Yazılım Test Simülatörleri Geliştirilmesinde Yazılım Ürün Hattı Yaklaşımının Kazanımlarına Dair Bir Eylem Araştırması

M. Erdem ERGÜL[1] H. İbrahim BALCI[2]

[1, 2] Radar Elektronik Harp ve İstihbarat Sistemleri (REHİS) Grubu, ASELSAN A.Ş. Ankara

[1]eergul@aselsan.com.tr, [2]ibalci@aselsan.com.tr

Özet. Yazılım Ürün Hatları (YÜH) yaklaşımı, benzer yazılım ürünlerini bir bütün olarak ele almakta, yazılım ürünlerinin farklarını ve ortak yönlerini açıkça ortaya koyarak değişkenliği üst seviyede yönetilebilir kılmaktadır. Bu yaklaşım yazılımların kalitesini arttırmakta, maliyetlerini düşürmekte ve nihai ürünün daha hızlı şekilde hazırlanmasını sağlamaktadır. Bu çalışmada öncelikli olarak radar projelerinin yazılım testleri kapsamında geliştirilen simülatör yazılımlarının yazılım geliştirme faaliyetlerinde karşılaşılan sorunlar ele alınmıştır. Bu sorunların giderilmesi için YÜH yaklaşımı önerilmiştir. Simülatörler arasındaki benzerliklerin fazla olmasının ürün hattının kazanımını arttırdığı görülmüştür. Yeniden kullanım oranı iyileşmiş, işçilik maliyetleri azalmış, ürün kalitesi yükselmiş ve simülatörler arasında görünüm benzerliği sağlanmıştır. Bu bildiride yazılım test için yazılım geliştirme alanında YÜH yaklaşımının uygunluğu ve edinilen kazanımları, yapılan ölçüm ve analizlerle incelenmiştir.

Abstract. Software Product Line approach enables improved variability management by handling similar software products as a whole and expressing commonalities and variabilities implicitly which leads to high software reuse levels. This approach increases product quality, decreases costs and time-to-market. In this work, firstly, problems encountered in software development efforts performed in scope of simulator development for software testing of RADAR projects are discussed. Later, SPL approach is proposed to deal with these problems. It is observed that having much similarities between software products enabled increased SPL benefits. Reuse levels and quality of products are improved, costs are reduced and common look-and-feel is ensured. In this paper, convenience of SPL approach to the software development for software testing and its benefits are examined by measurements and analysis.

Anahtar Kelimeler: Yazılım Test, Yazılım Ürün Hattı, Simülatör, Yazılım Geliştirme İyileştirmesi

1 Giriş

Yazılım pazarı her geçen gün biraz daha büyümekte ve üreticiler artan rekabete ayak uydurabilmek için ürün yelpazelerini genişletme çabası içine girmektedirler. Bu durum YÜH yaklaşımının da önemini artırmaktadır. YÜH, belirli bir amaç veya pazara yönelik ihtiyaçları karşılayan yüksek benzerlikteki yazılım bileşenleri grubudur [1]. Klasik yazılım geliştirme yaklaşımında, fırsatçı (oppurtunistic) yeniden kullanım hakimken, yazılım ürün hatlarında ise önceden planlanmış (predictive) yeniden kullanım yaklaşımı hakimdir.

Literatürde YÜH kazanımlarının şu başlıklarda toplandığı görülmektedir: Yüksek yeniden kullanım oranı [2, 3], düşük maliyet ve işçilik [4, 5], planlama ve markete çıkış hızında iyileşme [6, 4], yüksek ürün kalitesi [7, 2], yüksek ürün çeşitliliği ve rekabet gücü [8, 9], bakım ve değiştirilebilirlikte iyileşme [2, 10]. Fakat YÜH uyarlaması her yazılım ürünü için uygulanabilir değildir. Öncelikle uygun bir market ve ürün ailesinin belirlenmesi gerekmektedir. Birbirine yakın özellikteki ürünlerden oluşan ürün ailesindeki yazılımlar arasındaki benzerlik ve farklılıklar açıkça ortaya konulup değişkenliğin en üst seviyede yönetilebilmesi sağlanmalıdır. Bu durum ürün hattından çıkan ilk yazılımlar ele alındığında daha fazla efor harcanmasına neden olsa da uzun vadede ürün hattından çıkan ürünlerin gerektirdiği toplam iş gücü önemli ölçüde azalmaktadır [1].

