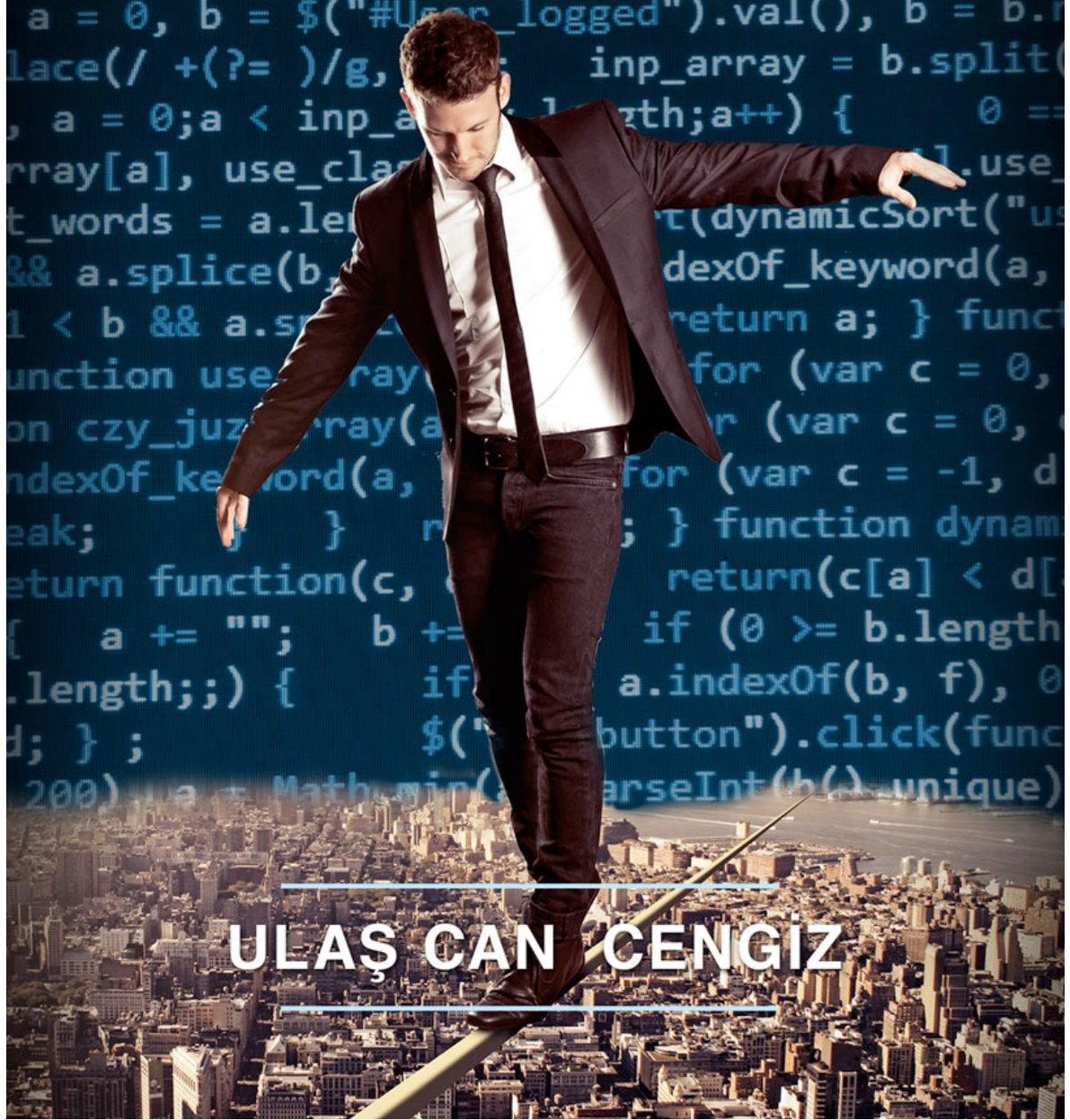
DENKLEM

Girişimciler için Yazılım Dünyasında
Hayatta Kalma Rehberi



ULAŞ CAN CENGİZ

Kullanım Hakları

Karıma.

Yazar Hakkında

Merhaba, Ben Ulaş Can Cengiz. 80'lerin sonunda bir bahar günü doğdum. "Seri" denebilecek kategoride bir girişimci ve danışmanım. Blogları ve makaleleri saymazsak bu kitapla beraber "yazar" da oldum denebilir.

İş hayatımın büyük bir bölümünde yazılım üzerine çalıştım. Kendi şirketim olan ProGeek Software ile 2 yıl, kurumsal hayatta 2 yıl ve öncesinde freelance olarak 6 yıl kadar bir sektör deneyimim var (2015 ortası itibariyle).

Girişimler ve girişimciler hakkındaki derdim daha kurumsal hayata girmeden filizlenmeye başladı. Girişimcilerle yazılımcılar arasında durması gereken bir "teknik iletişim kadrosu" eksiğini gördükten sonra da bu eksiği kapatmaya karar verdim. Bu kitap bu sürecin ilk çıktılarından biri.

Şu sıralar (2015 başı ve sonrası) girişimcilere yazılımcılarla nasıl ilişki kurabileceklerine dair eğitimler vermekteyim.

Eğitimlere, bundan sonra gelecek yazılara ve yazıyor olduğum kitaplara **ulascengiz.com** adresinden ulaşabilmeniz mümkün.

Aynı zamanda Medium (**@ulsc**), Twitter (**@ulsc**), Facebook (**fb/ulascancengiz**) ve LinkedIn (**in/ulascengiz**) üzerinden de dilediğiniz zaman iletişim kurabiliriz.

Ayrıca kahve içmeyi severim. Olur da ihtiyaç olursa aklınızda bulunsun :)

Teşekkür

Kitabın yazılması sürecinde desteklerini esirgemeyen; başta karım Diyetisyen P. Seda Cengiz ve ailem olmak üzere, sevgili arkadaşlarım Ahmet Arslan ve Ahmet Burak Şimşek'e,

Abilerim ve mentorlarım oldukları için kendimi çok şanslı hissetmemi sağlayan Umur Özal, Buğra Pamuksüzer ve Yrd. Doç. Dr. Adem Yavaş'a

Sizlerin huzurunda tekrar tekrar teşekkürlerimi sunarım.

İyi ki varsınız.

Proje Başlangıcı

1

Girişimciler açısından kendini yazılım alanında geliştirmek biraz zor.

Zira yazılımın yanında finans, iş ve müşteri geliştirme, yatırım süreçleri, yasal süreçler vb. hakkında da bir hayli bilgi sahibi olma sorumlulukları var.

