Katedral ve Pazar

TMMOB
ELEKTRİK MÜHENDİSLERİ ODASI
1954

KATEDRAL VE PAZAR

"The Cathedral and the Bazaar"

Eric Steven Raymond

Çevirenler: Cihan Gerçek; Aydın Bodur

EMO Yayın No: GY/2008/6

1.Basım, Ankara, Ağustos 2008

ISBN: 978 9944 89 598 9

Elektrik Mühendisleri Odası

Ihlamur Sokak No:10 06420 Kızılay/Ankara

Tel: (312) 4253272 **Faks:** (312) 4173818

e-posta: emo@emo.org.tr **web:** http://www.emo.org.tr

001.5 Ray
Ray, Eric Steven
Katedral ve Pazar = *The Cathedral and the Bazaar* : Çeviren Cihan Gerçek, Aydın Bodur-- 1.bs. -- Ankara : Elektrik Mühendisleri Odası Yayınları, 2008.
50 s. ; 24 cm
(Emo Yayınları ; - GY/2008/6) 978-9944-89-598-9

Program geliştirme

EMO adına Yayına Hazırlayan: Aydın Bodur
Kapak Tasarım: Nahide Akkuş

Baskı
Başak Matbaacılık ve Tanıtım Hiz. Ltd. Şti.
Tel. 0312 384 27 61 Ankara

BİLEŞİM YAYINEVİ

Katedral ve Pazar

Eric Steven Raymond
Thyrsus Enterprises [http://www.tuxedo.org/~esr/] *<esr@thyrsus.com>*
Version 3.0

Türkçe'ye Çevirenler: Cihan Gerçek, Aydın Bodur –Temmuz 2008

Orijinal Eser:
The Cathedral and the Bazaar
Eric Steven Raymond
Thyrsus Enterprises [http://www.tuxedo.org/~esr/]
<esr@thyrsus.com>
Version 3.0

Te'lif Hakları

$Tarih: 2002/08/02 09:02:14$

Revizyonlar

Revizyon 1.57 11 Eylül 2000 esr
Yeni Başlık "Karışıklığı kaç gözlem giderir"
Revizyon 1.52 28 August 2000 esr
MATLAB, Emacs'a paralel bir destektir. Corbatoó ve Vyssotsky bunu 1965'te anlamıştı.
Revizyon 1.51 24 Ağustos 2000 esr
İlk DocBook versiyonu. 2000 sonbaharında, zaman duyarlıklı malzeme üzerinde küçük değişiklikler.
Revizyon 1.49 5 Mayıs 2000 esr
Mühlet ve tarifelere HBS notu eklendi.
Revizyon 1.51 31 Ağustos1999 esr
O'Reilly'nin basıldığı ilk versiyon.
Revizyon 1.45 8 Ağustos 1999 esr
Snafu İlkesi, pazarın esası ve gelişimi ile ilgili (ön)tarihsel örneklere ait dipnotlar eklendi.
Revizyon 1.44 29 Temmuz 1999 esr
Tasarım alanının keşfine yönelik olarak pazarların yararlılığıyla ilgili incelikleri içeren "Yönetim ve Maginot Hattı Hakkında" bölüm eklendi; Epilog büyük ölçüde genişletildi.
Revizyon 1.40 20 Kasım 1998 esr
Halloween Belgeleri temelinde Brooks'la ilgili düzeltme eklendi.

Revizyon 1.39 28 Temmuz 1998 esr
RMS'in inandırıcı argümanları sonucunda, Paul Eggert'in GPL-pazar grafiğini kaldırdım.
Revizyon 1.31 10 Şubat 1998 esr
"Epilog: Netscape Pazar'ı Kucaklıyor" eklendi
Revizyon 1.29 9 Şubat 1998 esr
"Serbest yazılım" değiştirilerek "açık kaynak" yapıldı.
Revizyon 1.27 18 Kasım 1997 esr
Perl Konferansı anekdotu eklendi.
Revizyon 1.20 7 Temmuz 1997 esr
Bibliyografya eklendi.
Revizyon 1.16 21 Mayıs 1997 esr
Linux Konferansında ilk resmi sunum.

Burada, başarılı bir açık_kaynak projesi olan fetchmail, yani Linux'un tarihinde ortaya konulan yazılım mühendisliğiyle ilgili, şaşırtıcı kuramların sınanması denilebilecek bir çalışma masaya yatırılıyor. Söz konusu kuramlar, iki gelişme tarzı temelinde ele alınıyor. Biri, ticari alemin büyük kısmının modeli olan "katedral"; diğeri de onun karşısında yer alan ve Linux dünyasının modeli olan "pazar". Önce, bu modellerin yazılıma özgü hata ayıklama (arıza giderme, *[:debuging]*) süreciyle ilgili karşıt varsayımlardan kaynaklandığı gösteriliyor. Ardından, Linux deneyiminden gelen ve bireycil unsurların kendini düzelten diğer sistemleriyle üretken benzerlikler gösteren "*yeterince gözlem yapıldığında, bütün hatalar yüzeyseldir*" şeklindeki tez ileri sürülerek; yazılımın geleceğiyle ilgili muhtemel gelişmelere yönelik sonuçlara varılıyor.

EMO'dan Türkçe Çevirisi için

Sunuş

"Richard Stallman, 1971 yılında MIT'nin "Artifical Intelligent" labarotuvarında çalışmaya başladığında, oradaki durumu şöyle anlatıyor"...Çalışmaya başladığımda hemen yazılım paylaşım grubunun bir parçası oldum ve uzun bir zaman birlikte çalıştık. Labarotuvarın şefi olan HACKER yapının assembler'da tasarım ve yazılım işini yapmaktaydı....." diye anlatır bu süreci.

Burada karşımıza iki kavram çıkmaktadır. Birincisi ***yazılım paylaşımı****, diğeri ise* ***HACKER****'dır. Bu kavramları inceleyerek bu günü irdelemek sanırım daha sağlıklı olacaktır.*

Yazılım paylaşımı ya da bugünkü ifadeyle ***özgür yazılım*** *[:Freesoftware] yazılım dünyasında devrim yaratan bir sürecin felsefesini içeren bir kavramdır. Bu olgunun kökeni yukarıda belirtildiği gibi 70'li yıllar hatta 40'lı yılların sonuna kadar gitmektedir. 40'lı yılların sonunda ağırlığını fizikçilerin oluşturduğu yazılım grupları İngiltere ve ABD'de çalışmalarını yürütürken kendilerini REAL PROGRAMMER olarak nitelemekteydiler ve en önemli özellikleri tüm bilgi birikimlerini sınırsız bir şekilde paylaşmalarıydı. Bu anlayış ve çalışma tarzı 80'li yılların ilk yarısında kendini freesoftware akımında buldu. Bu dönemde HACKER kavramı her yerde ve her düzeyde duyulmaya başlandı. Saygı duyulan ve yazılımcıların kendilerine zevkle yakıştırdıkları bu kavram CIA ve FBI'ın çarpıtmaları sonucu CRACKER'lar ile aynı kurguda anılır oldu.*

Neydi özgür yazılımı ve hacker kavramını geleneksel olandan ayıran? Hacker kavramı ve hacker etiği temelde kapitalizmin protestan etiğine kökten bir başkaldırıdır. Bilgi çağında yeni bilgi en etkili olarak, esprili ve insanın kendi bireysel/özgür ritminde çalışması ile üretilebiliyor. Bu nedenle açık model sadece etik anlamda doğru olmakla kalmıyor, uygulamada da çok güçlü ve üretken olarak kendini gösteriyor. Network ve İnterneti birleştirerek NET diye tanımlarsak, Net'i ve Linux'u olanaklı kılan işte bu özgür yazılım kavramıdır ve onun etik değerleridir.

Torvalds, Linux üzerinde çalışmaya 1991'de, Helsinki üniversitesinde bir öğrenci iken başladı. Önce evindeki bilgisayara Andrew Tanenbaum'un minix'ini kurdu. Onun üzerinde çalışarak geliştirilebilir bir iskelet ortaya çıkardı. Bu

çalışmanın en önemli özelliği, başından itibaren projeye başkalarını da dahil etmesiydi. 25 Ağustos 1991'de, Net'te konu başlığı "Minix'te en çok neyi görmek istersiniz?" olan bir mesaj gönderdi ve bu mesajda "özgür bir işletim sistemi" yaptığını ilan etti. Cevap olarak bir sürü düşünce ve hatta programı test etmede yardım etme sözleri geldi. İşletim sisteminin ilk versiyonu, Eylül 1991'de Net'te kaynak kodu herkese açık olarak çıktı.

Ortak çalışma ile yeni sürümü Ekim başında hazırdı. Torvalds bunun ardından tüm yazılım dünyasına, yeni sistemi geliştirmede ona katılmaları için, daha doğrudan bir davette bulundu. Bir ay içinde diğer programcılar işe karışmışlardı. O günden bu yana, Linux network'ü şaşırtıcı ve yaratıcı bir süratle büyüdü.

Eric Raymond, ilk Net'te yayınlanmış olan ünlü denemesi "Katedral ve Pazar Yeri"nde Linux'un açık modeli ile çoğu şirket tarafından tercih edilen kapalı model arasındaki farkı, modelleri pazara ve katedrale benzeterek açıklıyor. Bir teknoloji uzmanı olmasına rağmen Raymond, Linux'un gerçek yeniliğinin teknik ile birlikte ve daha da önemli olarak sosyal yanı olduğunu vurgular. Yeni, tamamen açık ve sosyal bir şekilde geliştirilmiş olması önemli bir yenilikti. Onun kelimeleri ile katedralden pazara geçişti.

Raymond katedrali, içinde bir kişinin veya çok küçük bir grup insanın, her şeyi önceden planladığı ve sonra kendi gücüyle planı gerçekleştirdiği bir model olarak tanımlıyor. Gelişim, kapalı kapılar ardında oluşur ve diğer herkes, sadece "bitmiş" sonuçları görür. Pazar modelinde ise, düşünce üretme süreci herkese açık ve düşünce başından itibaren, test edilmek üzere diğerlerine dağıtılıyor.

Görüşlerin çeşitliliği en önemli şeydir ve düşünceler erken bir aşamada geniş çapta yayıldığından, dışarıdan yapılacak ekleme ve eleştirilere de açık bir süreç oluşur. Katedral bitmiş halde sunulduğunda ise, temelleri artık değişmez. Pazarda ise, insanlar değişik yaklaşımlar bulmaya çalışırlar ve birsinin harika bir düşüncesi olduğunda, diğerleri onu alıp üzerine inşa ederler.

Bu özgür ve açık-kaynaklı model, genel anlamda şöyle anlatılabilir: Her şey, bir sorunun ortaya çıkması veya birinin önemli bulduğu bir amaçla başlar. O sorunun çözümü veya kişinin amacını gerçekleştirmesi sadece bu düzeyde kalmaz aynı zamanda 0.1.1 sürümünü de oluşturur. Açık modelde alıcı, bu çözümü özgürce kullanma, test etme ve geliştirme hakkına sahiptir. Bu kurgu ancak ve ancak çözüme giden bilgi de (kaynak) beraberinde var ise olanaklıdır.

Özgür ve açık-kaynaklı modelde, bu hakların verilmesi iki yükümlülüğü de beraberinde getirir: aynı haklar orjinal çözüm veya iyileştirilmiş sürüm (0.1.2) paylaşıldığında devredilmelidir ve katkıda bulunanlar, herhangi bir sürümü

paylaşıldığında belirtilmelidir. Açık kaynaklı model bu haliyle Platon'un Akademia'sının bir devamıdır.

Bilimsel etik, teorilerin ortaklaşa hazırlandığı ve hataların görülüp, tüm bilim camiasının eleştirileri yoluyla aşama aşama düzeltildiği bir model gerektirir. Elbette ki bilim adamları bu modeli, salt etik nedenlerden dolayı değil, aynı zamanda bilimsel bilgiye ulaşmanın en başarılı yolu olduğu için seçtiler. Doğa hakkında bildiklerimizin tümü, bu akademik veya bilimsel modele dayanır. İlk Hacker'ların açık kaynaklı modelinin bu kadar etkili biçimde çalışmasının nedeni tutkularını gerçekleştiriyor olmalarına ve diğer yazılımcılar tarafından motive edilmelerine ek olarak büyük ölçüde, bilgi üretimi için ideal açık akademik modele uymasıdır.

Manastır modelinde ise sadece bilgi erişimini kısıtlamakla kalmayıp, otorite yanlısı olan bir modeldir. Bu modele göre yapılandırılmış bir iş girişiminde otorite, amacı belirler ve gerçekleştirilmesi için küçük kapalı bir grup insan seçer. Grubun kendisi testini tamamladıktan sonra, diğerleri sonucu olduğu gibi kabul etmek zorundadır. Sonucu başka şekilde kullanmak, "yetki dışı kullanım"a girer. Kapalı model, bir faaliyeti daha yaratıcı ve kendi hatalarını düzeltici olmasına olanak verecek şekilde inisiyatif kullanımına veya eleştiriye izin vermez.

*Hacker öğrenim modelinin asıl gücü, hacker'ın öğrenirken aslında diğerlerine de öğretmesidir. Bir hacker, bir programın kaynak kodu üzerinde çalıştığında, genellikle onu geliştirir ve başkaları da onun çalışmasından faydalanır. Kaynaklarını kontrol ettiğinde, genellikle kendi tecrübesinden edindiği yardımcı bilgileri ekler. Çeşitli sorunlar etrafında devam eden, eleştirel ve geliştirici bir tartışma oluşur. Bu tartışmaya katılmak ve bilgiyi eklemenin ödülü ise diğer katılımcılar tarafından kabul görmektir." **

Elektrik Mühendisleri Odası başından beri destek olmaya çalıştığı açık kaynak kodlu yazılım alanında neredeyse bir klasik olan elinizdeki kitabı kamuoyu ile paylaşmaktan onur duymaktadır.

Amacına hizmet etmesi dileğiyle

Saygılarımızla
Ağustos 2008

Musa ÇEÇEN
TMMOB Elektrik Mühendisleri Odası
41.Dönem Yönetim Kurulu Başkanı

**Kaynaklar : http://www.ozguryazilim.net vd...

İçindekiler

Katedral ve Pazar

Linux yıkıcıdır. Kim derdi ki gezegenin orasına burasına dağılmış birkaç bin programcının birbirine Internet'in ince telleriyle bağlı bölük pörçük çalışmaları, büyü yapılmış gibi birleşsin; dünya çapında bir işletim sistemi olup çıksın!

Ben kesinlikle diyemezdim! Linux'un izinin radarıma yakalandığı 1993'ün başlarında, ben on yıldır Unix'e ve açık_kaynak geliştirmeye sardırmış durumdaydım. 1980'lerin ortalarında, GNU'da katkısı olan ilk kişilerdendim. Açık_kaynak yazılımı konusunda, nete (ağ ortamına) hatırı sayılır miktarda malzeme sunmuştum. Bunlar, tek başıma yaptığım veya ortaklaşa çalışmalar (nethack, Emacs's VC ve GUD modları, xlife ve diğerleri) şeklinde, bugün de yaygın kullanılan çeşitli programlardır. Neyin nasıl olduğunu bildiğimi sanıyormuşum!

Linux, bildiğimi sandığım çok şeyi altüst etti. Ben yıllarca Unix'in küçük gereçler, hızlı prototipleme ve evrimci programlama ile ilgili düsturunu vaz etmiştim. Ama öte yandan da, bunları aşan, nazik bir durum arz eden bir karmaşa yaşandığını, daha merkezi ve a priori (önceliklikli) bir yaklaşım gerektiğini düşünüyordum. Önemli yazılımların (Emacs programlama editörleri gibi gerçekten iri programların ve işletim sistemlerinin) izole bir ortamda çalışan ustalar yahut küçük bir üstatlar grubu tarafından özenle işlenmiş katedraller gibi inşa edilmesi ve bir beta sürümünün vaktinden önce yapılmaması gerektiği kanaatinde idim.

Linus Torvald'ın tarzı -yani, erken ve sık aralıklarla sürüm yapmak; mümkün olduğu ölçüde çok şeyi vermek; karışıklıklara açık olmak- epey şaşırtıcı olmuştu. Linux çevresinde, katedral tarzının ihtişamından ziyade, tutarlı ve kararlı bir sistemin doğmasının birtakım mucizelere bağlı olduğu imajını veren, çeşitli gündemlerin ve yaklaşımların konu edildiği gürültülü bir pazar yeri havası vardı (bu en iyi, Linux'un, *alelade, herhangi birinin* fikirlerini sunduğu arşiv siteleriyle sınırlanır).

Bu pazar tarzının işleyecek gibi durması ve iyi işlemesi de bir sarsıntı yarattı. Aslında, sırf bireysel projeler için değil; ayrıca Linux dünyasının katedral kurarken hayal bile edilmeyecek hızlarda karmaşaya düşmemesinin; hatta tersine gücüne güç katmasının sebebini anlamak için de çok kafa patlattım.

1996 ortasında, durumu anlamaya başladığımı düşünüyordum. Şans eseri açık_kaynak projesi formunda karşıma çıkan harika bir fırsat,

pazar tarzında bilinçli bir çabaya girmemi sağladı. Projeyi yaptım -sonuç, anlamlı bir başarıydı.