Yazılım test faaliyetleri sırasında ihtiyaç duyulan simülatör yazılımları geniş bir yelpazede fazlaca ortaklık barındıran ürün çeşitliliği gerektirmektedir. Bu ürünler arasındaki ortaklık ve farklılıkların daha üst düzeyde yönetilmesini sağlamak amacıyla yazılım ürün hattı yaklaşımının uygun olacağı öngörülmüştür.

Bu bildirinin 2. bölümde REHİS Yazılım Test Mühendisliği bölümünde geliştirilen yazılımların özellikleri ve bu yazılımları hazırlama aşamasındaki zorluklardan bahsedilecek, 3. bölümde YÜH yaklaşımının nasıl uygulandığı anlatılacaktır. 4. bölümde ise ölçüm ve analizler ortaya konulacak ve 5. bölümde yapılan çalışma değerlendirilecektir.

2 Yazılım Test için Yazılım Geliştirme

ASELSAN REHİS Sektörü Yazılım Test Mühendisliği Müdürlüğü'nde kara kutu test yöntemi ile yazılım ürünleri fonksiyonel gereksinimler üzerinden test edilmektedir. Yazılım gereksinimlerinin çoğunun doğrulanması için yazılım test simülatörlerinin kullanılması gerekir ve bu yazılımlar da yazılım test mühendisleri tarafından geliştirilir. Yazılım gereksinimleri genellikle sistemdeki diğer birimlerle yapılan mesajlaşma senaryolarını içerir. Bu mesajlaşmaların doğru şekilde test edilebilmesi için yazılım konfigürasyon biriminin (YKB) haberleştiği yazılım veya donanım konfigürasyon birimlerinin (DKB) benzetimi hazırlanan simülatörler yoluyla yapılır. Bu arayüzlerde haberleşme seri kanal, ethernet, PCI-Express veya MIL-STD-1553 gibi protokoller üzerinden gerçekleştirilir.

Geçmiş tecrübelerden yararlanarak her yazılım ürününün birbirinden bağımsız olarak ele alınıp, geliştirildiği tekil sistem yazılım geliştirme (TSYG) yönteminde karşılaşılan sorunlar şu şekilde belirlenmiştir:

- **Fazla Efor:** TSYG yönteminde geliştirilen yazılımlarda geliştiriciler tarafından farklı geliştirme yöntemleri kullanılması ve programlama stillerinin farklı olması nedeniyle yazılımların geliştirilmesi, test edilmesi, uygulanması ve idame edilmesine fazladan efor harcanması gerekiyordu.
- **Kalite:** Yazılım ürünleri diğer yazılımların testinde kullanıldığından ürün kalitesinin yüksek olması gerekmektedir. Simülatör yazılımının kullanıldığı bir test sırasında görülen uyumsuzluk simülatör kaynaklı olduğunda hatanın bulunması ek iş yükü getirmektedir ve genel yazılım geliştirme sürecinde gecikmeye yol açmaktadır.
- **Farklı Görünüm: S**imülatörlerin zaman zaman farklı test ve yazılım mühendisleri tarafından kullanılması ihtiyacı nedeniyle kullanılabilirliğini artırmak için benzer kullanıcı arayüzlerine sahip olması önemlidir. TSYG yöntemiyle simülatörlerin benzer bir görünüme sahip olması sağlanamamıştır.
- **Düşük Yeniden Kullanılabilirlik:** Tekil sistemde geliştirilen yazılımlarda yeniden kullanım oranı düşüktür. Yeniden kullanım ancak sistematik olmayan bir şekilde kod parçalarının kopyala-yapıştır yapılmasıyla sağlanmaktadır. Ayrıca bu kod parçalarının idamesi de fazladan efor gerektirmektedir.