Yazılımcıya Nasıl Ulaşırım?

Yazılımcıya ulaşmak, günümüz girişimlerinin en büyük problemlerinden bir tanesi. Özellikle Silikon Vadisi gibi yazılımcıya en çok ihtiyaç duyulan yerlerde bu ihtiyaç artık bir açlık seviyesine gelmiş durumda.

Bunun, teknolojinin hızlı gelişiyor olması dışında bir sebebi daha var: Yazılımcıların artık seçme şansı var ve bu şansı zekice değerlendirebilecek bir ortama sahipler.

Bu değerlendirmelerin sonunda ortaya çıkan ise, pek de girişimlerden yana değil. En iyi yazılımcılar en iyi şirketlerde çalışmak istiyor. Paraya ihtiyacı olanlar orta düzey şirketlere razı oluyor, olmayanlar kendi şirketlerini kurma yoluna gidiyor. Bu denklem içerisinde bir girişim için gece gündüz çalışıp riskin büyük bir kısmını yüklenip üstüne getiriden küçük bir pay almak pek de yer bulamıyor ne yazık ki.

Teknoloji girişimlerinde yazılımcıların önemli olmasının en temel sebebi süreçlerin kümülatif gelişimi. Yani, bir girişimci kendi başına yazılım öğrenebiliyor bile olsa; işin içinde gelişmiş bir yazılımcı kadar kendini geliştirmesi hemen mümkün olmuyor. Bu da yazılımcının rolünü gittikçe kritik bir noktaya taşıyor.

Öte yandan yazılımcıların kendilerini geliştirmelerine fırsat verebilcek pek çok kanal var. Bu kanalların pek azı ülkemizde geçerliliğini sürdürmekte, lâkin öyle ya da böyle varlar.

Başta üniversiteler geliyor. Ülkemizde en fazla insan kaynağı ve her ne kadar içi boş olabilse de birkaç sene bir şeyler öğrenmiş olabilecek insanlar çıkartma şansına üniversiteler sahip.

Bununla birlikte, kendi kendini geliştirme yoluna gitmiş yazılımcılar var. Şahsen ben de yazılım geliştirdiğim süreçler içerisinde bu grup dahilindeyim.

Girişimciler açısından kendini yazılım alanında geliştirmekse biraz daha zor. Zira yazılımın yanında finans, iş ve müşteri

geliştirme, yatırım süreçleri, yasal süreçler vb. hakkında da bir hayli bilgi sahibi olma sorumlulukları var.

Bu yüzden elimden geldiğince bir girişim için doğru yazılımcı profilini tanımlayıp, girişimcilerin onlara nasıl ulaşabileceği üzerine fikirlerimi paylaşacağım.

Bir yazılımcıyı diğer yazılımcılardan daha iyi yapan özelliklere göz atacak olursak, ben başa insan ilişkilerini koyardım. Zira insan ilişkilerinde iyi olabilmek bir girişimin kadroları açısından kritik bir noktada. İletişimi kaybettikten sonra kodun hızlı yazılıyor olmasının çok da bir önemi kalmayabiliyor. Bununla birlikte, yazılan kodun temiz olması da gayet önemli bir etken. Temiz koddan kastımı ilerleyen bölümlerde anlatacağım; lâkin kabaca, insanların da anlayabileceği kod bloklarını tanımlamak için kullandığımı belirteyim.

İyi bir yazılımcı bulabilmek için ilk bakılması gereken yer arkadaş çevresi. Eğer yeni bir girişimci değilseniz, mutlaka çevrenizde yazılım işiyle uğraşmış birileri veya onları tanıyan başka birileri olacaktır. Onlarla başlamanızı öneririm. Zira süreç her zaman güzel işlemeyebilir, sürecin bütün adımlarını mümkün olduğunca güvenli bir bölgede geçirmek en iyi yöntem olacaktır.

Bununla beraber LinkedIn ve Facebook gibi sosyal mecralarda yazılımcı toplulukları bulunmakta. Bu topluluklar içerisinde dili düzgün ayarlamazsanız hüsrana uğramanız mümkün; ilerleyen bölümlerde dile dair de birkaç şey anlatacağım.

Kritik noktalardan bir tanesi, yazılımcı ararken bulabileceğiniz herhangi biriyle aranıza bir mesafe koymuyor olmanız. Özellikle yeni başlayan bir girişimde girişimci, ekibiyle arasına ne kadar mesafe koyarsa, gerçeklikten o kadar uzaklaşıyor ve o kadar antipatik hale geliyor. Tecrübeyle sabit.

Motivasyon yine buradaki en önemli başlıklardan bir tanesi. Girişimcinin, ekibini motive etmek gibi bir zorunluluğu olduğu aşikar. Bu durum, yazılımcılarla çalışırken kendini daha net belli ediyor olacaktır.

Yazılımcı bulmanızda size yardım etmek isteyenler olacaktır. İnsan kaynakları ajansları, profesyonel yazılımcı bulma ekipleri ve benim de arasında bulunduğum topluluk olan girişim danışmanları bu kategoridedir.

Bu süreçte size birilerinin yardım etmesini istiyorsanız, öncelikle ne istediğinizi bilmeniz gerekiyor. Örneğin kalifiye bir yazılımcı bulabilmek için insan kaynakları ajansları teknik yeterlilik sahibi olmayabilir. Danışmanlar için de bu geçerli. Veya profesyonel yazılımcı bulma ekipleri çok para isteyebilir.

Bütün bunlar göz önünde bulundurularak, yazılımcı bulma sürecinde dikkat edilmesi gereken püf noktaları belirtmek isterim.

Yazılımcı Bulmanın Püf Noktaları

Yazılımcı ararken dikkat edilmesi gereken temel nokta, tekrar söylüyorum, kullanılan dildir. Eleman arıyor olmak yerine, projenizi birlikte geliştireceğiniz birilerini arıyor olmanız, hem sizi hem de aradığınız yazılımcıları rahatlatacaktır.

Her şeyi yapabilecek yazılımcı, aynı zamanda hiçbir şey yapamayacak yazılımcı olduğu gibi, her şeyi yapabileceğini söyleyen girişimci de hiçbir şeyi yapamayacak biri olarak algılanabilir.

Bu şekilde anlaşmanın önüne geçebilmek için yazılımcı ararken kullanılan dile dikkat edilmelidir. Aynı dikkat durumu yazılımcıyla yapılan görüşmeler süresince de yürütülürse, daha gelişkin bir ilişki oluşturulabilir.