O projenin öyküsü olan bu yazıda, açık_kaynak temelinde proje geliştirme konusunda hisseler çıkartacağım. Linux dünyasında öğrendiklerim bu ilk bilgilerden ibaret değildir. Ayrıca, Linux dünyasının bu şeylere neler kattığı da görülecektir. Bunların, Linux çevresini bir çeşit yazılım hayratı haline getirenin ne olduğunu kavramanıza yardım edeceklerini sanıyorum - hatta kim bilir, daha verimli olmayı bile sağlayabilirler.

Mektup Adresine Varmalı

1993'ten itibaren, Pennsylvania'daki Batı Chester'da serbest erişimli bir Internet Servis Sağlayıcı olan CCIL'in (Chester County InterLink) teknik tarafını yürütüyordum. Kurucu ortağıydım ve çok kullanıcılı bülten paneli yazılımı da benim üstümdeydi - *telnet://locke.ccil.org* adresi üzerinden bakabilirsiniz. Bugün otuz hat üzerinde üç bin kullanıcıya hizmet veriyor. Ben bu sayede CCIL'in 56K hattı üzerinden günde 24 saat nete erişim sağlıyordum - aslında bu, bir yerde işin gereğiydi.

Internet e-postayı sıkça kullanmam gerekiyordu. *locke* sitesinde postama bakmak için telnet'e girme mecburiyeti, illet bir şeydi. Benim istediğim, gelen bir mektuptan haberdar olmak ve kendi yerel gereçlerimle ilgilenmek üzere mektubun (kendi ev sistemime) teslimi idi.

Internet'in malum posta transfer protokolü, yani SMTP (Basit Posta Transfer Protokolü, *[:Simple Mail Transfer Protocol]*) uygun değildi. Çünkü, makinelerin gereğince işlemesi, tam zamanlı bağlantıda olmalarına bağlıdır. Oysa benim cihazım, sürekli Internet'te değildi ve statik IP adresi yoktu. İhtiyacım olan şey, çevirmeli bağlantı üzerinden, postamın yerel olarak teslimini sağlayacak bir programdı. Bu türden bir şeylerin olduğunu ve pek çoğunun POP (Postahane Protokolu, *[:Post Office Protocol]*) denilen basit bir uygulama protokolüne dayandığını biliyordum. POP, bugün pek çok posta istemcisi [*:client*] tarafından tanınmasına rağmen, o zamanlar benim kullandığım okuyucuda kurulu değildi.

Bana bir POP3 istemcisi gerekiyordu. Internet'te aradım ve buldum. Aslında üç-dört tane bulmuştum. İçlerinden birini bir süre kullandıysam da; olması gereken özelliklerinden biri, yani, yanıtlamanın çalışabilmesi için çekilip alınmış [*:fetched*] mektuptaki adreslerle başa çıkma [*:hack*] yeteneği yoktu.

Problem şuydu: Diyelim 'joe' diye biri *locke*'da bana mektup göndermiş olsun. Eğer mektubu yerel sistemime alır da yanıtlamaya kalkarsam; postacım [:*mailer*], tüm iyi niyetiyle, bu cevabı sistemimde mevcut olmayan bu 'joe' adlı kişiye göndermeye kalkışır. Ayrıca <@ccil.org> üzerinde dönüş adreslerini elle düzenlemek de ciddi sıkıntı demektir.

Oysa, bilgisayarın yapması gereken şeylerden biri olan bu iş konusunda, mevcut POP istemcilerinden hiçbiri bir şey bilmiyordu. Böylece ilk dersimize gelmiş oluyoruz:

> 1. Her iyi yazılım, programcısının kendi yarasını kaşımasıyla başlar.

Aslında bu herkesin malumu olmakla birlikte ("İcatların anası zaruretlerdir" ünlü bir deyiştir), yazılımcılar günlerini genelde ihtiyaç duymadıkları ve sevmedikleri programlarla uğraşarak geçirirler.

Ama Linux dünyasında öyle değildir - nitekim Linux çevresinde üretilen yazılımların ortalama kalitesinin yüksek olması da muhtemelen bu nedenledir.

Bu durumda derhal mevcutlarla yarışacak yepyeni bir POP3 sunumcu kodu yazmaya mı giriştim? Olacak iş değil! Ama, elimin altındaki POP uygulamalarını inceledim ve "Bunlardan hangisi benim istediğime yakın?" diye sordum. Çünkü:

> 2. İyi programcı, ne yazacağını bilen programcıdır. Ama, neyi yeniden yazacağını (ve kullanacağını) bilen programcı, büyük programcıdır.

Büyük programcı olduğumu söylemiyorum fakat öyle olmaya özeniyorum. Büyük olmanın ayırt edici karakteristiği, yapıcı tembelliktir. En yüksek not, sonuca değil çabanıza verilir. Ayrıca, eldekiyle başlamak ta, daima sıfırdan başlamaktan iyidir.

Örneğin Linus Torvalds da [http://www.tuxedo.org/~esr/faqs/linus] yazmaya sıfırdan başlamaya girişmemiş; tersine elinin altındaki bir programı, PC klonlarına yönelik Unix benzeri ufak bir işletim sistemi olan Minix kodunu kullanmıştı. Yani bu kod, büyüyüp Linux olacak bebek için, bir çıkış noktası olmuştu. Benim hazır bir POP işlevini kalkış noktası olarak kullanmam da, aynı hikayedir.

Unix dünyasının kod paylaşma geleneği, kodun yeniden kullanımı konusunda dostane bir tutum sergiler (nitekim bu, GNU projesinde işletim sistemi olarak, çeşitli çekincelerine rağmen, neden Unix'in seçildiğini de açıklar). Linux dünyası da bu geleneği teknolojik sınırların

sonuna kadar kullanmıştır. Çoğu durumda terabaytlar ölçüsünde kaynak, kullanıma açık durumdadır. Yani, birinin Linux dünyasındaki eli yüzü düzgün denebilecek çalışmasına ulaşmak için sarf edilen çaba, başka yerlere göre çok daha iyi sonuç verir.

Benim için de tam olarak böyle oldu. İşin başında bulduklarım sayesinde, ikinci adımdaki araştırmalarım sonunda dokuz aday tespit ettim: fetchpop, PopTart, get-mail, gwpop, pimp, pop-perl, popc, popmail ve upop. İlki, yani fetchpop, Seung-Hong Oh'nun geliştirdiği bir işti. İçine, mevcut başlık-yeniden yazımı [*:header-rewrite*] özelliğinden başka eklemeler de yaptım. Zaten o da bunların 1.9 sürümünde yer almasını uygun gördü.

Ancak birkaç hafta sonra Carl Harris'in popclient/popistemci (posta istemcisi, postahane protokolü istemcisi, protokol istemcisi [*:popclient*]) koduyla tanıştığımda, bir sorunum olduğunu fark ettim. Fetchpop (örneğin arkaplan-daemon modu gibi), iyi sayılabilecek özgün fikirler içermekle birlikte, sadece POP3'le baş edebiliyordu ve oldukça amatörce yazılmıştı (o sıralar, Seung-Hong parlak fakat deneyimsiz bir programcıydı ve bu iki nitelik hemen göze çarpıyordu). Carl'ın yazdıkları daha iyi, profesyonel ve derlitoplu idi. Ama uygulamaya özgü hileler de denilebilecek bazı önemli fetchpop özelliklerine ihtiyacı vardı (ki onları da bizzat ben yazdım).

Gitmek mi zor, kalmak mı? Gitsem, yani diğerine geçsem; hazırladığım kod, daha iyi bir geliştirme zemini uğruna heba olacaktı. Fakat öte yandan da gitme yönünde, çoklu-protokol desteğinin varlığından doğan güçlü bir dürtü vardı. Postahane sunucu protokollerinin en yaygın olanı, tek olmamakla birlikte, POP3'tür. Fetchpop ve diğer rakipleri POP2, RPOP yahut APOP işini görmezler. Ayrıca ben de, sırf eğlence olsun diye, IMAP [http://www.imap.org] (Internet Mesajı Erişim Protokolü, *[:Internet Message Access Protocol]*, yani en son ve güçlü postahane protokolü) eklemek konusunda kararsızlık içindeydim. Yine de gitmenin iyi bir fikir olacağı konusunda teorik sayılabilecek bir gerekçem vardı ki daha Linux'a gelmeden epey önce öğrendiğim bir şeydi:

> 3. "Heba etmek hesap dahilindeyse, bir şekilde heba edersin." (Fred Brooks, *The Mythical Man-Month, Chapter 11*)

Bunu şöyle de formüle edebiliriz: Bir çözüm bulana kadar, sorunun ne olduğu genelde anlaşılmaz. İşi hakkıyla yapmayı öğrenmek, belki ikinci

denemede mümkün olacaktır. Bir işi hakkıyla yapma arzusundaysanız, bir kere daha başlamaya hazır olacaksınız [JB].

Neyse, fetchpop'a geçişlerim ilk denememde oldu (diyordum kendime). Böylece diğerine geçtim.

İlk grup popclient yamalarını [*:patch*] 25 Haziran 1996'da Carl Harris'e yolladığımda, kendisinin popclient konusundaki ilgisini epeyce yitirmiş olduğunu keşfettim. Kod, bir parça tozlanmıştı ve de ufak tefek arızalar (hatalar) içeriyordu. Değiştirilecek çok şey vardı. Ayrıca, yapacağım en mantıklı şeyin, programı bitirmeyi üstlenmek olduğunda derhal anlaşmıştık!

Ben henüz farkına varmadan proje ısınmıştı. İş, var olan bir POP istemcisine ufak tefek yamalar tasarlamaktan çıkmıştı; projenin arkasını getirmekle uğraşıyordum ve zaten önemli değişiklikler yapmam icap edeceğine dair bir takım kuşku bulutları da şekillenmeye başlamıştı. Kod paylaşımını öne çıkaran bir yazılım kültüründe evrim, projenin doğal sürecidir. Şu ilkeye göre hareket ediyordum:

> 4. Doğru yoldaysan, birbirinden ilginç problemlerle karşılaşacaksın.

Ama Carl Harris'in durumu daha önemliydi. Çünkü o, şunun farkındaydı:

> 5. Bir programa olan ilgini yitirdiysen; geliştirmek için rakip programı seçmek, en son yapacağın şey olmalı.

Değerlendirmesine bile girmemiş olmakla birlikte, Carl da ben de biliyorduk ki; en iyi çözüme varmak ikimizin de ortak hedefiydi. Yegâne sorun, benim sağlam pabuç olarak iş görüp göremeyeceğimdi. Ama (Carl), benim işi kotardığımı gördükten sonra, gayet zarif ve yardımcı bir tavır sergiledi. Umarım sıram gelince ben de onun gibi olurum.

Kullanıcı Sahibi Olmanın Önemi

Böylece popclient, bana miras kaldı. Ayrıca, kullanıcı havuzu da miras kalmıştı. Kullanıcının olması, harika bir şeydir. Bu, sadece bir ihtiyacı gördüğünü değil; işi doğru yaptığını da gösterdiği için böyledir. Gereğince donatılırlarsa, programın oluşumuna katkıları olabilir.

Unix geleneğinin bir başka kuvveti, -ki Linux çok hoş bir uç noktaya vardırmıştır, kullanıcıların çoğunun programcılar olmasıdır. Kaynak

kodu el altında olduğu için *etkili* programcılar olmaları mümkündür. Arıza (hata) giderme süresini kısaltmakta müthiş yararlı oabilirler. Kullanıcılar biraz teşvikle, sorunları teşhis eder; onarım için öneriler getirir ve kodu tek başınıza yapmaya kıyasla çok daha hızlı geliştirmenize yardımcı olurlar.

> 6. Programın oluşumunda kulanıcıların katkısına başvurmak, kod geliştirmede ve arıza (hata) gidermede en etkili ve en rahat yoldur.

Bu etkiyi göz ardı etmek gayet kolaydır. Aslında bizler, -yani açık_kaynak dünyasında yer alanların hatırı sayılır kısmı, Linus Torvald tersini gösterene kadar, bu etkinin, sistemin karmaşıklığıyla ve kullanıcı sayısıyla artacağını ciddi ölçüde göz ardı ettik.

Nitekim, bence Linus'un en zekice ve mühim eseri, inşa ettiği Linux çekirdeği değil; fakat icat ettiği Linux geliştirme modelidir. Bu fikrimi onun da bulunduğu bir sırada açıkladığımda, gülümsemiş ve sıkça söylediği şu noktayı yinelemişti: "Ben aslında, başkalarının yaptıklarını kendime yontmaktan hoşlanan aşırı tembel biriyim." *Tilki* gibi tembel. Yahut, Robert Heinlein'in, karakterlerinden biri için dediği gibi, başarısız olamayacak kadar tembel.

Geçmişe bakılarak, GNU Emacs Lisp kitaplığının ve Lisp kod arşivlerinin geliştirilmesi, Linux'un başarısına ve yöntemlerine örnek gösterilebilir. Emacs C kalbinin ve diğer bütün GNU gereçlerinin oluşumundaki katedral tarzının tersine, LISP kod havuzunun evrimi akışkandı ve yönlendireni de kullanıcıydı. Fikirler ve prototipler, kararlı bir duruma bürünmeden önce üç –belki de beş kez, yeniden yazılmışlardı. Genelde, Internet üzerinden *a la* Linux, gevşek bağlara sahip bir işbirliği geçerliydi.

Esasen, fetchmail öncesinde yaptığım bana ait en başarılı tekil çalışma, muhtemelen Emacs VC (sürüm kontrolü, [*version control*]) moduydu ki üç kişiyle e-posta üzerinden yürüttüğüm Linux-vari bir işbirliği idi. Bugüne kadar da içlerinden sadece biriyle, Richard Stallman (Emacs'ın yazarı ve Free Software Foundation'ın [http://www.fsf.org] kurucusu) ile tanıştım. Çalışma, Emacs içinden SCCS, RCS ve sonradan da CVS için "tek-tuş" kontrol operasyonları sağlayan bir ön-uç [:*front-end*] idi.

Bu çalışma, başka birinin yazdığı ufak, hamı sccs.el modundan türemişti. VC'nin geliştirilmesi başarı getirdi. Çünkü, Emacs Lisp kodu, bizzat

8. Geliştirme ortağı havuzu ve beta sınayıcı yeterince büyükse, sorunlar çabucak belirlenir ve halledilir.

Daha içten ifade edersek, "Ne kadar gözden geçirirsen; arızalar, o kadar azalır." Ben buna mim koyuyorum: "Linus Kanunu." Benim formülasyonumla bunun karşılığı, "her sorun muhakkak biri tarafından keşfedilir." Linus, bir sorunu anlayıp gideren kişinin ille de -hatta çoğu kez de, o sorunu ilk fark eden kişi olması gerektiğine muhalefet ediyordu. "Biri sorunu fark eder," diyordu, "bir *başkası* kavrar. Hatta, kayda geçsin diye söylüyorum, fark etmek daha büyük önem taşır." Bu düzeltme önem taşıyor; bunu sonraki başlıkta, arıza giderme sürecini incelerken detaylarıyla göreceğiz. Burada kilit nokta, sürecin iki parçasının da (yani, fark etmenin ve gidermenin) çok hızlı bir şekilde gerçekleşmesidir.

Bence, katedral inşasıyla pazar tarzının temel farkı, Linus Kanunu'nda yatıyor. Katedral tarzı programlama anlayışında, arıza ve geliştirme süreci gizli, hain, derinden işleyen bir nitelik arz eder. Güvenilir, dört başı mamur bir şey çıkartmak, kendini o işe adamış birkaç kişinin aylar süren kılı kırk yaran çalışmalarıyla çıkar. Bu da sürümler arası zamanın uzamasına ve onca zaman beklenen sürümün kusursuz olmaması halinde hayal kırıklıklarına yol açar.

Öte yandan, pazar yaklaşımında, arızaların önemsiz olgular olduğunu, yahut en azından, yeni her sürümün, çok sayıda hevesli geliştirme ortağının gözaltında kalacağı düşünüldüğünde, önemini yitireceğini kabul etmek mümkündür. Dolayısıyla sürüm sıklığını artırmak, düzeltmelerden daha fazla yararlanmayı sağlayacak ve istenmeyen bir durum ortaya çıktığında, daha az kayba yol açmak şeklinde bir yan etki yaratacaktır.

Tamam artık. Bu kadarı yeterli. Linus Kanunu yanlış ise, Linux çekirdeği gibi çok çeşitli ellerden geçmiş karmaşık herhangi bir sistem, belli bir noktada, öngörülmeyen kötü etkileşimlerin ve bilinmeyen "derin" arızaların yükünü çekemeyip dağılacaktır. Yok eğer doğru ise, bu kez de Linux'un görece hatasızlığının ve aylarca -hatta yıllarca- süren yükselişini gösterecektir.

