Bir Yazılım Kalitesi Olarak Evrilebilirlik

Evolutionary Adaptability as a Software Quality

Mehmet Gençer
Bilgisayar Bilimleri Bölümü
İstanbul Bilgi Üniversitesi, İstanbul
mgencer@cs.bilgi.edu.tr

Özet

Geniş ve heterojen bir kullanıcı kitlesi olan yazılım sistemlerinde gereksinimlerin tespiti zor olduğu gibi bu gereksinimler değişken olabilir. Bu durumda kapsamlı ve planlı bir geliştirme yerine dağıtık bir geliştirme ve yazılımın ortaya çıkan gereksinimler göre hızlı evrilebilmesi önem kazanır. Bu makalede güncel tecrübeler üzerinden bu gereksinimlerin nasıl karşılandığı incelenmektedir. Çok paydaşlı dağıtık geliştirme ile evrilebilirliğin birleştirilmesi için gevşek eşleşme önemli bir unsurdur. Güncel modeller içerisinde özellikle açık kaynak yazılım modeli bunu sağlamakta başarılı olmuştur. Buna karşılık hem gevşekliğin hem de açık kaynak modelinin bazı sakıncalarını giderme yolunda yazılım piyasasındaki deneyimlerden ortaya çıkan melez modeller önem kazanmaktadır.

Abstract

In software systems which have a wide and heterogeneous user base, not only it is difficult to determine the requirements but also these requirements are volatile. In such a case it becomes important to replace a planned development with distributed development and evolutionary agility. In this paper, we investigate how these challenges are addressed with a review of contemporary examples. Loose coupling is an important element to combine multi-stakeholder distributed development with evolutionary agility. Among contemporary development models, open source model has been particularly successful to satisfy these requirements. On the other hand, hybrid models that emerge from software market's experiences become important to overcome certain difficulties of both loose coupling and open source model.

1 Giriş

İstisnai durumlar bir yana yazılım geliştirme süreçlerinde gereksinimlerin sabit olduğu varsayımı geçerliliğini yitirmiştir. Gereksinimlerin ve hedeflerin kesin olmadığı bir yazılım geliştirme süreci ise mühendislik pratiğine ilaveten tasarım pratiği ile de benzerlikler sergiler [1 ,2]. Bundan dolayı da burada eski model olarak adlandıracağımız, 'gereksinim analizi', 'gerçekleştirme', 'test ve düzeltme' aşamalarına dayalı yazılım geliştirme modelinin uygulanabilirliği kalmamıştır. Gereksinimlerin görece az değişken olduğu durumlarda eski 'aşamalı' geliştirme modeli etkin bir şekilde uygulanabilir. Belirli aralıklarda model aşamalarının tekrar edilmesi suretiyle yeni yazılım sürümlerinin yavaş değişen gereksinimlere uyarlanması sağlanır. Ancak yazılım projelerinin çoğunda değişkenlik yüksektir.

Buna karşılık, gitgide yaygınlaşmakta olan ve burada yeni model olarak adlandıracağımız geliştirme modelinde ise değişmez bir hedefe doğru ilerleme yerine değişen koşullara uyum sağlama, yani evrilebilirlik esas alınmaktadır. Yeni modelde detaylı bir plan ve bunu izleyen doğrusal ilerleyen bir geliştirme süreci yerine biyolojik evrimdeki doğal seçimi andıran, çeşitlenme ve seçim çevresinde ilerleyen sarmal bir süreç yeralmakta, ve yazılım mimarisi tasarımı bu sürecin sağlıklı ilerlemesini gözeterek yapılmaktadır.

Modülerlik eski modelde olduğu gibi yeni modelde de geliştirme verimliliği için esastır. Ancak eski modeldeki planlı bir işbölümünü sağlamak için hiyerarşik bir mimari çerçevesinde uygulanan ve sıkı eşleşme esasına dayalı modülerlik yerine, yeni modelde dağıtık bir mimari, gevşek eşleşmeye dayalı bileşen bağımsızlığı, ve bileşen evrilebilirliğini amaçlayan bir modülerlik tarzı yeralmaktadır.

Bu çalışmada güncel örnekler üzerinden yeni

geliştirme modelinin temel özellikleri ve eski modelden farkı ortaya konulmaktadır. Özellikle büyük yazılım sistemleri, kullanım ve geliştirme koşullarının heterojen olduğu durumlarda, gereksinimlerin öngörülemez ve değişken, ve evrilebilirliğin elzem olduğunun altı çizilmektedir. Eski modelin çevik geliştirme türü yeniliklerle değişken durumlara uyarlanması denendiyse de, iddiamız dağıtık geliştirme ve evrilebilirlik için açık kaynak ve kamusal lisansların yeni modelin gerçekleştirilmesi için bilinen en iyi strateji olduğu yönündedir. Ele alınan örneklerin bir kısmı yazılım firmalarının bu modeli nasıl ticari olarak uyguladıklarını göstermektedir.