Yazılımcı ararken girişimcinin kendisinde dikkat etmesi gereken noktalar bunlar. Yazılımcıda dikkat edilmesi gereken noktalar ise;

1- Teknik Yeterlilik

En sık sorun yaratan kavramların başında gelir. Yazılımcının teknik yeterliliğini ölçebilecek bir kaynağa ihtiyaç olması girişimler için büyük bir sıkıntı olabilir. Bu konuda tavsiyem, ya profesyonel bir yardım alınması, ya da yazılımcının geçmiş referanslarının iyice incelenmesi olacaktır.

2- Öğrenme İsteği

Öğrenme isteği, bir girişimin en önemli başlıklarından bir tanesi. Zira gittikçe hızlanan bir şekilde gelişmekte olan teknoloji çevremizi sarmışken, olan bitenden bir şeyler anlayabiliyor olmak önemli.

3- Takım Çalışmasına Uygunluk

Başta bir takımınız olmayabilir, ama zamanla oluşacaktır. Takımın oluşma sürecinde yazılımcıların takım çalışmasına olan uyumluluğu önemli bir yer tutar. Yazılımcı en nihayetinde bilgisayarında kodlarla karşı karşıya kaldığı dönemler geçirecektir; lâkin onun dışındaki süreçleri de kavrayabiliyor olmak büyük bir artıdır.

4- Sorun Çözme Becerisi

Bir yazılımcıyı diğer yazılımcılardan ayıran özelliklerin başında gelir. Zeka ile ilgisi olduğu kadar tecrübe ile de ciddi şekilde

ilgilidir. Bir yazılımcı ne kadar sorunla karşılaşmış ve yoluna devam edebilmişse, o kadar sorunla karşılaşıp süreçlerin zararsız atlatılmasını sağlayabilir.

5- İnsan İlişkileri

İster istemez bulunması gereken özelliklerden bir tanesi. Lâkin iyi insan ilişkileri kötü bilgisayar ilişkilerini beraberinde getiriyor ve işin çıkmasının önüne engeller koyuyorsa; pek işe yarama-yabilir. İnsan ilişkisi ve yazılımcılık arasında denge kurabilen kadroların tercih edilmesi önemlidir.

6- Tembellik

İronik olabilir, ama işini yapıyor olduktan sonra tembellik bir yazılımcının en eğlenceli özelliklerinden bir tanesidir. Zira makineler tembelleri severler. Tembeller her zaman yapılacak işin daha kolay bir yolunu bulur ve oradan ilerlerler; bu durum da genelde süreçlerin hızlanmasını sağlar.

7- Strese Dayanıklılık

Girişim süreçleri stresli süreçlerdir. Strese olabildiğince dayanıklı olmak gerekliliği bundan ötürüdür. Lâkin strese dayanıklılık derken, herkesin kafayı yediği bir dönemde hiçbir şeyi sallamamak anlaşılmamalıdır. Süreçlere dahil olmak, stresi yaşamak ve ardından o strese dayanabilmek daha önemli olacaktır.

8- Planlama Kapasitesi

Planlama yapabilen yazılımcı bulmak görece zor olabilir. Bunun temel sebebi, yazılımın öğrenildiği hiçbir yerde planlamanın öneminden yeterince bahsedilmiyor olmasıdır. Halbuki planlama kapasitesi, yazılımcının kendi üzerindeki yükü azaltacağı gibi, girişim için de bir yük haline gelmesinin net olarak önüne geçecektir.

9- Girişimci Kafası

Elzem olmamakla birlikte, girişimci kafası bir yazılımcı için her zaman artı puandır. Sadece önünüze gelen üzerinden düşünmeyip, süreçleri ileriye taşıyabilecek birine ihtiyaç duyarsanız, onun temel özelliklerinden bir tanesi girişimci kafasına sahip olmak olacaktır.

Bütün bu özelliklerin mükemmel birlikteliğini bir kişide bulabileceğinizi sanmıyorum. Bununla birlikte, aradığınız özelliklere dikkat etmeniz gerekecek. Zira hiçbir süreç tek başına bir kişinin itelemesiyle ilerlemez, yanınızda sizinle beraber elini taşın altına koyabilecek kalifiye arkadaşlara sahip olmanız gerekir.

Diyerek, yazılımcı bulma sürecini geçiyorum. Diyelim ki bir şekilde yazılımcı bulduk. Sırada onunla nasıl aynı dili konuşacağımız, nasıl anlaşabileceğimiz ve çalışma koşullarını nasıl en iyi şekilde oluşturabileceğimiz var.

Fikir Analizi

Yazılımcı bulma sürecini atlattıktan sonra, karşımıza derdimizi yazılımcıya anlatabilme süreci çıkıyor. Bu süreç de bir önceki gibi biraz can yakıcı olacaktır, ama bir girişimci için can yakıcı süreçlerin problem arz ettiğini sanmıyorum.

Her girişim fikrinin bir analize ihtiyacı vardır. Sadece yazılımcı için değil. Herhangi birine kendini anlatabilmek için, önce kendini anlayabilmek gerekir. Haliyle, fikrin analizini doğru yapabilmek önemli bir adım.

Fikrin ne olduğunu sadece sizin biliyor olmanız yetmeyeceği gibi, çoğunlukla bu sizin de tam ve net olarak bilmiyor olduğunuz anlamına gelecektir. Yani fikrin analizinin temel konusu, kendimizden başlayarak dünyaya fikrin tam ve net olarak ne olduğunu aktarabilmek.

Girişimin girişimcisi dahil bütün çalışanlarının aynı noktada duruyor olması, süreçleri hızlandıracaktır. Bu bilgi akışını sağlayabilmenin yöntemleri ise değişkendir. Örneğin birileri bütün ayrıntıları yazılı olarak bir düzen içerisinde tutmayı yeğlerken, bir diğeri parça parça notlar halinde bulundurmayı seçebilir. Bunu söylememin sebebi, fikrin analizi yapılırken kullanılan yöntemlerin büyük ölçüde kişiden kişiye değişebilmesi.

Kişiden kişiye değişebilen bir sürecin herkese aynı şekilde anlatılabilmesi ise, bu bölümün temel konusu. Yani size tam bir yöntem tanımlamak yerine, kilit noktaları iletmek isterim.

1- Ayrıntıları Netleştirmek

Başlık zaten kendini yeterince anlatıyor, ama tekrar tekrar üzerinden geçmekte bir sakınca görmüyorum: Fikrinize dair bütün ayrıntıları o veya bu şekilde netleştirip yazıya dökmeniz gerekiyor. İş planı yaparken zaten bunun bir kısmını yapmış olacaksınız, ama çoğunluk durumda yaptığınız yeterli olmayacaktır.