Aslında bunun pek öyle şaşırtıcı olmaması gerekir. Yıllar önce sosyologlar, elemanları eşit düzeyde uzman (yahut cahil) olan bir kitlenin ortalama fikrinin, aynı kitleden tesadüfen seçilen tek elemanın fikrinden daha güvenilir bir gösterge olduğunu keşfetmişler ve adını da Delphi *(Delfi)* etkisi koymuşlardı. Netice itibarıyla Linus'un gösterdiği

de, aynı şeyin işletim sisteminin arızalarını gidermeye uygulanabileceği oldu -yani, Delphi etkisi, geliştirmenin karmaşıklığını, bu karmaşıklık işletim siteminin çekirdeği düzeyinde bile olsa, ortadan kaldırabilir. [CV]

Linux'un Delphi etkisi sayesinde öne çıkan başlıca özelliği, bir projeye katkıda bulunanların, o projeye kendi arzularıyla girmiş olmalarıdır. Ayrıca katkının rasgele birinden değil, söz konusu yazılımı kullanmak niyetinde olan, nasıl çalıştığını bilmek isteyen, karşılaştığı sorunların yanıtlarını araştıran ve anlamlı çareler üreten kimselerden geldiği tespit edilmiş bir olgudur. Nitekim, bu süreçlerde yer alan birinin gerçekten katkıda bulunması da kuvvetle muhtemeldir.

Linus Kanunu'nu "Arıza giderme, paralleştirebilir" diye ifade etmek mümkündür. Arıza giderme süreci (hata giderme, [:*debuging*]), arıza gidericilerin bir koodinatörle iletişimini gerektirse de, kendi aralarında koordine olmalarını gerektirmez. Bu nedenle, geliştiricilerin sırtına ilave sorun yükleyen ikinci dereceden bir karmaşıklık ve bir ek külfet (yönetim masrafı) söz konusu olmaz. Pratikte, arıza gidericiler için, Linux dünyasında, işlerinin katlanmasına bağlı teorik verimlilik kaybı diye bir mesele yok gibidir. İşte "erken ve sık sürüm yapma" politikasının bir etkisi de, onarım sürecini beslemek yoluyla bu türden angaryaları minimize etmektir [JH].

Brooks (*The Mythical Man-Month* adlı kitabın yazarı), bu konuyla ilgili şu gözlemi yapar: "Yaygın kullanılan bir programın bakımının toplam maliyeti, geliştirme maliyetinin ortalama yüzde 40'ı civarındadır. Bu maliyet, şaşırtıcı bir şekilde, kullanıcı sayısının güçlü etkisi altındadır. *Daha çok kullanıcı, daha çok arıza bulur*" [vurguyu biz ekledik].

Daha fazla kullanıcı, daha fazla hata bulur. Çünkü farklı yöntemler, programı daha fazla kıstırır. Ayrıca, kullanıcıların aynı zamanda geliştirme ortağı da olması halinde; bu etki daha da güçlenir. Her biri, arızanın belirlenmesi işine farklı bir algılama şekliyle ve çözümleme aracıyla yaklaşır. İşte Delphi etkisi de, bu çeşitlenme nedeniyle, kendini çok açık bir şekilde gösterir. Arıza giderme sürecindeki bu çeşitlenme, gereksiz çabayı azaltan bir eğilim yaratır.

Dolayısıyla, daha fazla beta-sınayıcı eklemekle, *geliştirici* açısından o sıradaki "en ciddi" arızayla ilgili karmaşa ortadan kalkmasa da, uygun araç-gereçe [:*toolkit*] sahip olduğu için arızayı yüzeysel bulacak birine tesadüf etme şansı artacaktır.

Linux çekirdeğinin versiyonu, ciddi arızalar çıkma durumu nedeniyle, muhtemel kullanıcıların "kararlı" son versiyonla çalışmalarını, yahut

yeni özelliklerden yararlanmak adına arıza riskini göze almalarını mümkün kılacak şekilde numaralanır. Bu taktik, Linux programcıları tarafından henüz yeterince uygulanmıyor. Ama uygulansa iyi olur, çünkü iki seçeneğin mevcudiyeti, ikisinin de cazibesini arttırıyor. [HBS]

Karışıklığı Kaç Gözlem Giderir

Pazar tarzının arıza gidermede ve kod geliştirmede hızı arttırması bir şeydir; ama bu durumun mikro düzeyde geliştiricinin ve sınamacının gündelik davranışında tam olarak niçin ve neden böyle olduğunu anlamak başka bir şeydir. Bu bölüm (ki orijinal metinden üç yıl sonra, metni okuyan ve kendi tutumlarını bir daha sınayan geliştiricilerin kavrayışları sayesinde yazıldı) mevcut mekanizmalara etraflı bir bakıştır. Teknik noktalara ilgi duymayan okuyucular doğrudan sonraki bölüme geçebilirler.

Kavramada kilit noktalardan biri, kaynağa-uzak kullanıcıların normalde başvurdukları arıza raporu türünün pek de yararlı olmamasının nedenini fark etmektir. Kaynağa_uzak kullanıcılar, genelde sadece yüzeysel belirtileri bildirmeye eğilim gösterirler. Onlar için bulundukları ortam verilmiş bir durumdur. Bu kullanıcıların (a) kritik geri plan verilerini göz ardı etmelerine ek olarak, (b) arızanın yeniden ortaya çıkmasına yönelik güvenilir bir liste verdikleri de çok nadirdir.

Burada esas mesele, geliştiricinin ve sınayıcının kafalarındaki program modellerinin uyumlu olmamasıdır. Sınayıcı dışarıdan; esas geliştirici içeriden bakar. Kapalı kaynak geliştirmede her ikisi de, bu rollerine gömülürler ve olup bitmiş hakkında konuşup birbirlerinin umudunu kırarlar.

Açık_kaynak geliştirme süreci bu döngüyü kırar. Bu süreç, sınayıcının ve esas (iç) geliştiricinin halihazır kaynak kodu temelinde müşterek bir sunum geliştirecek şekilde verimli bir iletişim kurmalarına olanak verir. Pratikte, arıza (hata) raporunun, dışarıdan görünen semptomları içeren türüyle, -doğrudan doğruya esas geliştiricinin aklındaki- programın kaynak kodu temelli sunumuna yönelik türü arasında, esas geliştirici açısından büyük bir fark vardır.

Arızaların birçoğu, genellikle, kolayca tespit edilir ki; kaynak kodu düzeyindeki hata koşullarının sadece fikir vermesi halinde bile bu böyledir. Beta sınayıcılarınızdan birinin, "nnn satırında sınır ihlali" yahut "X, Y veya Z şartları altında şu değişken yenileniyor" gibi bir noktaya sadece işaret edebilmesi halinde bile, koda üstünkörü bir bakış, asıl sorunu belirlemeye ve çözüm üretmeye yeter.

Yani, her iki tarafın da kaynak kod hakkında fikir sahibi olması, hem iletişimi, hem de beta sınayıcının rapor ettiği ile esas geliştiricinin bildiği arasındaki sinerjiyi iyileştirir. Bunun sonucunda da esas (iç) geliştirici(ler) zamandan tasarruf eder. Açık_kaynak yönteminde esas geliştiriciye zamandan tasarruf ettiren bir diğer unsur da, açık_kaynak projelerinin iletişim yapısıdır. Yukarıda "esas (iç) geliştirici" [*:core developer*] tabirini kullandım. Bu, projenin (bir geliştiriciden oluşan fakat normali bir ila üç arasında olan, dar) merkezi ile projeyi kuşatan (ve sayıları yüzlere varabilen) beta sınayıcıların ve katkıda bulunanların oluşturduğu çevresi (dışı) arasındaki ayrıma işaret eder.

Geleneksel yazılım geliştirmenin işaret ettiği temel problem, Brooks Kanunu'dur: "Gecikmiş bir projeye yeni programcı eklemek gecikmeyi arttırır." Genellersek, Brooks Kanunu, yapılan işin doğrusal, fakat projenin karmaşıklığının ve iletişim masraflarının geliştirici sayısının karesiyle arttığını söyler. Bu kanun, arızaların, farklı ellerin ürettiği kodlar arasındaki arayüzlerde öbeklenme eğiliminde olduğu ve bir projenin iletişim/eşgüdüm yükünün de kişiler arasındaki arayüzlerin sayısıyla birlikte çoğaldığı olgularına dayanır. Dolayısıyla, sorunların ölçeği, geliştiriciler arasındaki iletişim kanallarının sayısıdır ve geliştirici sayısının karesiyle ölçülür (daha açık olarak, N*(N - 1)/2 formülüne göre belirlenir ki: burada N geliştirici sayısını gösterir).

Brooks Kanunun analizi (ve bunun sonucunda ortay çıkan geliştirme gruplarındaki çokça kimsenin korkusu), şu zımni kabule dayanır: Projenin iletişim yapısı ister istemez eksiksiz bir grafiktir; herkes, herkesle konuşur. Ne var ki açık_kaynak projelerinde, dış geliştiriciler, sonuçta birbirinden ayrılabilen alt görevleri yürütürler ve birbirleriyle ilişkileri çok azdır; koddaki değişiklikler ve arıza raporları esas (iç) gruba gelir ve Brooks tarzı sadece bu küçük grup *içinde* geçerlidir. [SU]

Kaynak kodu düzeyinde arıza bildiriminin verimliliğine dair daha pek çok neden vardır. Bunların merkezindeki olay, belli bir hatanın (erör, [*:error*]) kullanıcının kendi çalışma örüntüsünün ve ortamının ayrıntıları temelinde farklılaşan çeşitli semptomlarının olmasıdır. Böyle hatalar, tam anlamıyla karmaşık ve incelikli (dinamik bellek yönetimi hataları yahut deterministik olmayan kesme_penceresi türünden) arızalar olmaya eğilimlidirler. Bunların isteyerek yeniden oluşturulmaları veya statik analizle tespit edilmeleri zordur. Ayrıca yazılımda uzun vadeli sorunlar çıkarmakta en etkili olanlar da bunlardır. Böyle, (örneğin "1250nci satırın bitişiğindeki sinyal rutininde bir pencere var sanki." yahut "Tamponu nerede sıfırlıyorsunuz?" gibi) geçici kaynak kodu düzeyi karakterinde çok semptomlu bir arıza gönderen bir sınayıcının,

geliştiriciye, birbiriyle ilgisiz yarım düzine semptoma ilişkin kritik bir ipucu göndermesi mümkündür.

Bu türden durumlarda, hangi arızanın, dıştan görülebilen hangi hatalı tutuma sebep olduğunu bilmek çok zor, hatta imkânsız olabilir. Ama sürüm sıklığının yüksekliği ölçüsünde bunu bilmek gereksiz olur. Grubun diğer çalışanlarının, arızalarının giderilip giderilmediğini süratla keşfetmeleri muhtemeldir. Pek çok durumda, kaynak düzeyindeki arıza raporları, uygun olmayan tutumların, belli bir onarımla ilişkilendirilmeden terk edilmesine sebep olacaktır.

Çok semptomlu karmaşık hataların [*:errors*], aynı zamanda, yüzeysel semptomlardan fiili arızaya giden çoklu patikaları olması yüksek ihtimaldir. Belli bir geliştirici yahut sınayıcının hangi patikayı izleyeceği, o kişinin ortamının inceliklerine bağlı olabilir ve zaman içinde pek de öyle bariz deterministik bir değişme göstermeyebilir. Neticede, bir semptomun nedenlerini arayan her geliştiricinin veya sınayıcının karşısında, program durum uzayından yarı tesadüfi bir örnek seti vardır. Arıza ne kadar incelikli ve karmaşık ise, becerinin de ilgili örneğin işe yararlılığını garanti etmesi o ölçüde azalacaktır.

Kolayca yeniden üretilebilen ve basit arızalarda, vurguyu "tesadüfi" teriminden "yarı tesadüfi" terimine kaydırmamız gerekir; ayrıca arıza giderme becerisi ve kodla haşır neşirlik ve kodun mimarisi de önemlidir. Fakat, karmaşık arızalarda vurgu "tesadüfi" üzerindedir. Bu şartlar altında iz kovalayan çok sayıda kişi, az sayıda kişiye yeğdir -o az sayıdakilerin ortalama beceri düzeyi kat be kat daha yüksek olsa bile bu böyledir.

Farklı yüzeysel semptomlardan arızaya ulaşan yolları takibin zorluğu, semptomlara bakılarak kestirilemeyecek ölçüde anlamlı değişiklikler gösterirse, yukarıdaki etki ciddi ölçüde büyür. Bu yolları arka arkaya örnekleyen tek geliştirici, ilk denemesinde zor bir yolu kolay bir yol olarak seçmiş gibi olacaktır. Şimdi bir de sık ve hızlı sürüm sayesinde çok sayıda kişinin paralel yollara girdiklerini varsayalım. Bu durumda, içlerinden birinin derhal en kolay yolu bularak arızayı en kısa sürede halledeceği gayet açıktır. Proje sorumlusu, bu durumu görüp yeni sürümü çıkaracak ve aynı arızayı başka şekilde çözmeye çalışanlar da boş yere çabalamaktan kurtulacaklardır [RJ].

Gül Ne Zaman Gül Değildir?

Linus'un yaklaşımını incelemiş ve yaklaşımın başarısının nedeni üstünde bir teori kurmuş olarak, teorimi (daha az karmaşık ve daha az iddialı)

yeni projemde sınamaya karar verdim. Ancak yaptığım ilk şey popclient'ı yeniden düzenleyerek basitleştirmek oldu. Carl Harris'in uygulaması gayet anlamlı olmakla birlikte, çoğu C programcısı için gereksiz bir karışıklık arz ediyordu. Koda aşırı merkezi bir yer vermişti, veri yapılarına da kod desteği muamelesi yapmıştı. Sonuç olarak kodlar çok güzeldi; ama veri yapısının dizaynı duruma özgüydü ve ziyadesiyle çirkindi (en azından bu kıdemli LISP programcısının yüksek nitelikleri açısından öyleydi).

Kodu ve veri yapısı dizaynını iyileştirmeye ek olarak, yeniden yazmamın başka bir amacı daha vardı. O da kendi bildiğim bir aşamaya vardırmaktı. Bilmediğin bir programın arızalarının giderilmesinden sorumlu olmanın eğlenceli bir yanı yoktur. İşin ilk bir ayı gibi, sadece Carl'ın temel tasarımının devamını getiriyordum. Yaptığım ilk ciddi değişiklik, bir IMAP desteği eklemek olmuştu. Onu da, protokol makinelerini jenerik bir sürücüde ve üç metot tablosunda (POP2, POP3 ve IMAP) yeniden düzenleyerek yapmıştım. Bu ve önceki değişiklikler, programcıların, özellikle de C gibi doğal olarak dinamik yazım yapılmayan dillerle çalışanların akılda tutması gereken şu genel ilkeyi betimliyordu:

> 9. Akıllı veri yapıları ve aptal kodlar, kodların akıllanması ve veri yapısının aptallaşmasından çok daha iyi işler.

Brooks dokuzuncu bölümde şöyle der: "İşin içinden çıkmamı istemiyorsan, akım şemanı gösterip tablolarını gizlersin. Yok eğer tablolarını gösterirsen, akım şemanı görmeme gerek kalmaz; çünkü her şey ortadadır." 30 yıllık terminolojik ve kültürel kaymadan sonra gelinen yer, yine aynı nokta.

Bu noktada (1996 Eylül'ün başı gibi, başlangıçtan altı hafta sonra), bir ad değişikliği icap ettiğini düşünmeye başlamıştım - zaten artık sadece bir POP istemci olmaktan çıkmıştı. Yine de karar veremiyordum. Çünkü tasarımımda öz itibarıyla yeni denilebilecek bir şey, henüz yoktu. Benim popclient versiyonum kendi kimliğini geliştirmemişti. Ama hepsi, popclient'ın alınan [*:fetched*] postayı SMTP porta iletmeyi öğrenmesiyle kökten değişti. Yine de, daha önce de dediğim gibi, ben bu projeyi Linus Torvald'ın yaptığıyla ilgili teorimi sınamakta kullanmaya karar vermiştim. Peki nasıl yaptın, diye sorabilirsiniz. Şu şekilde:

- Erken ve sık yayınladım (10 günde 1'i neredeyse hiç aşmadı; geliştirmenin yoğun olduğu dönemlerde günde 1 oldu).

- Fetchmail konusunda benimle temasa geçen herkesi ekleyerek beta listemi genişlettim.
- Her yeni yayınlamada beta listemdekilere sohbet mahiyetinde duyurular yaptım, herkesi katılmaya teşvik ettim.
- Beta sınayıcılarımın dediklerine kulak verdim; dizaynla ilgili kararlarda fikirlerini aldım; yama ve bilgi beslemesi yaptıkları her durumda minnettarlığımı belirttim.

Bütün bunların geri dönüşü çok hızlıydı. Projenin başından itibaren pek çok geliştiricinin gıpta edeceği kalitede arıza raporları aldım; ki bazılarında çok iyi düzeltmeler ekliydi. Aklı başında eleştiriler, taraftar postaları ve zekice özellik önerileri aldım. Vardığı yer şu idi:

> 10. Beta sınayıcılarınıza en değerli kaynaklarınız olarak muamele ederseniz, en değerli karşılığı vereceklerdir.

Fetchmail'ın dikkate değer bir başarısı, projenin beta listesinin, yani fetchmail dostlarının büyük boyutudur. Bu yazının son revizyonunda (Kasım 2000'de) listede 287 üye vardı ve haftada 2 ya da daha çok artıyordu.