3 Yazılım Geliştirmede Yapılan İyileştirmeler

3.1 Önceki Çalışmalar

Yazılım Test Mühendisliği Bölümünde simülatör yazılımlarının altyapısını geliştirmek için daha önceden yapılan çalışmalar kapsamında COBALT (Communication Basics Library for Testing) çerçevesi (framework) oluşturulmuştur [11]. Bu çalışmada en önemli amaç yeniden kullanılabilirlik oranını artırmaktır. COBALT altyapısı, YÜH için miras sistem olmuştur ve yeniden kullanılabilir yazılım bileşenlerinin çoğu ve YÜH referans mimarisi bu çerçeveyi temel almıştır.

3.2 Yazılım Test Simülatörleri-Yazılım Ürün Hattı

İş-Merkezcil Yaklaşım.

YÜH kurulmadan önce hitap edilen pazarı doğru analiz etmek gerekmektedir. Bu çalışmada yazılımı geliştirenler ayrıca yazılımın müşterisi olduğundan diğer pazarlardan farklıdır. Geliştirilen simülatör yazılımları arasındaki benzerliklerin çok fazla olması ve ihtiyaçların önceden yüksek oranda belirli olması ürün portföyünün rahatça belirlenebilmesini sağlamıştır. YÜH ürün portföyü; bir haberleşme arayüzü üzerinden mesaj alıp gönderebilen, aldığı ve gönderdiği mesajları kayıt edebilen, bir kullanıcı arayüzü ile gönderilen mesajların parametrelerinin değiştirilmesine izin veren ve alınan ve gönderilen mesajları anında gösterebilen Java tabanlı yazılım test simülatör yazılımları olarak tanımlanmıştır.

Ürünler arasındaki benzerlikler ve farklılıklar net olarak ortaya konulabildiğinden ve ürünlerin hepsi aynı amaca hizmet ettiğinden kısa vadeli iş stratejisi olarak YÜH kurulumu avantajlı görülmüştür. Uzun vadede ise alan potansiyeli analizleri yapılarak yazılım test faaliyetlerinde kullanılan simülatör yazılımlarının çoğunun YÜH kapsamına alınması hedeflenmiştir.

Gereksinimlerin Belirlenmesi.

YÜH gereksinimleri, değişkenlik ve sınırlayıcı bağlılık etiketleri olarak ikiye ayrılmıştır. Gereksinimler belirtilirken, değişkenliğin gösterimi için etiketleme yöntemi kullanılmıştır:

Değişkenlik etiketleri aşağıdaki gibi kullanılmıştır:

- <Zorunlu>: Her yazılımda olması gereken gereksinimleri belirtir.
- <DN(X)_Seçenek(N)>: X numaralı değişkenlik noktası (DN)'nın N numaralı seçeneğini ifade eder. İsteğe bağlı olarak uygulamalarda yeniden kullanılabilecek bileşenlerdir.
- <DN(X)_Alternatif(N)> : X numaralı değişkenlik noktası (DN)'nın N numaralı alternatifini ifade eder. Uygulama mühendisleri var olan alternatiflerden birini seçerek yeniden kullanmalıdır.

Sınırlayıcı bağlılık etiketleri aşağıdaki gibi kullanılmıştır:

- <DN(X)_Seçenek(N):Gerektirir_DN(Y)_Seçenek(M)>: Eğer X değişkenlik noktasına bağlı N seçeneği varsa Y değişkenlik noktasına bağlı M seçeneği de olmalıdır.
- <DN(X)_Seçenek(N):Dışlar_DN(Y)_Seçenek(M)>: Eğer X değişkenlik noktasına bağlı N seçeneği varsa Y değişkenlik noktasına bağlı M seçeneği olmamalıdır.

Referans Mimari.

YÜH mimarisi MVC (Model-View-Controller) mimarisine benzer yapıdadır [11]. Her soyutlama katmanı kodlama detaylarını bir diğerinden saklayacak şekilde hazırlanmıştır. YÜH yeniden kullanılabilir bileşenleri kullanıcı arayüzü ve veri katmanlarında yer alır. Haberleşme ve kontrol katmanları projeye özel olarak hazırlanır. Haberleşme katmanında seri kanal, Ethernet ve MIL-STD-1553 gibi haberleşme standartları kodlanmaktadır. Veri katmanı mesajları giriş ve çıkış akıntıları (input and output streams) üzerinden ve haberleşme katmanının kodlama detaylarından bağımsız olarak alır/gönderir.