Bütün ayrıntılardan kastım, işleyiş mekanizmalarından, gelir modelleri kurgusundan, kullanıcıların nasıl üyeliklerini silebileceklerine kadar geniş bir kapsamda yazılı bir doküman oluşturmak. Kapsam ne kadar genişler ve derinleşirse, sonraki süreçler o kadar acısız geçer.

2- Personaları Netleştirmek

Persona, burada kısaca girişiminizin olası kullanıcıları demek. Bütün kullanıcılarınızın, gündelik yaşamlarında nasıl bir hayat sürdüklerine kadar, kedi besliyor olmaktan, e-ticaret sitelerinde en az bir kere alışveriş yapmış piyano çalan meksikalı cüceler olduklarına kadar olabildiğince fazla bilgi içeren personalar oluşturabilirseniz, bir sonraki adımda çok rahatlarsınız.

3- Kullanıcıların Dilinden Konuşmak

Bunu yapmak, girişiminizin ayrıntılarını yazıya aktarmaktan daha zor. Bu yüzden yardım almanız gerekebilir. Bununla birlikte aile ve arkadaş çevrenizde kullanıcı personalarınıza en çok uyanlara teker teker gidip onlarla sohbet ederseniz, onların gündelik dillerine kolaylıkla vakıf olabilirsiniz. Kullanıcılarınızın dilinden konuşabilmek, girişiminizin bütün süreçlerinde olduğu gibi yazılım süreçlerinde de büyük kolaylık sağlayacaktır.

4- Kullanıcı Hikayeleri Oluşturmak

Kullanıcı hikayeleri, yani “epic”ler, kullanıcılarınızın dilinden yazılan cümleciklerdir. Bütün süreçleri tek tek tarif etmenize yarayan bu cümlecikler, tam olarak şu şekilde organize olurlar: “Premium kullanıcı olarak, ödemelerimi ödemelerim bölümünde liste halinde görüntüleyebilmek istiyorum”.

Bu şekilde bir cümle kurmamızın birkaç sebebi var. “Premium kullanıcı olarak” demek, premium kullanıcı personamız içerisindeki kullanıcılardan biriyim demek. “Ödemelerimi” demek, ben ödeme yapıyorum demek. “Ödemelerim bölümü” demek, ödemelerimle ilgili ayrı bir bölüm var demek. “Liste halinde” demek, birileri bu ödemelerimi listeleyip bana sunuyor demek. “Görüntüleyebilmek istiyorum” demek, karşıma bir bilgi ekranı geliyor demek.

Dikkat edin, görüntülemekle tıklamak farklı şeyler. Burada tıklamak istiyorum yazmamasının sebebi, onun ayrı bir kullanıcı hikayesi olması. Bunun da temel sebebi, bir ekran veya ekran parçası içerisinde tıklanabilecek birden fazla yer olabilmesi ve herbirinin farklı işlevler yürütebilmesinin mümkün olması.

Bu başlıkları yeterince dikkatli bir şekilde inceleyip önermelerini gerçekleştirmek, fikir analizinden sonraki adım olan iş analizi sürecinde daha rahat olmanızı sağlayacaktır.

İş Analizi

İş analizi, işin ihtiyaçlarının ve gerekliliklerinin tam çözümlemesine denir. Fikir analiziyle birlikte yapılabileceği gibi, ben ayırmayı tercih ederim. Bunun temel sebebi, kesişen noktalarına rağmen farklı disiplinler olmaları ve işe farklı açılardan yaklaşıyor olmaları.

Girişimciler, kendi girişimlerinin iş analizini tek başlarına yapabilirler. Lâkin bu konuda tek başına çalışmayı önermem, zira birden fazla açıdan konulara yaklaşmak iş analizi sürecinde kritik noktalardan bir tanesi.

İş analizinin de birden fazla yöntemi olduğu gibi, burada kritik noktalarını derlemek istedim:

1- İşin Bütününü Tanımlamak

İşin en önemli ve en az üzerinde durulan kısmı. En az üzerinde durulan, zira her girişimci fikrinin sınırlarının net olarak belli olduğu yanılsamasına düşme potansiyeli taşıyor. Bu yanılsama bu başlığı es geçmeye sebep olabiliyor, bu yüzden dikkat etmek gerekiyor.

En önemli kısmı olmasının sebebi ise şu: Bütünü tanımlayabilmek, misyon cümlesi yazmaya benzer. Dışarıdan bakıldığında bir tek cümlecik gibi görünür, ama işin başından sonuna kadar varolma sebebi gibi değerli bir bilgi içerir. Gerçekten üzerine düşünülmüş bir misyon cümleniz varsa, işin bütününü o misyonun sınırları olarak tanımlamanızda bir sakınca yoktur.

2- Bütünü En Küçük Parçalarına Ayırmak

Burası biraz daha zor. En küçük parçalar demiş olmam sizi aldatmasın, kendi kendine işlevli en küçük parçaları kastediyorum. Yazılımlardaki fonksiyonlar gibi, istenildiği kadar iş yapabilen, ama bir tek iş yapabilecek kadar küçük olması tercih edilen parçalar.

Bu parçaların önemine bir sonraki başlıkta, yazılım analizinde, değineceğim. Burada kısaca bir örnek vereyim: Bütün olarak bir ödeme sistemi yapıyorsanız, en küçük işlevli parçalardan bir

tanesi kullanıcıların ödemelerinin alınmasından sonra devreye giren ödeme bölüştürme sistemi olabilir.

3- Genel Piyasa Çerçevesi Oluşturmak

Bunun yazılım süreçleriyle ne alakası var diye aklınızdan geçiyor olması normal. Bir de şu açıdan bakalım: Ortada sizin yapıyor olduğunuz işi yapan başka firmalar varsa, onların nasıl yapıyor olduğu üzerinden yazılım süreçlerinizi organize etmek size zaman ve para kazandırabilir.

Yani piyasa çerçevesinden kastım rakiplerinizin ne kadar yatırım alabildiği gibi metriklerle değil, altyapılarını nasıl oluşturuyor ve sürdürüyor olduklarına dair metriklerle oluşturulan bir çerçeve.

Bu tür bilgilere ulaşmak çok zor değil. Yapmanız gereken, geçen bölümde üzerinde durduğumuz gibi, kendinizi rakiplerinizin bir kullanıcısı gibi görüp olabilecek en detaylı şekilde çalışma sistemlerini incelemek.