Aslında Mayıs 1997'de gözden geçirdiğimde, listedeki üye sayısının 300'e yaklaşmışken gerilediğini görmüştüm. Sebebi çok ilginçti: Çeşitli kişiler, fetchmail'ın çok iyi çalışması nedeniyle liste trafiğini görmeye ihtiyaçları kalmadığı için, kendilerini listeden çıkarmamı istemişlerdi! Bu belki de pazar tarzındaki olgun bir projenin normal yaşam döngüsünün evresiydi.

Popclient, Fetchmail oluyor

Projenin gerçek dönüm noktası, Harry Hochheiser'ın bana gönderdiği, istemcinin makinesindeki SMTP porta iletilen postayla ilgili taslak kod idi. Özelliğin güvenilir bir uygulamasının diğer bütün posta teslim modlarını tedavülden kaldıracağını neredeyse bir anda fark ettim.

Haftalardır ilerleme kaydetmeden ince ayar yapmakla meşguldüm ve arayüz tasarımının yararlı ama pis bir iş olduğu hissine kapılmıştım -kabaydı, her yanı yetersiz seçeneklerle doluydu. Alınan postayı bir posta kutusu dosyasına veya standart bir çıkışa yollama seçenekleri beni özellikle çok sıkıyordu üstüne üstlük sebebini de keşfedemiyordum. (Internet postanın teknik incelikleriyle ilgilenmeyenler sonraki iki paragrafı okumasa da olur.)

SMTP iletimi üzerinde düşünürken gördüm ki: popclient, çok fazla şeyi halletmekle uğraşmıştı. Hem mektup ulaştırma birimi [:MTA - *mail transport agent*] hem de yerel teslim birimi [:MDA - *mail delivery agent*] olacak şekilde tasarlanmıştı. SMTP iletimi açısından MDA işini defetmek mümkündü. Yani, giden mektupta olduğu gibi yerel teslim için mektubun çekilip çevrilmesi diğer programlara bırakılabilirdi.

Port 25'in TCP/IP destekli herhangi bir platformda yer aldığı neredeyse garantiyken, ilk iş olarak bir MDA'yı konfigüre etme yahut posta kutusunda kilitle-ekle [:*lock-and-append*] ayarı zahmetine girmenin gereği neydi ki? Özellikle de getirilen mektuba gönderenin başlattığı normal bir SMTP mektup gibi bakmak mümkün iken; ki aslında bizim istediğimiz şey tam da budur. (Yine daha yüksek bir düzeye ...)

Önceki teknik jargonu izlemediyseniz bile, bu noktada önemli bazı dersler vardır. İlki, bu SMTP iletimi konseptidir ki: Linus'un yöntemlerini bilerek taklit etmenin en büyük getirisi olmuştur. Bu müthiş fikri bana bir kullanıcı vermişti -bana da sonuçlarını anlamak kalmıştı.

> 11. İyi fikirlere ulaşmada bir sonraki en iyi adım, kullanıcılardan gelen iyi fikirleri elde etmektir. Bazı durumlarda bir sonraki daha da iyidir.

Çok ilginçtir, başka insanlara minnettarlığınızın ölçüsü konusunda alçakgönüllü bir içtenlik gösteriyorsanız, şunu hemen fark edersiniz: Dünya size sanki bütün her şeyi siz keşfetmişsiniz gibi davranır ve siz de doğuştan gelen dehanız konusunda daha da yakışan bir alçakgönüllülüğe kavuşursunuz. Bunun ne kadar etkili olduğu Linus'a bakılarak görülebilir! (Ağustos 1997'de Perl Konferansında konuşmamı yaparken, ön sırada olağanüstü programcı Larry Wall oturuyordu ve konuşmamın yukarıdaki son satırına gelirken, yüksek sesle ve huşu içinde "Anlat, birader, anlat!" demişti. İzleyiciler çok gülmüştü çünkü aynı etkinin Perl'ün mucidi için geçerli olduğunu da biliyorlardı.)

Aynı ruh hali içinde geçen birkaç haftadan sonra ben de, kullanıcılarımdan değil ama sözümün kulaklarına gittiği başka insanlardan benzer tepkiler almaya başlamıştım. O mektuplardan bazılarını sakladım. Eğer yaşantımın buna değer olduğunu düşünmeye başlayacak olursam; o mektupları tekrar tekrar gözden geçireceğim gayet tabii :). Ama yine, genel olarak her türlü tasarımla ilgili olan, temel nitelikte, gayri-siyasi iki ders daha var:

12. Genellikle en çarpıcı ve yenilikçi çözümlere, problem konseptinizin yanlış olduğunu fark etmekle kavuşursunuz.

Popclient'ı her türlü yerel teslim modlarıyla birlikte bir MTA/MDA birlikteliği şeklinde geliştirmeye çalışırken yanlış bir meseleyi halletmeye çalışıyordum. Fetchmail'ın dizaynının baştan alınıp, SMTP-konuşan Internet posta patikasının bir parçası olarak salt bir MTA şeklinde yeniden planlanması gerekiyordu.

Geliştirme sürecinde duvara toslarsan -yani kendini yeni yama konusunda düşünmeye adamış halde bulursan-, doğru yanıta ulaşıp ulaşmadığını değil, fakat doğru soruyu sorup sormadığını sormanın zamanı gelmiş demektir. Muhtemelen de problem yeniden belirlenmeyi gerektirir.

Haliyle, problemi yeniden belirledim. Açıkçası, yapılması gereken şey (1) SMTP iletim desteğini jenerik sürücüye çevirmek, (2) bunu açılış değeri [*:default mode*] yapmak ve (3) diğer bütün teslim modlarını, özellikle de dosyaya_teslim ve standart_çıkışa_teslim şıklarını sepetlemek idi.

Üçüncü adımda bir süre tereddüt ettim. Çünkü poclient kullanıcıların uzun süreden rahatsız olup başka teslim düzeneklerine bağlanmasından korkuyordum. Teorik olarak kullanıcılar, aynı sonucu almak üzere, *.forward* dosyalara yahut kendi mektup gönderimi olmayan eş değerlerine dönebilirlerdi. Pratikte ise, her şey birbirine girebilirdi.

Fakat bitirdiğimde gördüm ki: faydaları çok büyük oldu. Sürücü kodunun en bayıcı yerini atlatmıştım. Konfigürasyon özlü bir şekilde sadeleşti -sistem MDA'sı ve kullanıcı posta kutusu için yerlerde sürünmeye, işletim sistemi destek dosyalarının kilitlenmesiyle ilgili kaygılara gerek kalmadı.

Ayrıca, mektubun kaybolma ihtimali de kalktı. Teslim dosyaya yapılacağı zaman disk dolu ise, mektup kayboluyordu. Oysa SMTP iletiminde buna imkân yoktu; çünkü mesaj teslim edilmedikçe yahut ileride teslim için beklemeye alınmadıkça; SMTP dinleyiciniz OK veremez ki.

Performans da yükselmişti (tek çalışmada fark edilmese bile). Bu değişimin önemi olmayan bir diğer faydası, kılavuz sayfasının sadeleşmesi olmuştu. Daha sonra, dinamik SLIP'in de içinde olduğu birtakım karışık durumların halli için teslimatı, kullanıcı tanımlı bir yerel

MDA ile geri getirmem gerekiyordu. Ama çok kolay bir yolunu buldum.

Kısadan hisse? Eğer verimlilik kaybına yol açmayacaksa miyadı dolmuş özellikleri silip atmakta hiç duraksamayın. ("Küçük Prens"in yazarı) Antoine de Saint-Exupéry (çocuklara kitap yazmadığı zamanlarda havacı ve hava taşıtı tasarımcısıydı) şöyle dediğini anımsadım:

> 13. "Mükemmele (tasarımda) eklenecek bir şey kalmadığı zaman değil, eksiltilecek bir şey kalmadığı zaman ulaşılır."

Kod daha iyi ve daha yalın oldukça, *doğru* olduğunu bilirsiniz. Nitekim süreç içinde fetchmail tasarımı, atası poclient'tan farklı, kendi kimliğini kazandı. Artık isim değişikliğinin zamanıydı.

Yeni ürün, eski istemciden (popclient) çok, gönderimin (sendmail) düaline benzemişti. İkisi de MTA idi; ama sendmail önce itiyor, sonra teslim ediyor iken; yeni popclient önce çekiyor, sonra teslim ediyordu. Ben de bu nedenle fetchmail (alınan mektup) olarak adlandırdım.

Bu olayda SMTP tesliminin fetchmail'e varışıyla ilgili olarak daha genel bir ders vardır. Parallelleştirilebilen şey, sadece arıza giderme değildir; geliştirme ve (daha şaşırtıcı olanı) tasarım uzayının keşfi de öyledir. Geliştirme modunun iteratif yanı, süratliyse; geliştirme ve genişletme, arıza gidermenin -"ihmal hatasını", yazılımın özgün kapasitesi yahut konsepti içinde onarmanın- özel hallerine dönüşürler.

Tasarımın daha üst düzeylerinde bile, ürününüze yakın tasarım uzayında rasgele dolaşan geliştirme ortaklarının çokluğu, anlamlı olabilir. Suyun akacak yol bulmasını veya karıncaların yiyecek bulmasını düşünelim: Keşif, zorunlu olarak yayılmaya dayanır; onu makul bir iletişim mekanizması ile çalışan sömürü izler. Süreç gayet iyi işler. Harry Hochheiser ve benim durumumda, eşlikçilerimizden biri, bizim göremeyecek kadar yoğun olmamızdan dolayı, pekâlâ çok büyük iş başarabilir.

Fetchmail Gelişiyor

Temiz ve yenilikçi bir dizayn olmuştu. Kodlar, iyi çalışıyordu; zira her gün kullanıyordum ve beta listem gelişiyordu. Bir süredir başkalarının işine yarayabilecek sıradan programlara ilişmediğimi fark etmem, biraz zaman aldı. Unix kutusu ve SLIP/PPP posta bağlantısı olan her programcının ihtiyaç duyduğu bir programla uğraşmaya başladım.

Program, SMTP iletim özelliği ile, rekabet ortamında muhtemelen bir "kategori katili" oldu; hatta yerini öylesine doldurdu ki, rakipleri sadece geçilmekle kalmayıp unutulmaya yüz tuttular.

İnsan sanıyorum, böyle bir sonucu hedefleyemez. Sonuçlarının kaçınılmaz, doğal ve sipariş edilmiş görünmesi, tasarıma büyük bir güçle dalmış olmayı gerektirir. Bu türden fikirlerle uğraşmanın yegâne yolu, çok çeşitli fikirlere ya da sonuçları yaratıcılarınca görülemeyen iyi konseptleri kapacak mühendis bakışına sahip olmaktan geçer.

Andy Tanenbaum'un, eğitimde kullanılmak üzere IBM PC'lere yönelik, (Minix adını verdiği) sade bir Unix yapma fikri vardı. Linus, bu konsepti Andy'nin düşünmediği bir yere taşıdı -geliştirip harika bir noktaya vardırdı. Benzer şekilde ben de, Carl Harris'in ve Harry Hochheiser'ın geliştirdiği birtakım fikirleri zorladım- ama çok daha küçük ölçekte. Hiçbirimiz romantik tasavvura uyan "özgün" dahiler değildik. Zaten, programcı mitolojisindekinin aksine; bilimin de, mühendisliğin de, yazılım geliştirmenin de büyük kısmı, özgün dahilerce yapılmış değildir.

Yine de bayağı baş döndürücüydü -aslında, her programcının can atacağı türdendi. Ayrıca standartlarımı yükseltmemin icap ettiğini ifade ediyordu. Artık fetchmail, benim gördüğüm kadar iyi olabilecekse; sadece kendi ihtiyaçlarım için değil, fakat benim sahanlığım dışında kalanlar için de gereken özellikleri içermeli ve desteklemeliydi; yalınlığı ve bütünlüğü devam ederken, elbette.

Bunu fark ettikten sonra yazdığım ilk ve etkileyici nitelikte önemli sonuç, dağıtım [*:multidrop*] desteği idi -bir kullanıcı grubu için birikmiş tüm postayı posta kutusundan çekip alabilme ve sonra da her birini tek tek alıcılarına yönlendirme yeteneği.

Dağıtım desteğini ekleme kararımın ardında, kısmen birtakım kullanıcılardan gelen şikâyetler vardı; ama asıl neden, tek dağıtım kodunun beni, adresleme bütünlüğü içinde kalarak arızaların hepsini temizlemeye zorlayacağını düşünmemdi. Öyle de oldu. RFC 822 adres ayrıştırıma hakkını almak [http://info.internet.isi.edu:80/in-notes/rfc/files/rfc822.txt] epey bir zaman aldı. Ama bu durum tek bir parçasının zor olmasından değil; bir dolu birbiriyle bağıntılı ve yorucu ayrıntıdan kaynaklanıyordu. Sonuçta, dağıtım adreslemesinin, tasarım olarak mükemmel bir seçim olduğu çıktı. Bundan eminim, çünkü:

14. Her aletin işlevi doğrultusunda iş görmesi gerekir; ama gerçekten

> müthiş bir alet, işlevlerinin çok ötesinde iş görür.

Dağıtımlı fetchmail'ın sürpriz bir kullanımı da, postaları çekip çevirirken listeyi ve Internet bağlantısının istemci tarafında takma ad açılımlarını korumasıdır. Bu, bir ISP hesabı üzerinden kişisel cihazla çalışan birinin, ISP'nin farklı (takma) adlı dosyalarına sürekli olarak erişmesine gerek kalmadan, posta listesini idare edebileceği anlamına gelir.

Beta sınayıcılarımın istediği bir diğer önemli değişiklik, 8-bit MIME (Muhtelif amaçlı İnternet Posta Uzatmaları, [*:Multipurpose Internet Mail Extension*]) operasyonlarına destek idi. Çok kolay bir işti. Çünkü, başından beri kodda 8-bit temizliğe (yani, ASCII karakter setindeki 8. biti, program içinde bilgi taşımak için kullanmak üzere boşta tutmaya) dikkat etmiştim. Sebebi de, bu özelliğin talep edileceğini tahmin etmekten ziyade, şu kurala riayet ediyor olmamdı:

> 15. Şu ya da bu türden bir gateway yazılımı üretirken; veri akışını bozmamaya itina etmek -ve alıcı mecbur bırakmadıkça bilgileri *asla* ziyan etmemek.

Bu kurala uymasaydım 8-bit MIME desteği, zor ve arızalı olurdu. Daha önce de olduğu gibi, yapmam gereken tek şey, MIME standardını (RFC 1652 [http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1652.txt]) okumak ve bir başlık_üretim lojiği biti eklemekti.

Avrupa'daki bazı kullanıcılar, seans başına alınan mesaj sayısını sınırlama seçeneği eklediler (bu sayede pahalı telefon şebekelerinden doğan masrafları denetleyebilirler). Bu konuda epeyce direndim ve hâlâ da memnun sayılmam. Ancak, insan dünyaya program yazınca, müşterilere kulak vermeye mecbur kalıyor -para ödenmiyor olması, bu durumu değiştirmez.

Fetchmail'dan Çıkan Birkaç Ders Daha

Genel yazılım mühendisliği mevzularına dönmeden önce, fetchmail deneyiminden çıkarılacak belli başlı birkaç ders daha var. Teknik açıdan ilgilenmeyen okurlar bu bölümü atlayabilirler.

rc (kontrol) dosyasının sentaksında, ayrıştırınca toptan gözardı edilecek, seçimlik "gürültü [*:noise*]" anahtar kelimeleri bulunur. Bunların mümkün kıldığı İngilizceye benzeyen sentaks, geleneksel kısa anahtar

kelime çiftlerinden çok daha daha anlaşılırdır. Bütün bunlar gecenin geç bir vaktinde, *rc* dosyası deklarasyonlarının kurallı bir mini dile benzemeye başladığını fark ettiğim zaman başladı. (Orijinali "server" olan popclient anahtar kelimesini "poll" ile değiştirmemin sebebi de budur.)

Bu kurallı mini dili İngilizce'ye benzetmek, kullanımını kolaylaştıracakmış gibi gelmişti bana. Aslında hâlâ tasarım alanında, Emacs ve HTML ve daha pek çok veri tabanı motorları ile örneklenen, "bir dil oluştur" ekolünün mümin bir yandaşı olmaya devam etmekle birlikte, artık genel olarak "İngilizce_benzeri" sentaksların büyük bir taraftarı değilim

Programcılar, geleneksel olarak, hassas, derli toplu ve fazlalıksız kontrol sentakslarına eğilimlidir. Bu, bilgisayar kaynaklarının pahalı olduğu dönemlerin kültürel mirasıdır. Aşamalar, mümkün olduğunca ucuz ve yalın olacak şekilde bölümlenmek durumundaydı. Oysa yüzde 50'si fuzuli olan İngilizce, bunun için hiç de münasip bir model değildi.

Burada İngilizce'ye benzer sentakslardan kaçınmak için mazeret üretmiyorum. Ucuz çevrimler ve kalp (iç, çekirdek, [*:core*]) itibariyle, özlülük kendi başına bir sonuç olmak durumunda değil. Bugün dilin insanlara uygun olması, bilgisayar açısından iktisadi olmasından daha önemli.