Veri katmanında mesaj sınıfları veri ağacı gösterimi (VAG) şeklinde tanımlanır. Mesajların yapısı ürüne özel olduğundan uygulama mühendisi tarafından tanımlanır. Bu katmanda yeniden kullanım, kalıtım adaptasyonu mekanizmasıyla sağlanır. Uygulama mühendisinin COBALT'tan miras alınan soyut mesaj sınıflarını türeterek oluşturduğu mesaj sınıfları, içerdiği parametreleri haberleşme kanalına gönderip alabilmek için mesaj çözümleme ve mesaj oluşturma yeteneklerine otomatik olarak sahip olur.

COBALT'tan miras alınan yeniden kullanılabilir kod bileşenleri olan kayıt tablosu ve ağaç gösterimi kullanıcı arayüzü katmanında yer alır. Ağaç gösterimi, VAG formundaki mesajın ağaç yapısında gösterilmesini ve parametrelerinin kullanıcı tarafından çalışma zamanında değiştirilebilmesini sağlar. Kayıt tablosu ise gönderilen ve alınan mesajları saklayıp çalışma zamanında karşılaşılan tüm olayların zaman ve kaynak bilgisi ile listelenip kullanıcıya sunulmasını sağlar [11].

Değişkenlik noktalarının belirlenmesi YÜH kurulumunun önemli bir safhasıdır. Bu çalışmada ürün hattının ilk değişkenlik noktasını veri katmanında yer alan mesaj sınıfları oluşturur. Olası her mesaj yapısı bu değişkenlik noktasının opsiyonları olarak tanımlanmıştır. Kullanıcı arayüzü kullanımı YÜH'ün otomatikleştirilmiş test yapabilecek yazılım ürünlerinde istenilmeyen bir özelliktir. Bu yüzden Ağaç Gösterimi ve Kayıt Tablosu bileşenlerinin kullanımı da isteğe bağlı değişkenlik noktaları olarak tanımlanmıştır.

Varlık Kapsama.

Miras sistemde [11] sadece ürünler arasındaki benzerlikler dikkate alınarak alan bileşenleri geliştirilmişti. YÜH yaklaşımı ile sadece zorunlu bileşenler değil değişken bileşen (variants) sayısı da arttırılmaya çalışılmıştır. Varlık kapsama (asset scoping) ile hedeflenen de YÜH altyapısındaki bileşen sayısının artırılmasıdır. Genel geçer bir kural olarak, 3 farklı üründe bulunan bir özelliğin ürün hattı yapısında yer almasına karar verilmiştir. Bu kapsamda haberleşme kanalında kullanılan protokollerden seri kanal ve Ethernet iki yazılım bileşeni olarak ele alınıp, haberleşme değişkenlik noktası şeklinde belirlenmiştir.

4 Ölçüm ve Analizler

Bu çalışmada gerçek dünyada kullanılan altı simülatör (A, B, C, D, E, F) ve kontrollü olarak geliştirilen dört simülatör (K, L, M, N) ele alınmıştır. A, B, C, K ve L simülatörleri TSYG yöntemi ile geliştirilmiştir. D, E, F, M ve N simülatörleri YÜH yaklaşımı ile geliştirilmiştir. A, B, C, D, E ve F simülatörleri seçilirken benzer yeteneklere sahip olmalarına önem verilmiştir. Tüm simülatörler aynı kişi tarafından geliştirilmiştir. A, B ve C simülatöründe C# programlama dili, diğer üçünde ise Java programlama dili kullanılmıştır. K, L, M ve N simülatörlerinin hepsi kısa zaman içerisinde aynı geliştirici tarafından ve Java programlama dili ile geliştirilmiştir. K ve M simülatörünün, L ve N simülatörlerinin mesaj yapıları ve gereksinimleri aynıdır. Bu simülatörler için yeniden kullanım oranı, uygulama/idame eforları, hata yoğunluk oranı ve müşteri memnuniyeti ölçülerek nicel ve nitel veriler toplanmıştır.