4- Olası Sorunları Ortaya Çıkartmak

Şahsen en sevdiğim bölümlerden bir tanesi. Bir girişimci olarak, hayat boyu işler iyi gitse bile sürekli artan bir sorun skalasıyla baş etmek durumunda kalacaksınız. Bu yüzden bu skalaya daha işe girişmeden vakıf olabilmek, en azından olurmuş gibi yapabilmek, sizi içinde bulunduğunuz süreçte ciddi şekilde ileriye taşıyacaktır.

Buraya kadarki adımlarımızda tanımladığımız süreç içerisinde, pek çok olası mantık hatasıyla karşılaşmanız mümkün. Bunlardan korkmamak lazım, zira bir girişimin yaşam döngüsü içerisinde bu tip şeylerin ortaya çıkması gayet normal. Kritik olan bunları not almak ve mümkün olduğunca üstünde durup çözüme ulaştırmaya çalışmak.

Çözüme ulaştırmaya çalışmanın temel sebebi, özellikle yazılım süreci işin içine girdikten sonra beklenmedik sorunlarla karşılaşıldığında zaman ve para kaybının had safhada olabilmesi. Yazılım sürecine geçilmeden sorunların mümkün olduğunca çözüme kavuşturulması bu yüzden önemli.

5- Süreci Yazılım Analizine Uygun Hale Getirmek

Girişimci açısından, yazılım süreci bir vadede devredilmek durumunda olan bir süreç. Bu yüzden, yazılımcıya fikri tam ve net olarak anlatabilmek çok önemli. Yazılımcıya fikir ve iş analiziyle birlikte gittiğinizde sizi anlayacaktır. Lâkin yazılımın ortaya çıkma sürecini hızlandırabilmek için daha fazlasına ihtiyacınız var. Yazılım analizine geçmeden önce, sürecin yazılım analizine hazır hale getirilmesi gerekiyor.

Süreci yazılım analizine hazır hale getirmek demek, aslında yazılı hale getirmek demek. Bütünü, parçalarını, piyasa durumlarını ve karşılaşılan sorunları (ve olası çözümlerini) tek bir doküman altında toplamak, bunun en iyi yöntemi. Devam edelim.

Yazılım Analizi

Fikir analizini ve iş analizini tamamladıktan sonra, önümüzdeki süreçte yazılımcı(lar) ile aramızdaki ilişkiyi en rahat şekilde sürdürebilmemizi sağlayacak dokümana geldi sıra.

Yazılım analizini kısaca tanımlarsak, fikrin ve iş analizinin yazılımcı tarafından üzerine çok düşünülmeden koda dönüştürülebilmesine olanak tanıyan dokümandır diyebiliriz. İşlevi de tıpkı bahsettiğimiz gibi, yazılımcının yazması gereken kodları yazdığı süreçte ona arkadaşlık etmek olarak tanımlanabilir.

Bir girişimcinin profesyonel bir destek almadan tek başına yazması zor olan bir dokümandan bahsediyoruz. Özellikle teknik bir arkaplanı olmayan girişimciler için işin nasılına dair derinlikli bir araştırma yapmak faydalı olacaktır.

Yazılım analisti, iş analisti gibi iş tanımlarının olduğu bir teknoloji çağındayız ve bütün süreçleri tek başımıza veya küçük bir ekiple geliştirmeye çalışıyoruz. Zor, biliyorum, lâkin süreçlere hakim olmak açısından önemli.

Dokümanın tam ve net içeriğinden ziyade, kesinlikle içermesini önerdiğim kritik özelliklerinden bahsedeceğim:

1- İş Süreçlerinin Tam Aktarımı

İş analizinde tanımlanan süreçlerden farklı olarak, kullanıcıların herbir adımda alacağı aksiyonların ve göreceği bilgilerin tam ve net olarak açıklandığından emin olmak gerekiyor.

Örneğin bir ekranı oluşturmaya çalışan yazılımcının o ekran içerisinde hangi bilgilerin kime ve hangi sırayla görüntüleneceği konusunda tam ve net bir bilgiye ihtiyacı var. Bununla birlikte o ekranda nerelerin tıklanabilir olduğu, kimin nereye tıkladığında nereye gidebileceğini açıklamak da yazılım süreçlerini ciddi oranda hızlandırabilecek bir adım.

Yapmaya çalıştığımız şey, yazılım süreçlerini hızlandırmak olduğu kadar, yazılımcı ile ilişkimizi de sağlam temellere

oturtmaya çalışmak. Bu yüzden, her iki tarafın da aklında olabilecek en az miktarda soru kalması önemli.

2- İş Akışlarının Oluşturulması

İş Akışı, iş süreçlerini yazılım analizine aktarma işleminin şematik olarak ifade edilmesi diye özetlenebilir. Proje fonlama sitelerindeki videolarda konuşmacıların arkasındaki beyaz tahtalardaki elipsler, dikdörtgenler ve oklar görüyoruz ya, işte onlar iş akışları.

Algoritma olarak da tanımlanabilmesi mümkün, ama algoritma çok daha spesifik bir terim, o yüzden bu tanım içerisine dahil etmiyorum.

Şematik olarak aktarma işlemi üzerinde çalışılarak geliştirilebilecek bir özellik. Mükemmel çemberi çizebilmek için değil, bütün süreçleri şemaya dahil ettiğinden emin olabilmek için.

3- Objektif Yaklaşım Geliştirmek

Analistlerin yaşam döngüleri boyunca karşılaştıkları en büyük sorunlardan bir tanesi de, analizini yapmak durumunda oldukları sorun ve süreçlerin, subjektif bir bakış açısıyla tanımlanmış olması.

Örneğin, “kullanıcı kolayca giriş yapabilsin” demenin yazılım analizi sürecinde hiçbir kıymeti yok dersem abartmış olmam. Kolay, bir tanımlama değildir, bir yorumdur. Yazılım analizi için geliştirilmesi gereken yaklaşım, yorumlar üzerine kurulu subjektif yaklaşımdan ziyade tanımlamalar üzerine kurulu objektif yaklaşımdır.

4- Gerekliliklerin Tam Saptanması

Kitap boyunca bunu birkaç kere daha tekrar edeceğim, lâkin gerekli olduğuna inanıyorum: Analizi yapılan uygulama, sorun ve süreçlerin içeriyor olduğu bütün teknik ayrıntılar üzerine düşünülmüş olması gerekiyor. İş başlamadan bunu yapmak her zaman mümkün olmuyor, bunun farkındayım, ama yapılması gereken buna olabildiğince yaklaşabilmeye çalışmak.