Genel olarak büyük, öngörülemez, ve çok paydaşlı geliştirme süreçlerinin ve bu süreçlerle üretilen yazılımların mekanik değil organik bir bakışa dayalı stratejilerle yönetilmesi sözkonusudur. Bakış açısındaki ve uygulamadaki bu kayma hem geliştirme süreçlerini hem de yazılım kalitesine bakışı ve iş stratejilerini esastan değiştirmektedir.

2 Dinozorların sonu

Frederick Brooks [3] 1970lerde IBM'in OS360 işletim sistemini geliştirmesinin hikayesini aktarırken yazılım geliştirme süreçlerini ölçeklemenin sıkıntılarını büyük bir canlılıkla anlatır. Sözü edilen projede -benzer projelerde sıklıkla görüldüğü gibi- işler sürekli sarkmakta ve gecikmeyi telafi etmek için projeye personel eklenmesi ise "yangına gaz dökmek" etkisi yaratmaktadır. Bu tür büyük sistemlerde bileşenlerden birindeki problemin giderilmesi sistemin tamamın hakkında geniş bilgi gerektiren bir geliştirme problemi olmakta ve bu yüzden işler uzamaktadır. Sistem geliştirmesi modüllere ayrılmış bile olsa bu modüllerin sıkı eşleşmiş olması durumunda bahsedilen sakınca aynen devam eder.

Buna karşılık günümüzün işletim sistemlerinden olan Linux farklı bir yazılım geliştirme modelinin arketipi olarak ortaya çıkmaktadır. OS360'ın çağdaşı olan atası Unix'ten farklı olarak, Linux sistemin becerilerinin büyük kısmı sistemden ayrı modüllerde gerçekleştirilmiştir. Modüller sistemin çekirdeğini geliştiren ekipten farklı geliştiriciler tarafından geliştirilirler. Burada sabit olan tek şey modüllerin sistem çekirdeğiyle nasıl etkileşeceğini belirleyen arayüzdür. Modülde bir hata ortaya çıksa bile bu hata ne sistem çekirdeğinin ne de diğer modüllerin çalışmasını engelleyecektir.

Bileşenler arasındaki bu türden bir eşleşmeyi 'gevşek eşleşme' olarak nitelemek yerinde olacaktır. Bu tür bir eşleşmenin en önemli özelliği bileşenlerin birbirini hemen değil ancak nihai olarak, değişikliklerin kapsamının fazlaca genişlemesi durumunda etkilemesidir [4]. Linux örneğinde de hatalı modüller nihai olarak sistemin kullanışlılığını ve tercih edilirliğini azaltacaktır. Ancak gevşek eşleşme bu türden problemlerin maliyeti yüksek krizlere dönüşmeden çözülmesine imkan verir [5]. Linux'te modüllerin çekirdek ile bütünleşik olarak derlenmesi imkanı vardır ancak özelleşmiş, performans öncelikli uygulamalar dışında bu yöntem tercih edilmemektedir.

Benzeri bir yaklaşım yaygın olarak kullanılan Apache web sunucusunda da görülmektedir. Bu sunucunun ilk sürümünde sıkı eşleşmeye dayalı bir modülerlik vardı: sistem becerilerinin gerçekleştirildiği modüller sunucu ile birlikte derleniyor, tek bir çalıştırılabilir programa dönüşüyor ve modüllerden birindeki problem bütün sistemi etkiliyordu. Sistemin ikinci sürümünden itibaren Linux'tekine benzer bir modülarizasyona geçilmiştir. Apache sisteminde de aynı Linux'te olduğu gibi çok paydaşlı bir geliştirme ortamı sözkonusudur: birçok firma ve gayri ticari organizasyon yazılıma katkı sağlamaktadır. Her iki türden modülerlik bu katkıda işbölümüne imkan verir, ancak gevşek eşleşme geliştirme sürecine katılan paydaşlar arasındaki koordinasyon gereksinimini azaltmakta, daha gevşek ve yavaş işleyen bir koordinasyonla da dağıtık geliştirme yapılabilmesine imkan vermektedir.