- **Yeniden Kullanım Oranı**

Yeniden kullanım oranı; yeniden kullanılan kod satır sayısının toplam kod satır sayısına bölünmesi ile hesaplanmıştır. Bu işlem için yeniden kullanım seviyesi hesaplama aracı geliştirilmiştir. Araç her üç uygulamada yeniden kullanılmış olan her

bir satır kodu yeniden kullanılmış kabul etmektedir [12,13]. Ulaşılan sonuçlar Tablo 1'deki gibidir;

Tablo 1. Yeniden Kullanım Oranı ve Uygulama Eforu

Simülatör Adı	Geliştirme Yöntemi	Kaynak Kodu Yeniden Kullanım Oranı	Kod Satır Sayısı	Uygulama Eforu (Adam.Saat)
A	TSYG	%6	4161	55
B	TSYG	%14	2176	30
C	TSYG	%22	951	17
K	TSYG	%55	2655	11.73
L	TSYG	%79	1627	1.5
D	YÜH	%85	9633	13
E	YÜH	%89	9213	8.5
F	YÜH	%94	8752	8
M	YÜH	%96	9298	2.58
N	YÜH	%97	9150	0.38

- **Uygulama Eforu**

Her bir simülatörün geliştirilmesi için harcanan zaman ile uygulama eforu ölçülmüştür. Sonuçlar Tablo 1'deki gibidir [12]. Kontrollü deneylerdeki (K, L, M, N) ölçülen uygulama eforlarının küçük çıkması kodlanan mesaj sayısının azlığından kaynaklanmaktadır. Gerçek uygulamalarda kodlanan mesaj sayısı bu örneklerin yaklaşık 40-50 katına çıkabilmekte ve uygulama eforu da bu durumla bağlantılı olarak büyük miktarda artış göstermektedir.

- **İdame Eforu**

Geliştirici haricinde iki uygulama mühendisinden K, L, M ve N simülatörlerine yeni özellikler eklemeleri ve mesaj yapılarını değiştirmeleri istenmiştir. Her biri için harcanan zaman Tablo 2'teki gibidir [12].

Tablo 2. İdame Eforu

Simülatör Adı	Geliştirme Yöntemi	Uygulama Mühendisi 1'in İdame Eforu (dakika)	Uygulama Mühendisi 2'nin İdame Eforu (dakika)	Hata Sayısı	Hata Yoğunluğu (Hata sayısı / (1000x Kod Satır Sayısı))
K	TSYG	33	47	12	4.5
L	TSYG	29	75	4	2.45
M	YÜH	9	30	2	0.215
N	YÜH	26	45	2	0.218

Ayrıca 4 uygulama mühendisi ile yapılan ankette katılımcıların hepsinin TSYG yöntemi ile geliştirilen bir yazılımın gereksinimlerinin %50'si değiştiğinde, yazılımı sıfırdan yazmayı tercih edeceklerini belirtmişlerdir. Yine aynı katılımcılar YÜH

yaklaşımı ile hazırlanan bir yazılımın gereksinimlerinin %50'si değiştiğinde veya mesaj yapılarının %50'si değiştiğinde, yazılımı güncellemeyi tercih etmektedirler.

- **Hata Yoğunluk Oranı**

K, L, M ve N yazılımlarının sistem testlerinde karşılaşılan hata sayıları ve bu hataların toplam kod satır sayısına bölünmesiyle elde edilen hata yoğunluk oranları Tablo 2'de verilmiştir [12].

Ayrıca geliştiricilerle yapılan ankete göre geliştiricilerin büyük çoğunluğu YÜH yaklaşımı ile hata sayısında azalma olduğu fikrine katılmaktadır.

- **Müşteri Memnuniyeti**

Üç kullanıcıya müşteri memnuniyetini ölçmek amacıyla SUS (System Usability Scale) anketi [12] uygulanmıştır. Alınan sonuçlar YÜH yaklaşımının kullanılabilirliği artırdığını göstermektedir.

Tablo 3. Simülatörlerin Kullanılabilirliği

Yazılım Geliştirme Yaklaşımı	Ortalama SUS Skoru
Tekil Sistem (K ve L)	51
Ürün Hattı (M ve N)	91

Ayrıca katılımcıların hepsi YÜH kurulumu ile simülatörlerin benzer görünüme sahip olduğunu, bunun da kullanım kolaylığını arttırdığını belirtmiştir. Ayrıca katılımcılar YÜH ile beraber simülatörlerinin kalitesinin arttığından bahsetmiştir. Bunun en önemli nedeni azalan hata sayısıdır.