Bununla birlikte tedbirli olmak için çok anlaşılır sebepler vardır. Bunlardan biri de aşamalandırmanın karışıklık maliyetidir -durumu, kullanıcının kafasını karıştıran ve adeta bir arıza, sorun kaynağı haline getirmeyi kimse istemez. Bir diğeri İngilizce benzeri bir dil sentaksı yapmaya çalışmanın, programın konuştuğu "İngilizceyi" fazlaca zorlamasıdır. Doğal dile yüzeysel benzerlik, geleneksel sentaks kadar kafa karıştırıcı olurdu. (Bu kötü etkiyi "dördüncü nesil" denilen ticari veri tabanı sorgulama dillerinde görürsünüz.)

Fetchmail kontrol sentaksı bu türden sorunlardan uzaktı; çünkü dil temeli son derece kısıtlanmıştı. Her işi gören bir dil yoktur. Söylenecek şeyler hiç de öyle dolambaçlı değildir. Dolayısıyla, sınırlı bir İngilizce ile fiili kontrol dili arasında zihinsel geçişte karışıklığa yol açacak çok az şey vardır. Buradan çıkan etraflı ders de şudur:

> 16. Dil, bilgisayımsal tamlık/yetkinlik düzeyinde değilse (*Turing testinden geçme yeterliliğine sahip* [:Turing-complete] *değilse*), sentaktik tatlar [*:syntactic sugar*] ile haşır neşir olunabilir.

Bir diğer ders de karartma yoluyla güvenliktir. Birtakım fetchmail kullanıcıları, benden parolaları *rc* dosyasında şifreleyerek saklayacak şekilde yazılımı değiştirmemi istediler. Böylece özel bilgilerin peşinde olanların tesadüfen ulaşması mümkün olmayacaktı.

Ama ben yapmadım, çünkü ek koruma sağlamayacaktı. *rc* dosyanızı okuma izni edinen herhangi biri, tıpkı sizin gibi fetchmail'ı çalıştırır -peşinde oldukları sizin parolanız ise, elde etmek için gereken dekoderi fetchmail'ın kendi kodundan çıkarabilirlerdi. Tüm *.fetchmailrc* parola şifrelemesi, güvenli olmadığını düşünen insanlara yanlış bir güvenlik hissi verirdi. Genel kural şudur:

17. Bir güvenlik sistemi ancak sırrı kadar güvendedir. Yani sözümona sırlara dikkat.

Pazar Tarzının Gerekli Önkoşulları

Bu yazının ilk gözden geçirilmesine ve sınamasına girenler, sürekli bir şekilde, başarılı pazar tarzını geliştirmenin ön koşullarıyla ilgili sorular sordular. İçlerinde proje liderinin nitelikleri, insanlara açılma sırasında kodun durumu ve ortak geliştiriciler topluluğu kurmanın başlangıcı vardı.

Kodun pazar tarzı içinde geliştirilmeyeceği yeterince açıktır [IN]. Sınama, arıza giderme ve iyileştirme pazar tarzı içinde yapılabilse de, *üretim* mümkün değildir. Linus hiç denememişti. Ben de denemedim. Büyüyen geliştirme topluluğunun oynayacağı, işleyen ve sınanabilir bir şeye ihtiyacı vardır.

Topluluğu kurmaya başladığınızda hazırlamış olmanız gereken şey *makul* bir *vaattir.* Programı o sırada ille de iyi işlemesi gerekmez. Ham, arızalı, eksik ve hatta dokümanları bile eksik olabilir. Asla başarısız olunmaması gereken de (a) işler durumda olması ve (b) menzil içinde bir gelecekte temiz bir program olacağı konusunda geliştirme ortaklarını ikna etmesidir.

Hem Linux hem de fetchmail, insanlara güçlü ve çekici bir temel tasarımla gitti. Bu yazıda sunduğum pazar modeli üzerinde düşünen pek çok kişi, bu eşiği doğru değerlendirdi ve sonra da buradan tasarımın kurumsal derecesinin yüksekliğinin ve proje liderinde zekânın vazgeçilmez olduğu sonucunu çıkardılar vardılar.

Linus, tasarımını Unix'ten almıştı. Ben de ilk önce atası mahiyetindeki popclient'tan almıştım (yine de, Linux'un tasarımındakine kıyasla, çok

daha ciddi bir ölçüde değişmişti). Peki o halde, pazar tarzı bir çalışmada liderin/koordinatörün müstesna bir tasarım yeteneğine sahip olması şart mıdır, yoksa başkalarının bu yeteneklerinde kaldıraç olması yeterli midir?

Bence koordinatörün parlak tasarımlar üretebilmesi pek o kadar kritik önem taşımaz. Asıl, *başka fikirlerin parlaklığını teşhis edebilmek* kritik önem taşır.

Linux da, fetchmail da bunun kanıtıdır. Linus, hiç de öyle (yukarıda değinildiği gibi) parmak ısırtan bir tasarımcı değilken; parlak tasarımı teşhis etmekte çok güçlü bir beceri sergilemiş ve bunu Linux çekirdeğine entegre etmiştir. Yine, fetchmail'daki en güçlü tasarım fikrinin (SMTP iletimi) başkasına ait olduğuyla ilgili süreci, az önce ben de anlattım.

Yazının ilk okuyucuları, çalışmanın büyük kısmının bana ait olması sebebiyle, pazar projelerinde tasarımın özgünlüğünü hafife alma eğiliminde olduğumu söyleyerek bana iltifat ediyorlardı. Bunda gerçeklik payı olduğu söylenebilir; kesinlikle, (kod geliştirme ve arıza gidermeye nazaran) tasarımcılık, benim en güçlü yanım.

Yazılım geliştirmede zeki ve özgün olmakla ilgili mesele, alışkanlıklardır -bir şey sağlam ve sade olması gerektiğinde, insan tepkisel olarak onu sevimli ve karmaşık hale getirmeye başlıyor. Bu hataya düşüp, kırıp döktüğüm projeler olmuştu; ama fetchmail'da bundan kaçınmayı başardım.

Bence fetchmail projesi, kısmen, parlak olma eğilimimi sınırlamam sayesinde başarılı oldu; bu da pazar projelerinin başarısı için tasarımın özgünlüğü şartına karşı çıkmak demektir. Ayrıca Linux'a bakalım. Diyelim ki Linus Torvalds işletim sistemini geliştirirken, tasarımındaki temel yeniliklerle uğraşıp durmuş olsun; böyle bir durumda, sonuçta ortaya çıkan çekirdeğin, şimdi sahip olduğumuz kadar istikrarlı ve başarılı olması muhtemel midir?

Belli düzeyde bir tasarım temeli ve kod geliştirme becerisi elbette şarttır; zaten pazar modelinde bir çabaya girişmekte niyeti ciddi olan birinin, bu asgari düzeyin üstünde olması beklenir. Açık_kaynak çevresinin meşhur iç piyasası, rekabeti sürdüremeyecek olanlara, girişimden uzak durmaları konusunda ustalıklı bir baskı hissettirir. Bunun da şimdiye kadar oldukça iyi işlediği söylenebilir.

Normalde yazılım geliştirmeyle eşleşmeyen, benim pazar projelerinde tasarım zekâsı kadar -hatta belki ondan da- önemli gördüğüm bir diğer

beceri vardır. Pazar projesinin koordinatörü yahut lideri, iletişimde ve insan ilişkilerinde becerikli olmalıdır.

Sebebi çok açıktır. Bir geliştirme topluluğu kurmak için insanları cezbedebilmek, yaptığın işle ilgilendirmek ve yapacakları çalışmadan mutluluk duymalarını sağlamak gerekir. Teknik çevrenin bunu halletmek için daha epeyce yolu var; henüz o noktada değil. Ayrıca kişisel özellikler de öyle.

Linus'un sevilen ve yardım edilen iyi bir insan olması tesadüf değildir. Benim de, kalabalıkta çalışmayı seven ve adeta tek başına meddahlık yapan bir komiğin güdülerine sahip, enerjik ve dışa dönük biri olmam tesadüf değil. Pazar modelini işler kılmak için, insanları etkileyecek ölçüde bir becerinin müthiş yararı vardır.

Açık_kaynaklı Yazılımların Toplumsal İçeriği

Şu hakikaten doğrudur: en iyi programlar, yazarının günlük sorunlarına bulduğu kişisel çözümlerle başlar ve yayılır; çünkü o sorun, büyük bir kullanıcı kitlesi için de bir tipiklik arz eder. Bu da bizi birinci kuralın meselesine geri götürür

> 18. İlginç bir sorunu çözmek için; işe, ilginç olabilecek bir sorun bulmakla başlayın.

Carl Harris için atadan kalma popclient ile benim için fetchmail böyle bir şeydi. Ama durum uzunca süre önce anlaşılmıştı. Asıl, ilginç olan sonraki safha -yazılımın büyük ve aktif bir geliştirme ortakları ve kullanıcılar topluluğu içindeki evrimi-, yani Linux'un ve fetchmail'ın yakından bakmaya çalıştığımız geçmişlerine ilgi duyulması meselesi ilginçtir.

The Mythical Man-Month adlı kitabında Fred Brooks, program zamanının, değişim dışı olduğunu fark etti. Daha önce de gördüğümüz gibi, bir projede iletişim maliyetlerinin ve karmaşıklığının geliştirici sayısının karesiyle yükseldiğini ama ürünün doğrusal olarak arttığını ileri sürmüştü. Brooks Kanunu, tartışmasız bir gerçek olarak kabul edilir. Ama biz bu yazıda, bunun ardındaki varsayımların açık_kaynak geliştirme sürecine uymadığını gösteren biçimleri inceledik -Brooks Kanunu'nun doğru olsaydı, Linux'un vücut bulmasına imkân yoktu.

Gerald Weinberg'in, *The Psychology of Computer Programming* adlı klasiğinde Brooks'u düzelttiğini -iş işten geçtikten sonra- görürüz.

Yazar, "egosuz programlama" tartışmasında, geliştiricilerin kodları konusunda mülkiyetçi olmayıp, koddaki arızaları bulma ve kodu iyileştirmeye gitme konularında insanları teşvik etmeleri durumunda, gelişmenin dikkat çekici şekilde daha hızlı olduğunu fark etti. (Yakın zamanda, Kent Beck'in fikri olan, birbirinin omuzu üstünden bakacak şekilde konuşlanan kodcularla "ekstrem programlama" tekniği, bu sonuca varma çabası olarak görülebilir.)

Weinberg'ün analizinin layık olduğu kabulü görmesini engelleyen şey, terminoloji tercihidir; belki -Internet programcıları için kullandığı "egosuz" sıfatı, insanı tebessüm etmekten alıkoyamıyor. Yine de, argümanının her zamankinden daha kuvvetli olduğunu düşünüyorum.

Pazar yöntemi, "egosuz programlama" etkisinin tam güçle çalışmasını dizginleyerek Brook Kanunu'nun etkisini hafifletebilir. Brook Kanunu'nun ardındaki ilke, ortadan kalkmış değilse de; geliştiricilerin çoğalması ve iletişimin ucuzlaması halinde; zaten başka türlü görülemeyen yarış halindeki doğrusal_dışı durumlarla, sonuçları baskılanabilir.

Bu durum Newton ve Einstein fiziklerinin ilişkisine benzer. Eski sistem, düşük enerji düzeylerinde geçerlidir. Ama kütleyi ve hızı yeterince zorlayacak olursanız, nükleer patlamalara ya da Linux'a yol açarsınız. Unix'in geçmişinin Linux'tan (ve benim tarafımdan daha küçük ölçekte doğrulanmış Linus yöntemlerinden [EGCS]) öğreneceklerimize hazırlamış olması gerekir. Yani, kod yazımı zaruri bir tek kişi etkinliği olarak kalmaya devam ederken; gerçek anlamda büyük çözümler, bütün topluluğun zihin gücünün ve dikkatinin yönlendirilmesinden gelir. Kapalı bir projede sırf kendi beynini işe koşan bir geliştirici, tasarım ortamını besleyen buluşların, kod katkısının, arıza tespitinin ve diğer iyileştirmelerin yüzlerce (belki de binlerce) insandan geldiği açık ve evrimci bir çerçevede çalışan geliştiricinin gerisinde kalacaktır.

Ne var ki geleneksel Unix dünyasının bu yaklaşımı sonuna kadar götürmesini engelleyen muhtelif unsurlar vardı. Bir tanesi farklı lisanlar, ticari sırlar ve çıkarlar ile ilgili yasal kısıtlamalardı. Diğeri de (tabii bunu sonradan anlıyoruz) Internet'in henüz yetkin olmamasıydı.

Ucuz İnternet'ten önce coğrafi olarak kapalı birtakım topluluklarda Weinberg'ün "egosuz" programlaması vardı ve bir geliştirici pek çok izleyiciyi ve geliştirme ortağını kolaylıkla cezbedebilirdi.

Bell Labs, MIT AI ve LCS labs, UC Berkeley -hepsi efsanevi ve hâlâ potansiyel taşıyan yeniliklerin yuvası oldular. Linux, bütün *dünyayı* bir

yetenek havuzu olarak değerlendirmede bilinçli ve başarılı olan bir ilk proje oldu. Linux'un kuluçka dönemiyle World Wide Web'in doğumunun örtüşmesinin bir tesadüf olmadığını düşünüyorum. Nitekim Linux da bebeklik döneminden 1993-94'te, ISP endüstrisinin kalkışa geçtiği ve İnternet'in patladığı dönemde çıktı. Yaygın İnternet erişiminin getirdiği yeni kurallara göre oynamayı öğrenen ilk kişi, Linus olmuştur.

Ucuz İnternet, Linux'un evriminde gerekli bir koşul olmakla birlikte, bizzat o haliyle yeterli bir koşul değildi. Diğer kritik faktör de liderlik tarzının ve geliştiricilerin ortak geliştiricileri cezbederek ortamdan maksimum desteği almayı sağlayan işbirliği alışkanlıklarının gelişmesi olmuştu.

Peki ama bu liderlik tarzı nedir, alışkanlıklar nelerdir? Bunlar güç ilişkilerine yaslanmazlar. Yaslansalar bile dayatmacı liderlik ile bu sonuca varılmaz. Weinberg, 19 yüzyıl Rus anarşisti Pyotr Alekeyeviç Kropotkin'in *Memoirs of a Revolutionist* adlı otobiyografisinden şu alıntıyı verir:

> "Serf sahibi bir ailenin çocuğu olarak dünyaya gelmiş olmakla birlikte, dönemimin tüm gençleri gibi, kumandanın, emrin, azarın, cezanın ve sairenin gerekliliği konusunda hiçbir kuşkunun olmadığı bir aktif yaşama dahil olmuştum. Ancak, erkence bir evrede, çok ciddi girişimleri idare etmem, (özgür) insanlarla uğraşmam gerekmişti ve yapılacak bir hatanın sonuçlarının ağır olması durumunda, kumanda ve disiplin temelinde iş görmekle, kavrayış temelinde iş yapmak arasındaki farka şükran duymaya başlamıştım. İlkinde hayranlık verici bir askeri gösteriş vardı; ama gerçek hayat söz konusuyken, hiçbir değeri olmadığı gibi, amaç da ancak birbirine yakın niyetlerin ciddi çabası ile gerçekleşebilirdi."

İşte Linux gibi bir projenin gereksindiği de bu "birbirine yakın niyetlerin ciddi çabası" idi. Ayrıca İnternet dediğimiz anarşizm cennetindeki gönüllülere "emir" vererek sonuç almak da imkansızdır. İş bölümü gerektiren projelerin başında olmak isteyen programcıların, verimli bir şekilde yarışmak ve çalışmak için, Kropotkin'in "kavrayış temelinde" diye belli belirsiz önerdiği çıkar birliğini bir araya getirmeyi ve beslemeyi öğrenmek durumundalar. Linus Kanunu'nu öğrenmeleri ise şarttır [SP].

Linus Kanunu'nun açıklaması mahiyetinde "Delfi etkisine" atıfta bulunmuştum. Ancak, biyolojideki ve iktisattaki adaptif sistemlere

yönelik benzetmeler de vardır. Linux dünyası, pek çok yönden, içinde fayda maksimizasyonu yapan bencil varlıklar topluluğunun yer aldığı, düzeltmeleri kendiliğinden yapan ve merkezi bir planlamayla erişilebilecek olandan daha çeşitli ve verimli bir sürece, bir serbest piyasaya veya ekolojik bir ortama benzer. Yani, "kavrayış temelinde" olmayı aramanın yeri burasıdır.

Linux programcılarının maksimize ettiği "fayda fonksiyonu" klasik anlamda iktisadi değil, fakat programcıların kendi ego tatminleri ve diğer programcılar içindeki ünleri gibi, göze görünmezdir. (Yani güdülenmelerinin ardında "fedakârlık" olduğu söylenebilirse de, bu durum fedakârlığın, öz itibarıyla, fedakâr olan açısından bir ego tatmin biçimi olduğu gerçeğini göz ardı eder.) Bu şekilde işleyen gönüllülük kültürleri bulunmaz değildir; örneğin bir tanesi benim de içinde olduğum bilimkurgu hayranlarıdır. Ama bu gönüllülük, temelinde gönüllü faaliyetin esas motifi olan "ego büyümesi"nin (hayranlar arasındaki ününü artırmanın) yer aldığı programcılıktakinden farklıdır.

