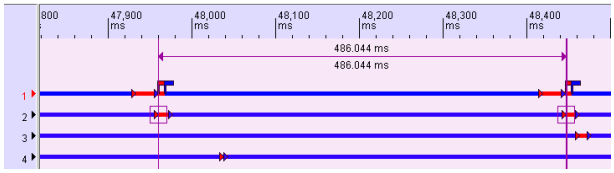
Şekil6'da, otomatik test gerçekleştiren örnek bir test yazılımı gösterilmektedir.

Otomatik testler daha çok, belli girdiler için belli sonuçlar üreten algoritma modülleri için uygulanabilmektedir.

3. Test Noktaları

Yazılımların, "belli zamanlarda belli işleri yap" türündeki gerekleri "test noktaları" kullanılarak doğrulanabilir. Bu gereklere, "300 ms.'de bir 50 ms. boyunca X işini yap", "istenen periyotlarda Y işini yap", "X ve Y işi zamanda çakışırsa Y işini seç" gibi örnekler verilebilir. Bu tür gerekler aşağıdaki adımlar uygulanarak doğrulanabilmektedir:

- Hangi zamanlarda yapıldığını ve ne kadar sürdüğünü bilmek istediğimiz işlevler için numaralar belirlenir.
- Yazılıma, bu işlevlerin başlangıcında, bitişinde ya da istediğimiz ara adımlarında, yazılımın çalıştığı işlemcinin saatini yüksek çözünürlükle okuyacak kod parçaları eklenir.
- Okunan zaman değerleri, işlevlere atadığımız belirleyici numaralar ile ilişkilendirilerek, işlemcinin hafızasında bu iş için ayrılmış bir alana yazılır.
- Yazılımda, bir komut ile hafızadaki verileri dosyaya aktaran bir işlev gerçeklenir; veriler incelenmek istendiğinde bu komut çalıştırılır.
- Oluşturulan dosya, görsel bir yazılım ile okunarak kolay incelenebilir hale getirilir.



Şekil 7: Test Noktaları.

Test noktalarını kolayca görebileceğimiz ve inceleyebileceğimiz bir yazılım Şekil7'de gösterilmektedir.

Çalışma sırasında, sadece işlemci saatini okuma ve hafızada bir alana yazma işlemi yapıldığı için ve bu işlemler çok kısa sürdüğü için, yazılımın performans gereklerini etkilememektedir. Oysa bu iş için kullanılacak ticari bir yazılımın daha fazla süreye ve hafızaya ihtiyaç duyması kaçınılmaz olacaktır.

İşletim sistemlerinin zaman analizi yapan araçları, işletim sisteminin tanıdığı "task" lar cinsinden bu analizi gerçekleştirmektedir. Ancak bizim ihtiyacımız, yazılımın işlevleri cinsinden bu işlemi gerçekleştirmektir; bir işlev için 1'den fazla "task" kullanılabilir.

4. Veri Kaydı ve Geri Oynatma

Sistem projelerinin gerekleri arasında, genellikle, sistemin çalıştığı sürece ne tür verilere maruz kaldığı ve bunlara ne tür sonuçlar ürettiği ile ilgili bilgilerin kaydedilmesi gereği vardır. Bu bilgiler ışığında, görev sonrasında sistemin durumu analiz edilir. Bu yetenek, hata durumlarında, hatayı lokalize etme amacıyla da kullanılmaktadır. Ana yazılıma entegre edilen algoritma modülleri, sistemde kaydedilen verilerle kendi başlarına çalışabilecek şekilde geliştirilimektedir. Böylece, görev sonrasında, görev boyunca kaydedilen verilerle algoritmalar çalıştırılmakta ve bulduğu sonuçlar ile sistemde bulunan sonuçlar karşılaştırılmaktadır. Eğer aynıysa ve sonuç hatalı ise algoritmada hata olduğu sonucuna varılmaktadır. Aksi takdirde hata, algoritmayı çalıştıran YKB'dedir

5. Donanım Birim Testleri

Sistemde bir hata oluştuğunda, bu hatanın yazılımla mı yoksa donanımla mı ilgili olduğuna karar vermek zordur ve bu kararı verme görevi genellikle yazılıma düşer. Sistem testleri sırasında karşılaşılabilecek bu duruma önlem almak için, donanımların, mümkün olduğu kadarıyla, tüm davranışlarını test edecek, görev yazılımı dışında bir yazılım geliştirilmektedir; buna birim test yazılımı adı verilir. Bu yazılım sık değişen bir yazılım değildir. Donanım mühendisleri ile birlikte, hem birim test yazılımının hem de donanımların doğruluğu test edilmekte ve bu yazılım değiştirilmeyerek gerektiğinde kullanılmak üzere saklanmaktadır.

4. Tartışma

Sistem gerekleri net bir şekilde belli olmadan yazılım geliştirmeye başlamanın verimli bir yaklaşım olmadığı düşünülebilir. Çünkü sistemde kullanılmama olasılığı olan yazılımlar üretme riski vardır. Ancak, bizim tecrübelerimiz ve literatürdeki çevik geliştirme yöntemleri [4], bunun bir kayıp olmadığını, kazandırdıkları yanında göze alınabilecek bir durum olduğunu söylemektedir.

5. Sonuçlar

Gömülü yazılımların geliştirilmesi sürecinde yaşanan zorluklar göz önünde bulundurularak, bu zorluklara karşı alınabilecek önlemler, gömülü yazılım geliştirme pratikleri şeklinde aktarılmaya çalışılmıştır. Bu önerilen yöntemlerin tümü hali hazırda yürüyen bir projede kullanılmakta ve faydaları görülmektedir.
Literatürde, bahsi geçen problemler, çevik yazılım geliştirme kapsamında adreslenmektedir ancak somut çözüm önerilerine ulaşmak zordur. Bu bildiride, gömülü sistem ve yazılım geliştirme kapsamında elde edilen tecrübeler paylaşılmaya çalışılmış, somut önerilerle benzer projelere yardımcı olunması hedeflenmiştir.

6. Teşekkür

Bu bildiride anlatılan yazılım geliştirme pratiklerini kusursuz şekilde uygulayan yazılım geliştirme ekibine teşekkür ederiz.

7. Kaynakça

[1] Greening, J., "Agile Embedded Software Development" : San Jose, April 2007, ESC Class# 349
[2] Yılmazer, Z., "Aselsan MST Grubu Yazılım Kalite Güvencesi Faaliyetleri", UYMS 2007
[3] Greening, J., "Extreme Programming And Embedded Software Development", Articles Directory – Extreme Programming (XP) : Software Development Articles
[4] Beck, K., Extreme Programming Explained: Embrace Change, Addison Wesley Professional Publishers, 2000.