Bu türden yaklaşımlar daha geç te olsa doğrudan ticari bir kurum tarafından liderlik edilen yazılım projelerinde de ortaya çıkmaya başlamıştır. Brooks'un [3] anlattığı dönemin tecrübeleri firmaları büyük monolitik dinozorlar yerine küçük bileşenlerin birikmesine ve eşleşmesine dayalı bir yönteme yönlendirmiş, ve bu yöntem firmalararası işbirliğinin yükseldiği günümüzde yazılım geliştirme pratiğinin yanısıra bu işbirliğine de uygunluğu ile önem kazanmıştır. Örneğin IBM'in çabalarıyla başlatılan ve şu anda birçok firmanın katkı verdiği Eclipse yazılım geliştirme platformu, uyumlu ek (İng. plug-in) denilen ve sistem çekirdeğiyle minimal bir arayüz vasıtasıyla eklemlenen bileşenlerden oluşmaktadır. Eclipse'teki bu mimari herbirinin farklı iş öncelikleri olan firmalar arasında sağlıklı işleyen, üretken bir ekosistem oluşturma amacını taşımaktadır [6]. Ekosistem metaforu Apache ve Linux gibi diğer sistemlere de çok uygun düşmektedir. Birbiriyle rekabet eden büyük dinozorlardan oluşan bir teknoloji pazarı anlayışının yerini işbirliğinin ve gevşek eşleşmenin ön plana çıktığı

ekosistemlerin birbiriyle rekabeti almış görünmektedir. Bu gözlem işletme yazınında da büyük şirketlerin hiyerarşik olarak yönetilmesiyle ilgili sıkıntılar ve şirketlerin ilişkisel konglomeralara benzeyerek büyümesi olgusu ile örtüşmektedir [7]. Bu tür bir yönetim tarzının egemen olduğu bilişim şirketleri projelerine de bu tarzı yansıtmaktadır. Örneğin Java pazarında IBM bu yeni tarz ve işbirliğine dayalı stratejisi ile başarılı olurken, eski tarzda devam eden Sun Microsystems firmasının başarısız olup satılması dikkat çekicidir.

Biraz farklı türden ekosistemleri günümüzün yaygın web uygulamalarında da görebiliyoruz. Örneğin Facebook uygulaması yaygınlaşmaya başladıktan kısa bir süre sonra bir uygulama ekosistemine dönüşmüştür. Çekirdek uygulamanın sahibi şirket kullanıcı memnuniyetini arttırmak amacıyla ticari veya gayri ticari, kendisinin veya başka şirket/şahısların geliştirdiği uygulamaların temel sistemden ayrı ama onunla entegre olarak yüklenip çalıştırılmasına kapı açmıştır. Farklı bir bütünleşme/eşleşme tekniği kullanılmasına rağmen bu durum yukarıdaki örneklerdeki gibi çok paydaşlı bir sistemde paydaşların bağımsız olarak geliştirdiği bileşenlerin temel ve sade bir mimari ile birleştirilmesi esasına dayanmaktadır. Netflix, Google maps gibi web uygulamaları da benzer bir tarzdadır.

Yine farklı bir örnek olarak Wikipedia gösterilebilir. Bu örnekte yazılım değil içeriğin dağıtık olarak geliştirilmesi sözkonusudur. Yine de bu örnek yazılım dünyası ile yakınlık gösterir, çünkü Wikipedia'nın esasını oluşturan Wiki kavramı yazılım dökümantasyonlarının gevşek bir işbirliği ile geliştirilmesi gereksiniminden doğmuştur, ve yukarıda bahsettiğimiz Linux, Apache, Eclipse gibi yazılımların hepsinde sistemin kullanılışlılığını arttıran büyük miktarda dökümantasyon merkezi bir planlamayla değil bu Wiki modeliyle üretilmektedir.

3 Açık kaynak yazılım modelinin evrimsel avantajı

İlk kez Linux işletim sisteminin yaygınlaşmasıyla adını duyuran açık kaynak yazılım geliştirme ve lisanslama modeli dağıtık geliştirme ve evrimsel çeşitlenmeye elverişlilik açısından önemli avantajlar sunmaktadır. Son on yılda da firmalar bu modelin avantajlarını kendi kar modelleriyle birleştiren melez modeller arayışında başarılı deneyimler ortaya koymuştur. Kökeni itibarıyle kar amacı gütmeyen, mesleki veya akademik camia içinden çıkan kollektifler tarafından kullanılan açık kaynak lisansları, yazılımı kamusal alana konumlandırır ve dışlayıcı kullanıma veya geliştirmeye izin vermez. Örneğin Linux'un değiştirilerek bir üründe kullanılması mümkün olmakla beraber lisans koşulları bu değiştirilmiş yazılımın kaynak kodlarının kamuya açılmasını zorunlu kıldığından firmalar için bir rekabet sıkıntısı yaratır. Buna rağmen özel sektörün açık kaynak modeline ilgisi melez lisansların ortaya çıkışına yol açmıştır. Bu melez modeller Apache'de olduğu gibi teknolojinin çok paydaşlı işbirliği ile geliştirilmesinin önünü açarken, dışlayıcılığın kapsamını sınırlı tutarak ticari kar modeli oluşturmaya uygun alanlar yaratır.