Yazılımcıyla birlikte geliştirilebilecek bir doküman olduğu kadar, önceki dokümanlar gibi yazılım analizini de gerçek kullanıcılardan alınan geribildirimler eşliğinde geliştirmek ciddi bir kolaylık sağlayacaktır. Bunun temel sebebi, ister girişimci, ister kullanıcı, ister yazılımcı olsun, hepimizin gözünden bir şeyleri kaçırma olasılığının olması. Üstüne üstlük bir de yoğun bir şekilde bir fikri gerçeğe dönüştürme isteği ile yanıp tutuştuğumuz sıralarda, hata yapabilme potansiyelimizin sınırlarını zorlaması söz konusu.

Yapılması gereken, sürecin analizini ve yönetimini üstlenirken, sürekli bir öğrenme süreci içerisinde olduğumuz gerçeğini unutmamak. Onu başardıktan sonra yazılım analizinin sorun yaratacağını sanmıyorum.

Yazılımcının Yeterliliği Nasıl Ölçülür?

Yazılım analizimizi de oluşturduktan sonra, sıra geldi elimizdeki kaynakların yeterlilikleri hakkında net fikirler edinmeye.

Bu süreç, teknik bir arkaplanı olanlar için bile zor. Bu yüzden teknik bir arkaplanı olmayan girişimcileri göz önünde bulundurarak teknik değerlendirme yapmak üzerinde duracağım.

Elinizdeki yazılım analizi, yazılımcının tam olarak ne yapmasını istediğinizi ona anlatmaya yarayacaktır, lâkin onun söz konusu dokümanda yazılı her şeyi gerçeğe dönüştürüp dönüştüremeyeceğini anlayabilmek için size birkaç ipucu sunmak isterim:

1- Objektif Değerlendirme

Yeğeninizin çok iyi kod yazdığını biliyorum ve takdir ediyorum, lâkin sürece objektif yaklaşmamız gerekiyor. Yeğeniniz bile olsa, yazılım ihtiyaçlarınızı tam ve net olarak karşılayabiliyor olduğundan emin olmalısınız.

Özellikle ülkemizde yapılan en büyük hatalardan birisi her konuda olduğu gibi bu konuda da subjektif yaklaşım geliştirmek. Dünyanın en büyük şirketlerini oluşturanların aynı zamanda iyi arkadaşlar olduğunu, birlikte güzel vakit geçirip muhabbet edip eğlenebildiklerini biliyoruz, tamam. Lâkin buradaki ince çizgi, bahsi geçen arkadaşlarınızın yapacağını söyledikleri her ne pahasına olursa olsun yapabileceklerinden emin olmanız.

Zira bir yazılımcıyla bir proje üzerine anlaştığınız durumda, olası sorunların sorumluluğunu bir şekilde paylaşabileceğinizi bilirsiniz. Arkadaşlarınız ve yakın çevreniz söz konusu olduğunda ise bu süreç ilişkilerin kopmasına kadar gidebilir. Bu yüzden her iş gibi bunu da gündelik hayatımızda geliştirdiğimiz ilişkilerden ayrı bir yere koymak ve onun üzerinden objektif olarak ilerlemek faydalı olacaktır.

2- Üretkenlik

Üretkenlik çok saçma bir konu. Birinin üretkenliğini net olarak anlayabilmek için yeterli veriye onu tanımadan sahip olmanız

hemen hemen imkansız. Lâkin burada kullanabileceğimiz kimi yöntemler var.

Örneğin, geçmişi ister uzun ister kısa olsun, bir yazılımcının birim yazılım sürecine düşen proje sayısı güzel bir metrik olabilir. Ortaya çıkmış ve sonlandırılmış ürünlerden bahsediyorum. Yani bir yazılımcı 20 yıldır çalışmış ve herhangi bir projeyi sona kadar götürememişse, oradan hızlıca uzaklaşmanızı öneririm.

Bununla birlikte, yazılım alanında çok yeni olmasına rağmen kendi çabalarıyla 2-3 iş çıkartabilmiş genç bir arkadaş ciddi şekilde işinizi ileriye taşıyabilir. Şahsen bu konumda tanıdığım çok insan oldu, hiç yanlışlarını görmedim.

3- Bağlılık

Hiç yanlışlarını görmedim derken, burada inceden bir övünme de söz konusu sanıyorum. Bunun sebebi de, yanlışların genelde karşılıklı yapılıyor olması ve ortadaki yanlışsız süreçlerin de genelde karşılıklı gelişiyor olması.

Yazılımcıların uzun vadeli süreçlerde sizinle birlikte yürüyebilmesini istiyorsanız, en az yazılımcı kadar karşınızdakine o güveni sağlamakla yükümlü olduğunuzu unutmayın.

Şahsen tanıdığım onlarca yazılımcı arasında, siz gerçekleri şeffaf biçimde ilettiğinizde arkanızdan iş yürütmeye çalışanına çok az rastladım. Haliyle şöyle söyleyebiliriz: Şeffaflık bağlılığı getirir.

4- Kalite Saplantısı

Güzel bir özellik, zira dünya çapında teknoloji ilerledikçe yapılan işlerin kalitesi de artıyor. Bu artan kalite ortamında kalitesiz bir ürünle ortaya çıkmak sizi komik duruma düşürecektir.

Ürünün kalitesini sadece dış görüntüsü belirlemez, hatta çoğunlukla kaliteyi belirleyen ürünün iç yapılanmasıdır. Bu yüzden bir yazılımcıyı iyi yapan önemli özelliklerden biri de kalite takıntısı olarak ortaya çıkar.

5- Öğrenme Yetileri

Bu sadece yazılımcılar için değil, herkes için önemli. Birinin öğrenme yetilerini değerlendirmek tecrübe ile geliştirilebilecek bir yetenek ve belirli bir formülü yok. Bu noktada kalbinizin sesini dinleyebilirsiniz, lâkin yine de biraz araştırmanın bir zararı olmaz.

Bütün bu süreçleri tek başınıza gayet güzel atlatabiliyor olabilirsiniz, lâkin yine de güvendiğiniz bir kaynaktan fikir almak önemli. Güveninizin kaynağı yapılan işler ve alınabilen sorumluluklar olduktan sonra, başkalarına teknik değerlendirme anlamında güvenmenizde bir sakınca yok.

Yazılımcılarla Anlaşma Teknikleri

Yazılımcılarla anlaşmak, yazılımcı olmayanlara dünyanın en zor işi gibi gelebilir. Aslında yazılımcı olanlara da öyle gelir, yani bunda bir problem yok. Yıllar içerisinde yazılım süreçlerinin iç dinamiklerini anlamanın tek başına yeterli olmadığını defalarca görme şansı yakaladım. Süreç içerisinde geliştirdiğim yöntemlerden derlediğim adımları paylaşmak isterim:

1- Güven Temeli

Bütün ilişkilerde olduğu gibi, yazılımcılarla olan ilişkilerde de en temel nokta güven. Güveni oluşturmak da hayatın diğer alanlarında olduğu gibi insan ilişkileriyle ilgili. Lâkin yazılımcılar söz konusu olduğunda ek bir nokta daha var: Yazılımcı perspektifinden bakıldığında, işin temel taşlarını yerine oturtmak olarak tanımlanabilecek bir işlem söz konusu.