5 Sonuç

Bu çalışmada Aselsan'ın REHİS Yazılım Test Mühendisliği bölümünde gerçekleştirilen YÜH deneyimi üzerine bir eylem araştırması anlatılmıştır. Araştırmanın amacı yazılım test simülatörlerinin geliştirilmesinde YÜH yaklaşımının uygunluğunun incelenmesidir. Bunun için ilk olarak geleneksel tekil sistem geliştirme yöntemlerinde karşılaşılan sorunlar; aşırı iş yükü, daha yüksek kalite ihtiyacı, heterojen görünüm ve düşük yeniden kullanılabilirlik olarak belirlenmiştir. Karşılaşılan bu sorunlara çözüm olarak COBALT altyapısı miras alınarak genişletilmiş ve Yazılım Test Simülatörleri Yazılım Ürün Hattı (YTS-YÜH) kurulmuştur. Kısa ve uzun vadeli iş stratejisi ve YÜH ürün portföyü tanımlanmıştır. YTS-YÜH gereksinim özellikleri, referans mimarisi ve değişkenlik modeli belirlenmiştir. Daha sonra varlık kapsamı çalışmaları ile YÜH geliştirilmiştir. Kullanılan etiketleme sistemi ile gereksinimlerin yeniden kullanımının daha kontrollü ve kolay olması sağlanmıştır. Yapılan ölçüm ve analizler ile YÜH yaklaşımının önemli ölçüde daha az işçilik, daha yüksek ürün kalitesi, homojen görünüm ve daha yüksek yeniden kullanılabilirlik oranları sağladığı görülmüştür. Uzun vadede daha

fazla kontrollü deney yapılarak YÜH kazanımları üzerine nicel veriler toplamak hedeflenmektedir. Ayrıca ürün hattı yönetimi için araç desteğinin önemi düşünülerek yeniden kullanılabilir değerlerin yönetimi için araç desteği sağlanması, YTS-YÜH'ün otomasyon yeteneği ile genişletilmesi ve test verilerinin otomatik oluşturulması için veri oluşturucu bir bileşen geliştirilmesi hedeflenmektedir.

6 Kaynaklar

1. "http://www.sei.cmu.edu.tr/productlines/index.cfm", Erişim tarihi: 07.05.2015

2. F. v. d. Linden, K. Schmid, E. Rommes, "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering" Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

3. P. Clements, J. K. Bergery, "The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study" Technical Report, 2005.

4. L. Liang, Z. Hu, X. Wang, "An Open Architecture for Medical İmage Workstation", sayfa 470-479, 2005.

5. A. Jaaksi, "Developing Mobile Browsers in a Product Line," IEEE Software 19, 4, July/August, 2002, p 73-80. Institute of Electrical and Electronics Engineers Computer Society.

6. S. Cohen, E. Dunn, A. Soule, "Susccesful Product Line Development and Sustainment: A DOD Case Study" Technical Report, Carnegie Mellon University, 2002.

7. K. Pohl, G. Böckle, F. v. d. Linden, "Software Product Line Engineering", Springer 2005, ch. 21.

8. L. Brownsword, P. Clements, "A Case Study in Successful Product Line Development" Technical Report, Carnegie Mellon University, 1996.

9. M. Vierhauser, M., Holl, G., Rabiser, R., Grünbacher, P., Lehofer, M., Stürmer, U. Sturner, "A Deployment Infrastructure for Product Line Models and Tools," Proceedings SPLC 2011, Munich, pp. 287-294.

10. D. Pech, J. Knodel, R. Carbon, C. Schitter, D. Hein, "Variability Management in Small Development Organizations – Experiences and Lessons Learned from a Case Study," Proceedings SPLC 2009, San Francisco, August 2009.

11. U. Zöngür, S.T. Erdoğan, "Cobalt: Test Uygulamaları için Protokol Kütüphanesi", UYMS'14

12. M. E. Ergül, "An Action Research of Achievements in a Software Product Line Implementation", M. Sc. Thesis, METU, Ankara, 2014.

13. W. Frakes and C. Terry, "Software Reuse: Metrics and Models." ACM Computing Surveys 28(2), pp. 415-435, 1996.