Linus, kendini, geliştirmenin esas olarak başkalarınca yapıldığı bir projenin gözetmeni olarak başarıyla konumlandırmakla ve projeye ilgiyi sürdürülebilir hale gelene kadar beslemekle, Kropotkin'in "ortak kavrayış temeli"ni derinlemesine kavradığını göstermişti. Linux dünyasıyla ilgili iktisadi gibi duran bu bakış, bu kavrayışın ne kadar uygulandığını görmemizi sağlıyor.

Linus'un yöntemini "ego büyütme"de -tek tek programcıların bencilliğini, ancak kalıcı bir işbirliğiyle ulaşılabilecek kadar zor amaçlara mümkün olduğunca sağlam bir şekilde bağlayacak olan- etkili bir piyasa oluşturmanın yolu olarak görebiliriz. Linus'un yöntemlerinin iyi sonuçlar verebileceğini (daha küçük ölçekte olmakla birlikte) fetchmail projesi ile göstermiştim. Belki, ona kıyasla, durumun daha bir farkındaydım ve daha sistematiktim. Pek çok kişi (özellikle de serbest piyasaya inanmayanlar), kendi yönünü çizen bir egoistler kültürünün bölgesellik, gizlilik, husumet ve müsriflik yüzünden dağılmasını bekler. Ancak bu bekleyişin yanlışlığı, Linux dokümanlarındaki dikkat çekici çeşitlilik, kalite ve derinlik ile açık bir şekilde ortaya çıkmıştır. Programcıların doküman hazırlamaya duydukları *nefret* meşhurdur. Buna rağmen nasıl oldu da Linux yazılımcıları o kadar çok doküman üretebildiler?

Linux'un egoya dayalı serbest piyasası, ticari yazılım üreticilerinin ciddi bir şekilde fonlanan dokümantasyon odaklarınca yönlendirilene kıyasla, virtüöz üretmekte daha iyi çalıştığı açıktır.

Hem fetchmail, hem de Linux çekirdeği projeleri gösteriyor ki: güçlü bir geliştirici/koordinatör, başka yazılımcıların da egolarının ödülünü gereğince alması sayesinde, İnternet'i pek çok geliştirme ortağının çıkarına hitap etmekte kullanarak, projenin uçurumdan yuvarlanıp gitmesini engelleyebilir. Bu noktada Brooks Kanunu'nun tam karşıtı olan şunu öneriyorum:

> 19. En az İnternet seviyesinde bir iletişim ortamıyla donatılmış bir geliştirme koordinatörü, dayatmacı olmadan önderlik yapabilirse; çok sesliliğin tek seslilikten iyi olacağı kesindir.

Açık_kaynaklı yazılımın gelecekte Linus'un oyununu oynamayı bilenlere, katedralden çıkıp pazarda dolaşanlara ait olacağını düşünüyorum. Ama bununla, bireysel vizyonun ve yaratıcılığın hükmünün kalmayacağını söylemiyorum; tersine, asıl bireysel vizyon ve yaratıcılık sahiplerinin açık_kaynaklı yazılımda en ön safta yer alacağını ve gönüllü çıkar birliğini daha doğru bir şekilde inşa ederek yapıyı daha büyüteceğini söylüyorum.

Açık_kaynaklı yazılımın tek geleceği bu olmak zorunda değil. Hiçbir kapalı-kaynak geliştirici, Linux topluluğunun bir sorunun çözümü için harekete geçireceği yetenek havuzuyla aşık atamaz. Çok azı, para ödeyerek bile, fetchmail için katkıda bulunan 200 (1999'da 600; 2000'de 800) kişiden daha fazlasını toplayamaz!

Sonunda zafer açık_kaynak kültürünün olacaksa da, bunun nedeni işbirliğinin ahlâken doğru ya da yazılım "istiflemenin (gömülemenin)" ahlâken yanlış olması değil (Linus'un ve benim düşüncemin aksine ikincisine inandığınızı varsayarak), fakat kapalı_kaynak dünyasının, problem çözümünde daha çok vasıflı zamanı örgütleyebilen açık_kaynak topluluklarıyla gireceği yarışı kazanamayacak olmasıdır.

Yönetim ve Maginot Hattı Üstüne

1997'deki *Katedral ve Pazar* yukarıdaki görüşle -örgütlü ve mutlu programcı/anarşist ordusunun, geleneksel kapalı ve hiyerarşik yazılım dünyasıyla kapışıp hakkından geldiği ile- sona eriyordu.

Ancak buna ikna olmayan çok kuşkucu vardı ve soruları yanıtlanmaya değerdi. Pazar tezine gelen itirazların çoğu şu sava dayanır: Bu tezin sahipleri, geleneksel yönetimin üretim artırıcı etkisini hafife almıştır.

Gelenekçi yazılım geliştirme yöneticilerinin itirazları, genelde, açık_kaynak proje gruplarının oluşum, değişim ve dağılma durumlarındaki rahatlığın, grubun herhangi bir kapalı kaynak geliştirici karşısındaki sayısal avantajını önemli ölçüde yitirmesiyle ilgilidir. Onlar, kaç kişinin çorbada tuzunun olduğuna değil, ilgili ürüne sürekli yatırım bekleyebilecek müşterilere ve çabanın gelecekte de kalıcı olmasına bakacaklardır.

Bununla noktayla ilgili bir yanıt var elbette; aslında, yazılım üretimi iktisadiyatında kilit meselenin beklenen hizmet değerinin olduğu fikrini *The Magic Cauldron* (*Sihirli Kazan*) adlı yazıda geliştirmiştim [http://www.tuxedo.org/- ~esr/writings/magic-cauldron/].

Ancak bu sav, ciddi bir sorunu barındırır; söz konusu çabanın, açık kaynaklı geliştirmede kalıcı olamayacağı şeklinde zımni bir kabulü vardır. Tersine, çizgisini tutarlı bir şekilde koruyan açık_kaynak projeleri yürütülmüş ve yürütenler de, geleneksel yönetim için kaçınılmaz olan teşvik ya da kurumsal denetim mekanizmaları olmaksızın, çalışma boyunca çabalarını sürdürmüşlerdir. GNU Emacs editörünün geliştirilmesi uç ve öğretici bir örnektir; yüksek temposuna ve sadece bir kişinin (yazarının) kesintisiz aktivitesine rağmen, yüzlerce insanın katkısını içeren ve 15 yıl süren birleşik bir mimari süreçtir. Hiçbir kapalı kaynak editörü, bu sürenin yakınına ulaşamamıştır.

Bu durum, geleneksel yönetilen yazılım geliştirmenin avantajlarının sorgulanması için, katedral-pazar karşılaştırmasıyla ilgili diğer tüm gerekçelerden bağımsız bir bir gerekçe sunuyor. Eğer GNU Emacs gibi 15 yıldan uzun zaman tutarlı bir yapısal süreç sergilemişse; yahut benzer şekilde, Linux gibi bir sistem için, hızla değişen donanım ve platform teknolojisi içinde 8 yıldan fazla sürebilmişse; ayrıca (ki asıl önemli olan da budur) mimarisi güçlü ve 5 yıldır süren çeşitli açık_kaynak projeleri varsa, bu durumda, geleneksel yönetilen geliştirme projelerinin anormal giderleri, aslında bizi satın almak için mi acaba, diye merak etmemiz gerekir.

Kesin olan şey, güvenilir bir uygulamanın mühlet, bütçe ya da bu türden diğer spesifikasyon özellikleri demek olmadığıdır; yukarıdaki hedeflerden değil üçüne, sadece birine bile ulaşan bir proje, olağanüstü "yönetilmiş" bir projedir. Ayrıca, projenin ömrü boyunca, iktisadi süreçteki ve teknolojideki değişmelere adapte olma kabiliyeti de ortada yoktur; nitekim, açık_kaynak topluluğunun bu anlamda da açık ara etkili olduğu kanıtlanmıştır (bunu, 30 yıllık İnternet tarihini, ağ teknolojilerinin kısa yarı_ömürleriyle kıyaslayarak görmek hemen mümkündür. Ayrıca Microsoft Windows'un 16 bitten 32 bite geçiş

maliyetiyle, Linux'un aynı dönemde neredeyse sıfır çabayla geçişi de aynı bağlamdadır ki: üstelik sadece Intel üzerinden değil, 64 bit Alpha da dahil diğer bir düzine donanımı kapsar).

Çoğu insan, geleneksel modelin projenin yanlış gitmesi halinde tazminat ödemek üzere yasal bir sorumlu tanımlamasıyla, işin bittiğini düşünür. Ancak bu bir yanılsamadır; çoğu yazılım lisansları, bırakın performansı bir kenara, ticari garantinin geçersizlik hallerini belirtmek üzere yazılır -tabii yazılımdaki sorunların nasıl halledileceği meselesi neredeyse buharlaşmıştır. Hadi, diyelim öyle olmasa bile, birini bu yüzden dava açmak zorunda bırakmak, zaten esası ıskalamak demektir. Çünkü alıcının amacı, mahkemelerde dolaşmak değil; aldığı yazılımın çalışır durumda olmasıdır.

Peki o zaman bu yönetim gideri denilen şey neyin nesidir? Bunu anlamak için yazılım geliştirme yöneticilerinin yaptıklarını sandıkları şeye bakmak gerekir. Işinde çok iyi olduğu izlenimini veren tanıdığım bir kadın, yazılım yönetiminin beş tane fonksiyonu olduğunu söylüyor:

- *Hedef koymak* ve insanları hedefe yöneltmek;
- *İzlemek* ve önemli ayrıntıları gözden kaçırmamak;
- Sıkıcı ve angarya işlerde insanları *motive etmek*;
- En yüksek verimliliği sağlamak için insan dağılımını *örgütlemek*;
- Projeyi sürdürmek için gereken *kaynakları düzenlemek*.

Bunların değerli hedefler olduğu açık. Ancak açık_kaynak modeli altında ve onun içinde yer aldığı toplumsal kurguda, bütün bunlar tuhaf bir şekilde ilgisiz görünmeye başlayabilir. Sondan başa doğru gidelim. Arkadaşımın dediğine göre *kaynak düzenleme* tamamen defansif nitelikte. Çalışanlar, makineler ve ofis, bunların benzer kaynakları kullanarak yarışa giren rakip yöneticilerden ve zaten sınırlı olan havuzu verimsiz kullanabilecek üstlerden korunmasını gerektiriyor.

Gel gör ki: açık_kaynak geliştiriciler, gönüllü kimselerdir; çalıştıkları projeye ilgileri, katabilecekleri olması nedeniyle kendi tercihleriyle gelir (ve bu durum açık_kaynak program için kendilerine bir ödeme yapılması halinde de genellikle böyledir). Yani kaynakları düzenlemedeki bu "saldırı", gönüllülük ile kendiliğinden hallolmaktadır. Çünkü herkes masaya kendi kaynağıyla gelir. Dolayısıyla, "savunucu rolünde" bir yöneticiye olan ihtiyaç çok azdır -hatta hiç yoktur. PC'lerin ucuz ve İnternet'in hızlı olduğu bir dünyada, sınırlı yegâne kaynağın uzmanlaşmış yoğunlaşma [*:skilled attention*] olduğu kanısındayız.

Açık_kaynak projeleri battığında, sebebi makine, bağlantı ya da yer sıkıntısı değil, geliştiricilerinin ilgilerini yitirmesidir. Şu nokta, ondan iki kat daha önemlidir: Açık_kaynakta katılım, yazılımcının kendi tercihiyle olduğu için *örgütlenme* maksimum verimlik sağlar -ayrıca sosyal ortam rekabet sürecindeki seçimde acımasızdır. Hem açık_kaynak dünyasına hem de kapalı büyük projelere alışkın olan arkadaşım, açık_kaynağın başarısının, kısmen, kültürü dolayısıyla, programcıların en yetenekli yüzde 5'i gibi bir kısmını razı etmesinden geldiğini düşünüyor. Bu durumda zamanının büyük kısmını geri kalan yüzde 95'inin yerini organize etmeye harcıyor. Böyle bizzat, en yetkin programcılar arasında verimlilik dağılımını tespit etmiş oluyor.

Bu dağılımın büyüklüğü şu acayip soruyu getiriyor: Bireysel projeler ve alanın bütünü, içindekilerin en az yarısı yetkin olmasa, fonksiyonel olur muydu? Aklı başında yöneticiler, uzun zamandır farkındalar ki: geleneksel yazılımda yönetimin yegâne işlevi, oynamaya değmez bir oyunda, kesin bir kaybı sınırlı bir kazanca çevirmekten ibarettir.

Açık_kaynak topluluğunun başarısı, bu sorunun ciddiyetini arttırıyor. Çünkü başka işlerle uğraşmak yerine binalara doluşup yönetilen insanlarla yapılana kıyasla, İnternet'ten derlenen gönüllülerle daha ucuz ve sonuca yönelik iyi işler üreterek somut kanıtlar sunuyor.

Böylece *motivasyon* meselesine geliyoruz. Arkadaşımın bununla ilgili söylediği şey: öz olarak, geleneksel geliştirme yönetiminin, aksi takdirde iyi iş çıkaramayacak olan kötü motive edilmiş programcıların yerini doldurduğudur.

Bu yanıt, genellikle, beraberinde şöyle bir savla gelir: Açık_kaynak topluluğu sadece "seksi" işlerde güvenilirdir. Buna göre, yöneticilerin kamçıladığı, parayla motive edilen yığınlar olmasa, başka şeyler yapılmadan kalır. Bu sav hakkındaki kuşkularla ilgili psikolojik ve sosyal gerekçeler konusunda bakınız: *Homesteading the Noosphere* [http://www.tuxedo.org/ ~esr/magiccauldron/].

Öte yandan buradaki amacımızla ilgili olarak, varılacak sonucun ilginçliği açısından bu savın doğru olduğunu kabul edeceğiz. Geleneksel kapalı_kaynak ağır_yöneticiliği tarzındaki yazılım geliştirme, sadece sıkıntı yaratan sorunlarda adeta bir Maginot Hattı olma yönüyle savunulabilir.

Öte yandan buradaki amacımızla ilgili olarak, varılacak sonucun ilginçliği açısından bu savın doğru olduğunu kabul edeceğiz. Geleneksel kapalı_kaynak ağır_yöneticilik tarzındaki yazılım geliştirme, sadece sıkıntı yaratan sorunlarda Maginot Hattı olma yönünden, yani şu ya da

bu uygulamada kimsenin ilginç bulmayacağı için aşmak üzere bir yol aramayacağı durumlar açısından işe yarar. Çünkü açık_kaynak sürecindeki rekabette, müşteriler, yazılımın "sıkıcı" kısmıyla ilgili olarak söz konusu problemin, problemi cazip gören kimse tarafından çözüldüğünü bilecektir. Bu ise, çözen yazılımcı için, diğer yaratıcı çalışma alanlarında olduğu gibi, paradan çok daha güçlü bir güdüleyicidir. Sırf güdüleme için gereken geleneksel bir yönetim yapısı, taktik olarak iyiyse de, stratejik olarak kötüdür. Kısa dönemde kazançlıdır, ama uzun dönemde kesin kayıp demektir.

Buraya kadar geleneksel geliştirme yönetiminin açık_kaynak karşısında iki kalemde (kaynakların düzenlemesinde ve organizasyonda) kötü bir seçenek olduğunu gördük. Üçüncüsü (motivasyon) açısından hâlâ dirençli gibi duruyor. Zayıf donanmış geleneksel yönetici için *izleme ve denetim* bir çare olmayacaktır. Açık_kaynak topluluğunun en güçlü yanı adem-i merkeziyetçi gözden geçirme özelliğidir ki: detayları gözden kaçırmamaya yönelik bütün geleneksel yöntemleri aşar.

Hedefleri belirleme'yi geleneksel yazılım geliştirme ek külfeti (yönetim sarfiyatları) için bir gerekçe sayabilir miyiz? Belki. Ama böyle yapmak için, yönetim komitelerinin ve şirket yol haritalarının, değerli ve geniş paylaşımlı hedefleri tanımlamada açık_kaynak dünyasındaki karşılıkları olan proje liderlerinden ve kabile büyüklerinden daha başarılı olduğuna inanmayı sağlayacak iyi bir sebep bulmamız icap eder.

Bunun da kolay olmadığı aşikar. Ama terazinin açık_kaynak tarafı (Emacs'ın ömrü, yahut Linus Torvald'ın "dünya egemenliği [*:world domination*]" sözüyle geliştirici ordularını harekete geçirme gücü) açısından onların işini zorlaştırmak da o kadar zor değil. Kaldı ki geleneksel mekanizmaların yazılım projelerinin hedefini belirlemedeki fecaati ortadadır.

Yazılım mühendisliğiyle ilgili malum teoremlerden biri, yüzde 60 ile yüzde 75 arasında geleneksel projenin ya hiç tamamlanmamış ya da hedef kullanıcılarınca reddedilmiş olmasıdır. Bu oranlar doğruya yakınsa (ki tekzip eden yöneticilerle hiç karşılaşmadım), bu durumda ya (a) gerçekleşebilir olmayan, ya da (2) yanlış projeler, hedefe yönelik projelerden sayıca daha fazla oluyor.