Bu işlem sırasında yazılımcının güvenini sağlayabilmek için en az onun kadar çalışıyor olduğunuzdan emin olmak faydalı olacaktır, zira yazılımcılar çok çalışmak durumunda olduklarının farkındadırlar ve bu durum benzer bir karşılık bulamazsa güven kaybına yol açabilir.

Güven sürecinin tek belirleyeninin para olmadığını da burada belirteyim. Süreci doğru kurgulamanın temel noktası her zaman taşın altına elimizi ne kadar koyuyor ve bunu ne kadar şeffaf bir şekilde gerçekleştirebiliyor olduğumuz.

2- Egoyu Dizginlemek

Teknolojinin ciddi bir hızla geliştiği günümüzde, yazılımcılar kendilerini dünyanın patronu olarak görürler. Aslına bakarsanız, bu bakış açısı bir noktasıyla doğrudur. Teknolojiye yön verebilme becerisi, pek çok alanda istenildiği gibi ilerleyebilme özgürlüğünü de beraberinde getirir.

Lâkin burada dizginlemek üzerinden bahsettiğim konu, yazılımcının egosu değil, girişimcinin egosu. Sürecin devindireni teknoloji olduğundan, yazılımcının egosunun temelleri sağlam. Girişimcinin kendi içerisinde orman kanunları işleyen bir piyasada kendisini savunmak için geliştirdiği egosu ise, görece daha flu bir noktada. Ortaya öyle ya da böyle çıkmak durumunda olan egoyu dizginleyebilmek, özellikle yazılımcı ilişkilerinde kilit bir noktada.

Daha somut konuşacak olursak, oluşturulan ürünün oluşturulma sürecinde “sahip” olmanın, “patron” olmanın ve “girişimci” olmanın verdiği egolardan mümkün mertebe sıyrılıp, işin en alt kademesinde nasıl yürütüldüğünü anlayıp, süreçlere o noktadan bir bakış açısı geliştirmek gerekiyor.

3- Sözlü Anlaşma

Belirli bir ilişki oluşturduktan sonra, yazılı anlaşmalara geçilmesi gerektiği zaten aşikar. Lâkin yazılı anlaşmaların öncesinde, iki taraf için de zarar içermeyecek bir “deneme süreci” tanımlamak ve bu süreç içerisinde karşılıklı verilen sözler üzerinden ilerlemek faydalı olacaktır.

Bu durum, faydasını özellikle süreç ilerlemeye başladığında gösterir. Yapılacak yazılı anlaşmalar genelde yasal olarak “üstesinden gelinebilecek” açıklarla dolu olacaktır. Bu açıklar üzerinde durmanın iki taraf için de rahatlatıcı olacağı süreçler gelecektir. Bu tarz süreçlerden sıyrılıp işin gerçekten ilerlemesini sağlayabilmek için, tarafların karşılıklı olarak birbirlerine verdikleri sözleri tam ve net bir şekilde tutabiliyor olmaları önemlidir.

Küçük sözlerle sürece başlamayı öneririm. Konsepti kanıtlayan minimum bir ürün olabilir, karşılığında verilebilecek bir para olabilir, sunulacak şeylerin değişmesi mümkün. Değişmeyen kural ise, sözlü anlaşmaya ne kadar uyulursa, yazılı olanlara da ancak ve ancak o kadar uyulabileceğidir.

4- Ortaklık Yaklaşımı

Hisse paylaşımı gibi konular üzerinde duracağım, lâkin şu anda bahsettiğim ortaklığın şirket ortaklığıyla doğrudan bir alakası yok. Gelişmekte olan bir sürecin sorumluluğunu yüklenmekteki ortaklıktan bahsediyorum. Tamamen psikolojik bir süreç, herhangi bir şekilde yazılması hatta konuşulması bile gerekmiyor.

Sorumluluğun ortak paylaşılıyor olduğunun anlaşılması, orta-uzun vadede oluşabilecek motivasyon sorunlarıyla baş edebilmekte ciddi fayda sağlayacaktır.

5- Küçük Adımlar ve Deneme Süreçleri

Sözlü anlaşma kısmında bahsettim, biraz daha açayım: Ortada herhangi bir süreç tanımı olmasa bile birlikte hareket etmeye başlayabilmek önemli bir özellik olacaktır. Bunun için ihtiyacınız olan temel şey, fikrinizi minimal düzeyde gerçekleştirmiş gibi

yapabilecek herhangi basit bir ürün olabilir. Bununla beraber, bu deneme süreci içerisinde aktif olarak bulunarak da sorumluluk paylaşımına dair ipuçları vermeniz mümkün.

Her ne kadar bu ipuçları işinize yarasa da, bir girişimci olarak sürekli geliştirmeniz gereken en önemli özelliğiniz insan ilişkileri. Yazılımcılarla ilişkiler de bunun önemli bir parçası, hatta kimi zaman kendinizi sınama tahtası bile olabilir.

Programlama Dili Seçmenin Püf Noktaları

Biraz daha eğlenceli bir konuyla devam edelim. Eğer teknolojiye herhangi bir şekilde dokunan bir girişimseniz, yazılım ile içli dışlı olacaksınız. Eğer yazılım ile içli dışlı olacaksanız da, oluşturmanız gereken sistemler olacak. Bu sistemleri oluştururken makinelere laf anlatmanız gerekecek. İşte programlama dilleri o makinelere o lafları anlatmak için kullanılan araçlar.

Her yazılımcının üzerinde çalıştığı bir veya birden fazla dil vardır. Herkesten hepsini bilmesini bekleyemeyeceğiniz gibi, herkesten hepsine dair bir fikri olmasını beklemenizin bir sakıncası yok.

Öyle ya da böyle, programlama dili seçmek durumunda kalacaksınız. Seçerken üzerinde durmanız gereken birkaç noktaya değinelim:

1- “Hepsini Döven” Tek Bir Dil Var Mı?

Kısa cevap: Hayır.

Uzun cevap: Evet.

İroni yapmıyorum, durum gerçekten böyle. Eğer kısa yoldan bir programlama dilini seçip onunla ilerleyip bütün alanlarda olası her türlü başlıkta diğer dillerden üstün gelmesini istiyorsanız, her platforma yazabileceğiniz, her türlü sorununuza çözüm olabilecek ve herkes tarafından yazılabilecek bir dil olmadığını söylemek isterim.