Benkler'in [8] kaydettiği gibi patent alışverişi, ikili stratejik ortaklıklar türü 'piyasa' usulü işbirliği yöntemleri, yazılım gibi geliştirilmesinde yoğun bilgi alışverişi gerektiren alanlarda ciddi bir sürtünme etkisi yaratır. Bu tür yoğun bir bilgi alışverişinin gerçekleşmesi firmaların içe kapalı bir ArGe tarzı yerine diğer firmalarla aralarındaki sınırların daha geçişken olduğu bir stratejiyi benimsemelerini gerektirir [9]. Çekirdek teknolojinin kamusal alanda konumlanması bir yandan işbirliğini son derece kolaylaştırırken bir yandan -melez lisanslamayla- firmaların çekirdek teknolojiyi kullanarak katma değeri olan ürünler yaratmasına imkan verir, ve dolayısıyla ortak teknolojiye karkı vermeleri için gerekli motivasyonu sağlar [10]. Bu model ayrıca firmaların müşteri taleplerine ve kullanıcı kaynaklı innovasyonun girişine açık olması sonucunu doğurduğu için [11] sadece çeşitlenmeyi arttırmakla kalmaz, çeşitlenmenin kalitesini ve dolayısıyla uyum başarısını da yükseltir.

4 Dinozorların geri dönüşü?

Simon'un [12] tespit ettiği gibi sistemler evrilirken başarılı alt-sistemler ortaya çıkar ve yaygınlaşır. Evrimsel değişimin seçim tarafında ortaya çıkan bu durum yazılım sistemlerinde dikkatli olunması gereken bir riske işaret eder. Eclipse ve Apache örnekleri yakından incelendiğinde görülebileceği gibi, belirli bileşenler o kadar yaygın kullanım alanı bulmuştur ki bunlar artık tercihli bileşenler olmaktan çıkıp temel sistemin bir parçası haline gelmişlerdir. Bu durumda bu bileşenlerdeki tasarım problemlerinin tüm sistemi olumsuz etkileme riski oluşur; bu bileşenlerle diğerleri arasındaki eşleşme artık gevşek değildir. Bahsi geçen her iki örnekte de gelişimin belirli bir noktasında sistem tasarımlarının gözden geçirilmesi ve zahmetli değişiklikler gerekmiştir. Üstüne yük binerek

dinozorlaşan bileşenlerin bu yükü kaldırmak için yeniden inşası ve/veya bölünmesi zorunlu olmuştur.

Yine de bu tecrübeler dağıtık geliştirmenin merkezi bir ilk tasarım süreciyle birleştirilmesinin bu riskleri hafifleteceğine işaret ediyor. Açık kaynak modelinin işleyişinde meritokratik yetki ve karar süreçleri esastır. Ancak özellikle Eclipse ve Apache gibi melez örneklerde meritokratik yapıya bürokratik/hiyerarşik bir yapı eşlik etmektedir. Bu hiyerarşik yapılar piyasa beklentilerin ve şirketlerin iş önceliklerinin yazılım sisteminin tasarımına yansıtılmasına aracı olarak olası sıkışmaları önlemeye imkan verirler. Eclipse projesinde bir vakıf yapısı kurulmuş ve bu teknolojiye ciddi yatırım yapan firmalar belirli yükümlülükler üstlenerek bu yapıya dahil olmuş, buna karşılık teknolojinin yönünün belirlenmesiyle ilgili belirli ölçüde söz sahibi olmuşlardır. Apache, R, GCC gibi başka örneklerde benzer bürokratik/yarı-bürokratik yapılar konsorsiyum veya vakıf formunda ortaya çıkmışlardır. Bu yapılar bir tarafı meslek camiasına ve meritokrasiye dayanan bu projelere piyasadaki şirketlerin de entegre olmasını temin ederler. Linux örneğinde ise varolan meritokratik yapı piyasa talepleriyle farklı yollardan entegre olduğundan (ve paydaş sayısı çok yüksek olduğundan) süreç farklı işlemiş ve böyle bir yapı ortaya çıkmamıştır.