Günümüzün yazılım dünyasında "yönetim komitesi" tamlamasını duyan herkesin -bizzat yöneticinin bile- tüyleri ürperir. Sadece programcıların homurdandıkları o günler geride kaldı; artık *idarecilerin* masalarında Dilbert karikatür bantları okunuyor.

Bu durumda geleneksel yöneticiye vereceğimiz yanıt basittir: -tamam, açık_kaynak çevresi geleneksel yöneticinin gerekliliğini ciddiye almıyor, *peki, sizin bu kadar iğrenmenizin sebebi ne?*

Açık_kaynak topluluğu, bu soruyu sormakta son derece haklı. Çünkü biz yaptığımız işten zevk alıyoruz. Bizim yaratıcı çalışmamız, teknik, pazar payı ve düşünsel paylaşım başarılarını şaşırtıcı düzeylere çıkarıyor. Yani sadece daha iyi yazılım üretebildiğimizi değil, fakat bunun *zevkli bir iş* olduğunu da kanıtlıyoruz.

Bu yazının ilk yayımından iki buçuk yıl sonra, kendimi yakın gördüğüm en radikal düşünce, açık_kaynak egemenliğinde bir yazılım dünyası vizyonunun olmadığıdır. Bunun yerine yazılım (ve belki de her türden profesyonel yahut yaratıcı çalışma) ile ilgili daha kapsamlı ders mahiyetinde bir öneride bulunmak istiyorum. İnsanlar, sıkılmanın zor olduğu, fakat sonuca varmanın kolay olmadığı optimal bir düzeyde kalmak kaydıyla, işle boğuşmaktan zevk alırlar. Mutlu programcı atıl kalmayan fakat kötü formüle edilmiş hedeflerle de uğraşmayan, stres yüklü biridir. *Haz, verimliliğin habercisidir.*

Çalışma süreci kaşısında duyulan korkuyu ve tiksintiyi (isterse Dilbert karikatürleri asarak matrak bir şekilde dile gelsin), sürecin çöküşünün açık kanıtı olarak değerlendirmek gerekir. Eğlence, mizah duygusu, oyun keyfi önemli varlıklardır; yukarıda yazdığım "mutlu ordular" laf olsun diye söylenmiş değildir. Kaldı ki Linux'un maskotunun cana yakın ve genç bir penguen olması da, salt bir espiri olmaktan ötedir.

Sanıyorum, açık kaynağın başarısının en önemli sonuçlarından biri, yaratıcı çalışmanın iktisaden en verimli yönteminin oyun olduğunu öğretmesi olacaktır.

Epilog: Netscape Pazarı Kucaklıyor

İnsanın kendisinin, tarihin oluşumunda katkısı olduğunu fark etmesi, tuhaf bir his...

22 Ocak 1998'de, Katedral ve Pazar'ın ilk yayımından yaklaşık yedi ay sonra, Netscape Communications Inc., Netscape Communicator'ın kaynağını karşılıksız vereceğini duyurdu [http://-www.netscape.com/newsref/pr/newsrelease558.html]. Durumu ben de duyurunun yapıldığı gün öğrenmiştim.

Netscape'in başkan yardımcısı ve teknoloji dairesinin başı Eric Hahn bana sonradan şu e-postayı yolladı: "Netscape'teki herkes adına, bu

noktaya varmamızda öncelikle sana teşekkür borçluyum. Bu karara varmamızda ilham kaynağımız senin düşüncelerin ve yazıların oldu."

Sonraki hafta (4 Şubat 1998'de), Netscape'in davetlisi olarak tam gün süren strateji konferansı için, üst yöneticileri ve teknik kadrosuyla birlikte Silikon Vadisi'ndeydim. Birlikte Netscape'in kaynak sürüm stratejisini ve lisanslarını kurguladık.

Birkaç gün sonra şunu yazmıştım:

> "Netscape, pazar modelinin ticari dünyada pratik sınanması yolunda kapsamlı bir imkân sunuyor. Artık açık_kaynak kültürü bir tehlikeyle yüzyüze. Netscape'in bu uygulaması işlemezse, açık_kaynak konsepti öylesine gözden düşebilir ki; ticari dünyada en az on yıl kendine gelemeyebilir. Öte yandan bu, olağanüstü bir fırsat. Wall Street'te ve başka yerlerde ilk tepkiler, ihtiyatlı ve olumlu. Kendimizi kanıtlamak için de, bir şans aynı zamanda. Netscape bu hamleyle ciddi bir pazar payı elde ederse, yazılım sektöründe beklenen devrim gerçekleşmiş olacak. Önümüzdeki yıl çok ilginç ve öğretici bir dönem olacak.

Oldu da, gerçekten. 2000 ortasında yazdığım gibi, sonradan Mozilla adını alan proje, büyük başarı kazandı. Netscape'in başlangıç hedefini gerçekleştirdi. Bu, Microsoft'un gezgin piyasasına attığı kilidi kırmak demekti. Ayrıca çok ciddi başarılara imza attı (Gecko motorunun yeni neslini sürdü).

Yine de Mozilla'nın yaratıcılarının umduğu gibi, Netscape dışında belirgin bir sonuç elde edilemedi. Buradaki sorun, Mozilla'nın dağıtımında pazar modelinin temel kurallarından birinin uzunca bir süredir çiğnenmiş olması gibi görünüyor. Katkı vermesi muhtemel kimselerin kolayca çalıştırıp işlediğini görebilecekleri şeyleri içermiyordu. (Piyasaya sürülmesinin üstünden bir yıl geçmesine rağmen, Mozilla'nın kaynaktan kuruluşu, Motif kitaplığı lisansı gerektiriyordu.)

En olumsuz nokta da (dış dünyanın bakışı itibarıyla), projenin başlamasından sonraki iki buçuk yılda, Mozilla grubunun ürün nitelikli bir gezgin çıkarmamış olmasıydı -bir de, 1999'da projenin yürütücülerinden birinin, beceriksiz yönetim ve kaçırılan fırsatları vurgulayarak ayrılmasının yol açtığı sansasyon vardı. Doğru bir tespitle, "Açık_kaynak, sihirli peri tozu değildir" diyordu.

Haklıydı. Mozilla'nın uzun dönemli gelişimiyle ilgili tahmin, Jamie Zawinski'nin ayrıldığı zamana nazaran şimdi (Kasım 2000'de) daha

olumlu görünüyor. Nihayet, son birkaç haftanın gece sürümlerinde, kullanılabilme eşiğini geçmiş durumda. Ancak, Jamie, bir projenin sırf açık olmakla yanlış belirlenmiş hedeflere, spagetti koda ya da yazılımcının kronik hastalıklarına karşı koruma sağlaması gerekmediğini söylerken haklıydı. Mozilla, açık kaynağın bir yandan başarılı, öbür yandan da başarısız olma koşullarını göstermek açısından bir örnek olmuştur.

Bununla birlikte, aynı dönemde açık_kaynak düşüncesi çeşitli başarıları hanesine yazdı. Netscape sürümüyle, açık_kaynak geliştirme modeline yönelik müthiş bir ilgiye, Linux işletim sisteminin başarısıyla süren ve başarının devam ettiği bir trende tanık oluyoruz. Öte yandan Mozilla'nın başarısı da görece artan bir hızla yoluna devam ediyor.

Açıklamalar

[JB] Programing Pearls'de Jon Bentley, Brooks'un tespitini, "Bir şeyi heba etmeyi planlarsan, iki şeyi de heba edersin" diye yorumluyor. Neredeyse kesinlikle haklı. Brooks'un tespiti ve Bentley'in yorumu, ilk denemenin hatalı olmasını beklemek gerektiğini değil, doğru düşünceyle başlamanın, bir yıkıntıyı kaldırmaya çalışmaktan daha anlamlı olduğunu söylüyor.

[QR][QR] Başarılı açık_kaynak örnekleri, pazar geliştirmesi Internet patlamasının öncesine denk geliyor ve Unix ve İnternet gelenekleriyle ilgisiz. 1990-x1992 arasındaki info-Zip sıkıştırması [http://www.cdrom.com/- pub/infozip/] projesinin hedefi DOS makinelerdi. Bir diğeri RBBS bülten sistemiydi (yine DOS içindir). 1983'te başlamış ve yeterince güçlü bir topluluk geliştirmiştir. Bugüne (1999 ortasına) kadarki sürümler, İnternet postadaki devasa teknik avantajlara ve yerel BBS üzerinden dosya paylaşımına rağmen, makul aralıklarla yapılmıştır. info-Zip topluluğu bir ölçüde İnternet postaya bel bağlarken, RBBS geliştirimi, RBBS üzerinde TCP/IP alt yapısından tamamen bağımsız, ciddi bir on-line topluluk zemini oluşturabilmiştir.

[CV][CV] Bu şeffaflık anlayışı, yeni bir konsept olmadığı ortaya çıkan işletim sistemi geliştirmenin karmaşıklığını terbiye etmek açısından değerli. 1965'te, zaman paylaşımlı işletim sistemleri döneminin başlarında, Multics işletim sisteminin tasarımcıları Corbató and Vyssotsky, şöyle yazmışlardı [http://www.multicians.org/fjcc1.html]:

> "Multics sistemin, gereğince işlemeye başladığında, yayınlanması umuluyor. Bu, iki açıdan anlamlı: İlki, okuyucu nezdinde sistemin kurcalanmaya ve eleştiriye dayanıklılığını görmek;

ikincisi de, giderek artan bu karmaşa çağında, bugünün ve geleceğin sistem tasarımcılarına, temel sistem meselelerini ortaya koyacak ölçüde yalın bir dahili işletim sistemi göstermek."

[JH][JH] John Hasler'in, çabanın ikiye katlanmasının, açık_kaynak geliştirme önünde kesin bir engel gibi görünmediğine dair ilginç bir açıklaması vardır. Benim "Hasler Kanunu" adını verdiğim önerisi şöyle: İkiye katlanmış işin maliyeti, ekibin boyutuyla karesel ölçekten az -yani, planlanlama ve kurtulunmak istenilen yönetim giderinden (ek külfetlerden) daha yavaş- artma eğilimindedir .

Bu açıklama Brooks Kanunu ile çatışmaz. Toplam karışıklık külfeti ve arızalar karşısındaki zaaf, ekibin boyutunun karesi ölçüsünde olabilse de, *ikiye katlanmış* işin maliyetinin daha yavaş artması özel bir durumdur. Bu konuda anlaşılır sebepler bulmak zor değil. İlk ve apaçık olgu şudur: Farklı geliştiricilerin kodları içinde çabayı ikiye katlamayı önleyecek fonksiyonel sınırlar üzerinde uzlaşmak, bütün sistemi saran, planlanmamış kötü etkileşimleri engellemekten kolaydır.

Linus Kanunu ile Hasler Kanunu'nun birleşiminin önümüze getirdiği şey, yazılım projelerinde üç tane kritik boyut olduğudur. Küçük (örneğin en çok üç geliştiricinin olduğu) projelerde yönetim yapısının olmaması, bir baş programcı (ya da takım lideri) seçmekten çok daha iyidir. Orta projelerde geleneksel yönetim maliyeti (ek külfetler) görece düşer (marjinal kalır) ve çabanın ikiye katlanmasından kaçınma ve arıza takibi (hata takibi [:*bug_tracking*]) gibi yararları yanında, detayları kaçırmama yönünden de pozitif etki edecektir.

Ancak, yine bu iki kanunun birleşimi, büyük projelerde, geleneksel yönetim sorunlarının ve sarfiyatlarının, çabanın ikiye katlanmasına bağlı beklenen masraflardan çok daha fazla artacağını ileri sürer. Söz konusu masraflar, arızaları ve detayları gözden kaçırmamak konusunda geleneksel yönetimden daha iyi işleyen çok sayıda katkı koyabilecek izleyen gözü devreye sokmada, yapısal bir acz anlamına gelir. Dolayısıyla, büyük projelerde, söz konusu kanunların birleşimi, geleneksel yönetimin net katkısını sıfıra indirir.

[HBS][HBS] Linux'un deneysel ve kararlı versiyonları arasındaki ayrım, riskin denetimiyle ilgilidir fakat bu denetimin dışındadır. Bu ayrım başka bir soruna toslar: Temrinlerin (mühletlerin) öldürücülüğü. Programcılar hem değişmez bir özellik listesine hem de sabit bir son tarihe bağlı olduklarında, kalite düşer ve tasarımda devasa bir karmaşa başlar. Harvard Business School'dan Marco Iansiti ve Alan

MacCormack'a, bu sınırlamalardan birini gevşetmenin, şedülü iş görür hale getirebildiğini gösterdikleri için müteşekkirim.

Bunun bir yolu, termini düzeltmek fakat özellik listesini esnek bırakmak ve böylece son tarihten önce yetişmemeleri halinde özelliklerin azalmasına izin vermektir. Bu, "kararlı" çekirdekte zorunlu stratejidir. Alan Cox (kararlı çekirdeğin sorumlusu) yayınları makul aralıklarla yapar; ancak belli arızaların düzeltilme zamanını veya hangi özelliklerin deneysel çekirdek şubesinden alınacağını garanti etmez.

Bir diğer yol da istenilen bir özellik listesi hazırlayıp sadece tamamlandığında teslim etmektir. "Deneysel" çekirdek şubesinin stratejisi de budur. De Marco ve Lister araştırması, bu şedül politikasının ("bittince beni uyandır") sadece en yüksek kaliteyi sağlamakla kalmadığını, fakat "gerçekçi" veya "agresif" şedüllerden, ortalama olarak, daha kısa teslim süreleri ürettiğini gösterir.

Bu yazının (2000 başlarındaki) ilk versiyonların "bitince beni uyandır" şeklindeki mühlet-karşıtı politikanın açık_kaynak topluluğunun üretkenliği ve kalitesi açısından önemini ciddi şekilde küçümsediğimden kuşkulanır olmuştum. Acilen yapılan GNOME 1.0'ın 1999 sürümündeki genel deneyim, prematüre sürüm baskısının, açık kaynağın normalde getirdiği kalite yararlarından çoğunu nötralize ettiğini gösterir.

Açık_kaynak sürecinin şeffaflığı, yani kalite, "bitince beni uyandır" şedülü ve geliştiricinin kendi tercihi şeklindeki üç ortak sürücü sayesinde olduğu böylece ortaya çıkmış oluyor.

[SU][SU] Açık_kaynak projelerinin kalp_artı_hale [*:core_plus_halo*] şeklindeki örgüt karakteristiğini, Brooks'un N'nin karesi karmaşıklığındaki problemin çözümüne yönelik tavsiyesiyle ve İnternet sayesinde "operasyon timi" örgütü olarak görmek, çekici fakat tam anlamıyla doğru değildir. Brooks'un ekip lideri üzerinden şekillendirdiği "kod kütüphanecisi" gibi uzman roller gerçekte yoktur; bu roller, aksine, Brooks'un zamanına göre epeyce güçlü alet setleriyle destekli çok yönlü kimselerce icra edilir. Ayrıca, açık_kaynak kültürü ciddi bir şekilde Unix geleneğinin modülerliğine, API'lerine ve istihbaratın saklanmasına dayanır –ki: hiçbiri Brooks'un reçetesinde yer almamıştır.

[RJ][RJ] Bir arızayı karakterize etmenin güçlüğüyle ilgili iz patikalarının büyük ölçüde değişen boylarının etkisi konusunda dikkatimi çekmek üzere cevap veren kişi, aynı arızanın çeşitli semptomları itibarıyla, iz patikasının "üstel"olarak değiştiğini düşünüyordu (ben Gaussian veya Poisson dağılımı kullanırım, ama fikrine katıldım). Dağılımın şekli

üzerinde bir fikre varmak deneysel olarak mümkün olsa, o veri son derece değerli bir veri olurdu. İz güçlüğüyle ilgili basık bir eşit olasılık dağılımından vazgeçmek demek, zamanı sınırlayarak pazar stratejisini taklit etmek zorunda olan münferit geliştiricilerin bile, bir diğerine geçmeden önce verili bir semptomu izlemeye zaman ayırdıklarını söylemektir. Israrcılık, her zaman meziyet olmayabilir...

[IN][IN] Bir projeyi pazar tarzı içinde sıfırdan başlatıp başlatmamakla ilgili bir mesele de, pazar tarzının gerçekten yenilikçi bir projeyi destekleyip destekleyemeyeceğidir. Bazılarının iddiasına göre, pazar, güçlü liderlik eksiği nedeniyle, mühendislik sanatının hazır fikirlerini iyileştirmek yahut klonlamak dışında bir şey yapamaz. Bu sav belki de en rezil şekliyle Halloween Dokümanları'nda [http://www.opensource.org/halloween/], Microsoft'un utanç verici iki notunda yer alır. Yazarlar, Linux'un Unix benzeri işletim sistemi olarak gelişimini "kuyruğa takılmak" ile kıyaslıyor ve "(bir proje en üst "değerine" ulaştığı zaman) yeni cepheleri zorlayacak bir yönetim düzeyi kaçınılmaz olur" diye fikir beyan ediyorlar.

Bu savda ciddi olgusal hatalar var. Bir tanesi Halloween yazarlarının sonradan bizzat gözledikleri durumda ortaya çıkıyor: "genelde [...] yeni araştırma fikirleri, müsait olmadan/başka platformlara eklenmeden önce ilk olarak Linux'ta uygulanırlar."

Eğer "açık_kaynak" ifadesini "Linux" olarak okursak, bunun yeni bir durum olmadığını görürüz. Bir kere, tarihsel olarak, Emacs ya da World Wide Web yahut bizzat Internet, açık_kaynak topluluğu tarafından kuyrukçulukla yahut yönetilerek icat edilmiş değildir. Ayrıca şu anda açık_kaynak dünyasında o kadar çok yenilikçi çalışma yürümektedir ki, bir tanesini seçmek çok zordur. GNOME projesi (sırf seçmiş olmak adına) GUI'lerde ve nesne teknolojisinde en üst düzeyi zorlar. Tek başına bu bile bilgisayar ticari basınında büyük çaplı bir ilgi konusudur. Sayısız örnek vardır. Freshmeat'e [http://freshmeat.net/] bir ziyaret, bunu günlük düzeyde kanıtlayacaktır.

Ama asıl, çok daha temel nitelikte bir hataya işaret eden şu örtülü varsayım yapılmaktadır: Güvenilir bir şekilde yenilik yapmak *Katedral modeline* (veya pazar modeline, ya da başka türden bir yönetim yapısına) özgüdür. Bu saçmalıktır. Çeteler hamle edecek basiretten yoksundur -bırakın statükoya göre yaşam savaşı veren şirket komitelerini, gönüllü pazar anarşistleri bile çoğu durumda hakiki bir özgünlük gösteremezler. *Basiretin kaynağı bireydir.* Onu çevreleyen toplumsal mekanizmanın umup umabileceği tek şey, bu hamle basiretine *tepki* verebilmektir

-ezmek yerine zenginleştirmek, özüllendirmek yahut dikkatle sınamak şeklinde.

Bazıları bunun romantik bir görüş, modası geçmiş bir yalnız mucit şeklindeki ilkel figürün [*:stereotype*] tersi olduğunu söyleyecektir. Öyle değil; ben burada grupların bir kez ortaya çıktıktan sonra, hamle basireti *geliştirmekten* aciz olduklarını iddia etmiyorum; tersine, nitelikli sonuçlara ulaşmak için böyle geliştirme gruplarının şart olduğu bir inceleme-gözden geçirme süreci sayesinde öğrendiğimize vurgu yapıyorum. Dolayısıyla, böyle her grubun, -ister istemez- bir kişinin kafasındaki iyi bir fikir nedeniyle ortaya çıktığını belirtiyorum. Katedraller, pazarlar ve diğer sosyal yapılar bu çakışı (kavrayışı) ve inceliği yakalayabilirler ama talep üzerine imal edemezler.

Bu yüzden yeniliğin özüne ilişkin sorun (yazılımda yahut başka alanda), yeniliğin nasıl ezilip geçilemeyeceğidir - ama daha da temel olarak, *öncelikle çok sayıda basiret sahibi insanın nasıl yetiştirileceğidir.*

Bu işi, süreç akışkanlığı ve düşük giriş engelleri olan pazar modelinin değil de katedral tarzının kotarabileceğini düşünmek saçma olurdu.

Eğer mesele iyi bir fikri olan biri ise, bu durumda, içindeki bir kişinin yüzbinlerce diğer insanı o fikir sayesinde hızla işbirliğine çekebildiği bir sosyal ortam, kişinin işinden atılma riski olmadan, fikri üstünde çalışabilmek için, önce bir hiyerarşiye politik satış yapmak zorunda olduğu yapıyı (yapının kendisini), kaçınılmaz bir şekilde yeniliğin dışına atacaktır (yenilik dışı bırakacaktır).

Ayrıca örgütlere göre yazılım yeniliklerinin tarihine, katedral modeli üzerinden bakarsak, ne kadar kıt olduğunu derhal fark ederiz. Büyük şirketler, yeni fikirler için üniversite araştırmalarına bel bağlarlar (zaten Halloween Dokümanlarının yazarları da, Linux'un araştırma işini hayata geçirmedeki süratinden ve işlevselliğinden rahatsızlık duyuyor). Ya da bir mucidin fikri üzerinde kurulmuş küçük firmaları satın alırlar. İki durumda da yenilik olgusu katedral kültürünün yabancısıdır. Tersine, bu şekilde iç edilen pek çok yenilik, Halloween Dokümanları yazarlarının hürmetle yücelttiği "kesif yönetim tabakası" tarafından sessiz sedasız boğulmuştur.

Ne var ki bu olumsuz bir noktadır. Okuyucuya olumlu bir şey sunmak gerekir. Bu amaçla, bir deneyim olması itibarıyla, şunu önereceğim:

- Düzenli olarak uygulayabileceğinize inandığınız bir özgünlük kriteri belirleyin. Tanımınızın "Gördüğüm zaman biliyorum" şeklinde olması, bu testin amacı açısından bir sorun olmaz.

- Linux'la rekabet halinde olan bir kapalı_kaynak işletim sistemi ve üstündeki cari geliştirmeyi izlemek için bir kaynak seçin.
- Bir ay boyunca, hem o kaynağı hem de Freshmeat'i izleyin. Her gün, "özgün" olarak belirlediğiniz işle ilgili Freshmeat'teki sürüm duyurularını sayın. Aynı "özgün" tanımını ilgili işletim sisteminin duyurularına uygulayın ve sayın.
- Otuz gün sonra her ikisini de toplayın. Benim bunu yazdığım gün Freshmeat yirmi iki sürüm duyurusu yapmıştı. Üç tanesi, en üst düzeyden işler gibiydi. O gün Freshmeat için yavaş bir gündü. Ama beni asıl hayrete düşürecek olan şey, bir kapalı_kaynakta bir ay boyunca üç tane yenilik denilebilecek bildirim çıkması olacaktır.

[EGCS][EGCS] Burada, fetchmail'dan ziyade pazarın öncülüyle ilgili gösterge netliğinde bir sınama sunabilecek bir projeyle ilgili bir öykümüz var: EGCS [http://egcs.cygnus.com/] Experimental GNU Compiler System.

Bu proje 1997'nin Ağustos'unda duyuruldu. Katedral ve Pazar'ın ilk açık versiyonlarındaki fikirleri uygulamaya yönelik bilinçli bir çabaydı. Projenin kurucuları GCC'nin (GNU C Compiler) geliştirilmesinin durgunlaştığı hissine kapılmışlardı. Yaklaşık 20 ay geçtikten sonra GCC ve EGCS paralel ürünler olarak devam ettiler. İkisi de aynı İnternet geliştirme topluluğundan ve aynı GCC kaynak temelinde idiler. İkisinde de neredeyse aynı Unix alet takımları ve geliştirme ortamı kullanılıyordu. Aralarındaki tek fark, EGCS'nin planlı bir şekilde daha önce anlattığım pazar taktiklerini uygulamasıydı. GCC daha bir katedral tarzıydı. Daha kapalı ve sürüm sıklığı düşük idi.

Durum neredeyse kontrollü bir deney mahiyetindeydi ve sonuçları da çok etkileyiciydi. Aylar geçtikçe, EGCS versiyonları, özellikler itibarıyla ciddi bir şekilde ileri gitti. Optimizasyon daha iyiydi. FORTRAN ve C++ destekleri daha iyiydi. Pek çok kişi, EGCS geliştirme enstantanelerini GCC'nin en yeni kararlı sürümünden daha güvenilir buluyordu. Başlıca Linux dağıtıcıları EGCS'ye dönmeye başladılar.

Nisan 1999'da Free Software Foundation (GCC'nin sponsoru), GCC geliştirme grubunu dağıttı ve projenin kontrolünü EGCS'nin yönlendirme ekibine teslim etti.

[SP][SP] Elbette ki Kropotkin'in eleştirisi ve Linus Kanunu, sosyal örgütlenmelerin sibernetiğiyle ilgili daha kapsamlı meseleler doğurur. Yazılım mühendisliğiyle ilgili bir diğer teorem de bunlardan biridir; Conway Kanunu -genelde, "Bir derleyici üzerinde çalışan dört

grubunuz varsa, 4-geçişli [*:4-pass*] bir derleyiciniz olur." şeklinde ifade edilir. Özgün ifadesi daha geneldir: "Sistem tasarımı yapan örgütler, bu örgütlerin iletişim yapılarının kopyası olan tasarımlar üretmekle sınırlıdır." Bunu daha açıkça şöyle de söyleyebiliriz: "Vasıtalar, sonucu tayin eder", veya "Süreç (bizzat kendisi), ürüne dönüşür."

Buna uygun olarak eklemeye değer bir nokta da, açık_kaynak topluluğunda örgütsel biçim ile işlevin pek çok düzeyde uyumlu olduğudur. Şebeke her şeydir ve her yerdedir: İnternet'ten ibaret değildir. Ancak işi yapan insanlar, dağınık, gevşek bir şekilde eşlenmiş, çok yedekli ve zarif bir şekilde rütbelenmiş, dengi-dengine [*:peer-to-peer*] bir şebeke oluştururlar. Her şebekedeki her düğümün başkaları için önemi, sadece o düğümle işbirliğine girmek istemeleri ölçüsündedir. Dengi-dengine olmak, topluluğun çarpıcı üretkenliği için vazgeçilmezdir. Kropotkin'in güç ilişkileri ile ilgili ulaşmaya çalıştığı nokta "SNAFU İlkesi" ile daha ileri götürülmüştür: "Gerçek iletişim sadece eşitler arasındadır, çünkü astlar, üstlerine doğruyu değil, onları memnun edecek yalanları söyledikleri için düzenli bir şekilde taltif edilirler." Yaratıcı ekip çalışması, doğru iletişime dayanır ve bu yüzden güç ilişkileriyle ciddi bir şekilde engellenir. Bu güç ilişkilerinden gerçek anlamda bağımsız olan açık_kaynak topluluğu, bize, bu ilişkilerin ne ölçüde arızalar olarak yansıdığını, üretkenliği düşürdüğünü ve fırsatları yok ettiğini öğretir.

Dahası, SNAFU ilkesi, otoriter örgütlerde karar yetkisinin sahipleriyle gerçeklik arasındaki bağlantı kopukluğunun büyüdüğünü, karar yetkisinin sahiplerine yönelik girişlerin giderek pohpohlayıcı yalanlara dönüşme eğiliminde olduğunu öngörür. Bu gidişi geleneksel yazılım geliştirme sürecinde izlemek çok kolaydır; astların problemleri örtmeleri, ihmal etmeleri ve küçük göstermeleri için çok güçlü özendiriciler vardır. Bu sürecin ürüne dönüşmesi, yazılımın felâket olması demektir.

Kaynakça

Frederick P. Brooks'un *The Mythical Man-Month* adlı klasik kitabından çeşitli parçalar aldım. Bunun sebebi, derin kavrayışının pek çok açıdan hâlâ geliştirilmeyi beklemesidir. Kitabın Addison-Wesley'den çıkan 25. Yıl (ISBN 0-201-83595-9) baskısını içtenlikle tavsiye ederim. Bu baskıya 1986'daki "No Silver Bullet" adlı makalesini de koymuştur.

Yeni baskı 20 yıllık paha biçilmez bir kapsamı kucaklar. Brooks, burada tüm açık sözlülüğüyle, orijinal metindeki birtakım hükümlerinin zamana yenik düştüğünü kabul eder. Makalemin tamamlanıp

yayımlanmasının ardından Brooks'u ilk okuduğumda, Microsoft'a pazar_vari pratikler yakıştırdığını şaşırarak keşfettim. (Aslında bu yakıştırmanın hatalı olduğu çıktı. 1998'de Halloween Dokümanlarından, Microsoft'un iç geliştirme topluluğunun, asla mümkün olmayacak bir pazar desteğine duyduğu genel kaynak erişimi türüyle, ciddi bir şekilde bölündüğünü öğrendik.

Gerald M. Weinberg, *The Psychology Of Computer Programming* (New York, Van Nostrand Reinhold 1971) adlı kitabında ziyadesiyle talihsizlikle_damgalı "egosuz programlama" kavramını takdim etmişti. "Komut ilkesinin" beyhudeliğini kavrayacak ilk kişi olmanın uzağında olmasına rağmen, muhtemelen, meselenin yazılım geliştirmeyle özel bağını görecek ve ileri sürecek ilk kişiydi.

Richard P. Gabriel Linux öncesi dönemin Unix kültürünü tasarlarken, 1989'da yazdığı "LISP: Good News, Bad News, and How To Win Big" adlı makalesinde ilkel bir pazar_vari modelin üstünlüğünü, gönülsüzce de olsa, ileri sürmüştü. Bu çalışması, (benim de dahil olduğum) LISP taraftarları arasında hâlâ çok tutulur. Bir muhabir bana, "Worse Is Better" başlıklı makalenin bir Linux beklentisiyle okunduğunu hatırlatmıştı. Makaleye http://www.naggum.no/worse-is-better.html adlı sayfadan erişebilirsiniz.

De Marco'nun ile Lister'in birlikte yazdığı *Peopleware: Productive Projects and Teams* (New York; Dorset House, 1987; ISBN 0- 932633-05-6) isimli çalışma değeri bilinmemiş bir eserdir. Fred Brooks'un kendi çalışmasında atıfta bulunduğunu görmek, beni çok mutlu etmişti.

Bu yazarların meselelerinin pek azı Linux'a yahut açık_kaynak topluluklarına uygulanabilir olsa da, yazarların yaratıcı çalışına koşullarıyla ilgili kavrayışları son derece keskindir ve bazıları da ticari bir kapsamda pazar modeline almak için çabaya değer niteliktedir.

Son olarak şunu teslim etmek zorundayım: Bu yazıya az daha "Katedral ve Agora" adını verecektim. İkinci terim açık pazarın yahut halka açık meydanın Yunanca adıdır. Mark Miller'ın ve Eric Drexler'ın "agorik sistemler" konusundaki tohum atan çalışmaları, piyasa_vari bilgisayımsal [*computational*] ekolojilerin doğuş özelliklerini tanımlayarak, beş yıl sonra Linux yüzünden burnumun sürtüldüğü sırada, açık_kaynak kültüründe bunlarla benzerlik gösteren olgular konusunu açık bir şekilde düşünmeme yardım etti. Söz konusu çalışmaları http://www.agorics.com/agorpapers.html'de görebilirsiniz.

Teşekkür

Bu yazı, sohbetleriyle kusurlarımızı gidermemize yardımcı olan çok sayıda kişi sayesinde gelişti. Öncelikle, "arıza gidermek paralleştirilebilir" düşüncesini öneren ve analizi o noktadan itibaren geliştiren Jeff Dutky'ye <dutky@wam.umd.edu> özellikle teşekkür ediyorum. Ardından, Kropotkin üzerinden Weinberg'ü taklit etmemi önerdiği için Nancy Lebovitz'e <nancyl@universe.digex.net>, elbette. Ayrıca, General Technics listesinden Joan Eslinger <wombat@kilimanjaro.engr.sgi.com> ve Marty Franz <marty@net-link.net> kapsamlı eleştiriler yaptılar. Glen Vandenburg <glv@vanderburg.org>, katkıda bulunan grubundakilerin kendilerini seçmelerinin önemine işaret ederek geliştirmenin çokluğunun "ihmal hatasını" düzelteceği fikrini verdi; bu kapsamda Daniel Upper <upper@peak.org> doğal analojileri öneren kişiydi.

PLUG, Philadelphia Linux User's Group adlı Philadelphia Linux kullanıcıları grubunun üyelerine, yazımın ilk açık versiyonuyla ilgili ilk test izleyicilerini oluşturdukları için şükran boçluyum. Paula Matuszek <matusp00@mh.us.sbphrd.com>, yazılım idaresi pratiğinde beni aydınlatan kişi oldu.

Phil Hudson <phil.hudson@iname.com>, programcı kültürünün toplumsal örgütlenmesi ile yazılımın organizasyonun, birbirlerinin aynası olduklarını hatırlattı. Ayrıca John Buck <johnbuck@sea.ece.umassd.edu>, MATLAB'ın Emacs için yönlendirici bir parallel yarattığına işaret etti. "Karışıklığı Kaç Gözlem Giderir" içinde ele alınan mekanizmalardan bazılarıyla ilgili olarak, aklımı Russell Johnston <russjj@mail.com> sayesinde topladım.

Son olarak, Linus Torvalds yorumlarıyla yardımcı oldu, ayrıca önceki girişimi yüreklendiriciydi.

Türkçe Çevirisi için Not:

Açık_kaynak kullanıcıları için, E.S.Raymond'un bu değerli eseri, İngilizce aslından Bileşim Yayıncılık ve TMMOB Elektrik Mühendisleri Odası işbirliği ile EMO yayın organlarında yayınlanmak üzere, Orhan Örücü'nün önerisi, Cihan Gerçek'in itinalı çevirisi ve EMO Yayın Kurulunun çabaları ile Türkçe'ye kazandırılmıştır. Tüm katkı koyanlara teşekkür ediyoruz.

Aydın Bodur, Bileşim Yayıncılık, 2008