Fitzgerald'ın [13] belirttiği gibi açık kaynak dağıtık geliştirme modeli birçok avantaj sunmasına rağmen stratejik planlama konusunda bir boşluk yaratır. Bu durum bizi merkezde, sınırlı bir katılımla, ve tutarlı bir planlama ile plansız ama dağıtık ve hızlı evrilen sistemler arasında bir tercihe zorlar görünüyor. Buna karşılık kısmi katılımlı bir planlama/ilk-tasarım sürecinin, gittikçe ağırlığı artan bir dağıtık geliştirmeyle devam etmesine dayalı melez süreçler ve modeller, kullanılabilir bir arayol olmuşlardır. Bu türden bir 'hafif' planlama gevşek eşleşmenin sağlıklı işleyişinin yanısıra süreç içinde yeni dinozorların oluşmasına karşı bir gözetim işlevi de yapmaktadır.

5 Sonuç

Tasarım mühendislikten farklı olarak hedefi belirli bir süreç değildir ve bu yüzden öngörülemez. Yaygın ve heterojen bir kullanıcı grubuna hitap eden yazılım sistemlerinde ve taleplerin sabit değil değişken olduğu durumlarda geliştirme hedeflerinin baştan konulması mümkün görünmemekte, bu yüzden yazılım geliştirme pratiği mühendisliğe ilaveten tasarım pratiğiyle de benzerlikler sergilemektedir. Bu durumlarda yazılım geliştirme planlanan bir süreç olmaktan ziyade olmasına izin verilen bir süreçtir. Bu türden bir sürecin başarılı bir şekilde yürütülmesi ise dağıtık olarak ve en düşük düzeyde koordinasyon ile gerçekleşen geliştirmenin gevşek eşleşme esaslı bir mimari ile harmanlanmasını gerektirir. Bu da esasen yazılımın mekanik değil organik bir algılama ile, ve öngörüden ziyade evrilebilirlik odaklı olarak yönetilmesine gerek duyar. Tecrübeler, evrimsel çeşitlenmeye yatkın bu gevşekliğin avantajlarının uzun vadeli olması için gevşek olmayan bir ilk tasarım sürecinin yanısıra meritokratik yapıya eşlik eden, yönetim değil gözetim odaklı bir bürokratik yapının yararlı olduğunu göstermektedir.

Kaynaklar

[1] D. J. Teece, "Inter-organizational requirements of the innovation process," *Managerial and Decision Economics*, vol. Special issue, pp. 35–42, 1989.

[2] G. Arango, E. Schoen, and R. Pettengill, "Design as evolution and reuse," pp. 9 –18, mar. 1993.

[3] F. P. Brooks, *The mythical man-month*. Addison Wesley Longman, 1995.

[4] J. D. Orton and K. E. Weick, "Loosely coupled systems: A reconceptualization," *The Academy of Management Review*, vol. 15, no. 2, pp. 203–223, 1990.

[5] R. Sanchez and J. T. Mahoney, "Modularity, flexibility, and knowledge management in product and organization design," *Strategic Management Journal*, vol. 17, pp. 63–76, 1996.

[6] P. G. Capek, S. P. Frank, S. Gerdt, and D. Shields, "A history of ibm's open-source involvement and strategy," *IBM Systems Journal*, vol. 44, no. 2, pp. 249–257, 2005.

[7] E. Penrose, "Strategy/organization and the metamorphosis of the large firm," *Organization Studies*, vol. 29, no. 8, pp. 1117–1124, 2008.

[8] Y. Benkler, "Coase's penguin, or, linux and the nature of the firm," in *CODE: Collaborative Ownership and the Digital Economy* (R. A. Ghosh, ed.), pp. 169–206, Cambridge: The MIT Press, May 2005.

[9] H. Chesbrough, *Open Innovation: Reaching a new paradigm*, ch. New puzzles and new findings. Oxford University Press, 2006.

[10] J. West, "How open is open enough?melding proprietary and open source platform strategies," *Research Policy*, vol. 32, pp. 1259–1285, 2003.

[11] E. von Hippel and G. von Krogh, "Open source software and the "private-collective" innovation model: Issues for organization science," *Organization Science*, vol. 14, pp. 209–223, March 2003.

[12] H. A. Simon, "The architecture of complexity," *Proceedings of the American Philosophical*

Society, vol. 106, no. 6, pp. 467–482, 1962.
[13] B. Fitzgerald, “The transformation of open source software.,” *MIS Quarterly*, vol. 30, no. 3, pp. p587 – 598, 2006.