Bununla beraber, iş yapacağınız platformları, müşterilere ulaşacağınız kanalları, elinizdeki kaynakları, fikir danışabileceğiniz insanları net olarak belirlemişseniz, sizin koşullarınıza uygun ve diğerleri arasından net olarak sıyrılan bir veya birkaç programlama dili mutlaka vardır.

2- Dillerin Karakteristik Özellikleri

Her programlama dilinin kendine has özellikleri vardır. Birden fazla olmalarının temel sebebi ise, odaklandıkları alanların birbirinden farklı olmasıdır.

Örneğin Ruby dilinin oluşturulma amacını “geliştiriciler için güzel kod tabanları oluşturmakta kolaylık sağlamak” olarak tanımlamak mümkün. Bununla birlikte, C dilinin oluşturulma amacı, “bu makinelere bir şeyler yaptırmamız lazım, bunu da insanların anlayabileceği şekilde yapmamız lazım” diye açıklanabilir.

Karakteristik özellikler hakkında fikir sahibi olmak, programlama dili seçerken ciddi kolaylık sağlayacaktır. En sık kullanılan dillerin karakteristik özelliklerini bir sonraki bölümde inceleyeceğiz.

3- Müşteriye Ulaşma Kanalları

Biraz önce bahsettiğim gibi, müşterilere ulaşma kanallarını doğru saptayabilmek, programlama dili seçmenizi ciddi şekilde kolaylaştırır. Örneğin Apple cihazlar üzerinde bir uygulama geliştirecekseniz, Cobol dilini seçmeniz tamamen işlevsiz olacaktır.

Bununla beraber, her ne kadar platforma özel olmasa da, birden fazla kanalı destekleme imkanı sunan programlama dillerinin varlığını belirtip içinizi bir nebze de olsa rahatlatmak isterim.

4- Yazılımcı Sayısı

Bir dilin ne kadar yazılımcısı varsa, kaynak sıkıntısı çekmeniz o kadar zor olacaktır. Lâkin bununla beraber, yazılımcı sayısı çok olan bir dil, geliştirme konusunda sıkıntı çekmeyeceğiniz anlamına gelmeyebilir. Burada dikkat edilmesi gereken, o dili kullanabilen yazılımcı sayısıyla birlikte, o dilde sizin ihtiyaçlarınızı karşılayabilecek denli derinleşebilmiş yazılımcı sayısının çok olmasıdır.

Tersi durumda ise, yani yazılımcı sayısı az olan ama çok yeni ve çok güçlü olan bir dil kullandığınız durumda, kaynak sıkıntısı çekeceğiniz hemen hemen garantilidir. Örnek olarak Türkiye’de Python programlama dili kullanarak yazılımlarını geliştirmeye çalışan ve sonuç olarak hüsrana uğrayan pek çok girişimle karşılaştığımı söyleyebilirim.

Önümüzdeki bölümlerde sıkça üzerinde duracağımız sürdürülebilirlik başlığı da, yazılımcı sayısıyla dolaylı olarak ilişkilendirilebilir. Yani süreç içerisinde koşullar değişiyor olsa da geliştirme süreçlerinizin tam hızıyla devam edebilmesini istiyorsanız, kaynakları güvence altında bulunan bir programlama dili ile işe girişmenizi öneririm.

Bütün bunlarla beraber, programlama dili seçmek üzerinde doktora yapmanız gerekmiyor. Temel bilgilere sahip olduktan sonra, zaten süreç içerisinde konuya görece daha hakim kaynaklara başvurmanız gerekiyor. Bütün mesele, kaynakları planlarken vermiş olduğunuz son kararın sizin yararınıza olduğundan emin olabilmek.

Hangi Programlama Dili?

Programlama dili seçerken yüksek ihtimalle elinizdeki yazılımcı kaynağının biliyor olduğu diller arasından seçim yapmak durumunda kalacaksınız. Yine de eğer bir gün yazılımcı bolluğu yaşarsanız ve dilediğiniz dili seçme özgürlüğüyle karşı karşıya kalırsanız yardımınıza koşması için küçük bir liste hazırladım.

Herbir programlama diline dair ayrıntılı bilgi verecek olsam kitabı sadece bir tanesine ayırmak bile yetmeyecektir, bu yüzden kısa kısa bilgiler geçeceğim:

1- Java

Dünyanın en popüler dillerinden bir tanesi. Oluşturulması 1995'e kadar dayanıyor. Şu anda arkasında dünyanın en büyük şirketlerinden Oracle var. İlk çıkış sebebi, bir kod tabanıyla heryerde çalışan sistemler yazmak olarak özetlenebilir.

Android işletim sisteminin anladığı dil Java. Scala gibi kimi programlama dilleri de yine Java altyapısı üzerinden geliştirilmiş.

Türkiye'de okullarda öğretiliyor, bu yüzden kaynak bulmakta zorluk çekilmeyecektir. Dünyada da popülerliği had safhada olduğundan herhangi bir ülkeden tonla yazılımcı bulmanız mümkün olan dillerden bir tanesi.

Bağımlıları için hepsini döven tek dil olarak gösterildiği doğrudur.

2- JavaScript

Java ile uzaktan yakından alakası olmadığını söyleyerek başlayalım. Ruby ve PHP gibi, alakası bile olmayan iki tane dil Java ve JavaScript. İsmini neden öyle koymuşlar derseniz, zamanında Netscape'deki pazarlamacı arkadaşlar yayılması için Java ile isim benzerliği yaratmanın mantıklı olabileceğini düşünmüşler, öyle de kalmış.

Arkasında pek çok aktör var, en büyük iki tanesi Mozilla (bu Firefox tarayıcısını yapan şirket) ve Google.

Şu sıralar JavaScript kullanılarak iOS ve Android uygulamaları, web uygulamaları, masaüstü uygulamaları, IoT uygulamaları, kısacası aklınıza ne gelirse geliştirmek mümkün.

Mobil cihazlarda uçan kaçan arayüzler aramıyorsanız, multi-platform bir çözüm için ciddi şekilde uygundur.

3- .NET

Bu aslında bir programlama dilinden çok bir yapının adı. Arkasında zamanla ASP, C#, VB, C++ da kullanılmış olmakla beraber, en sık karşılaşacağımız C#.NET olacaktır. Zira o da Türkiye'de okullarda okutuluyor.

En çok yazılımcı kaynağı bulabileceğiniz alanlardan biri olmakla beraber, arkasında Microsoft gibi bir yapı olduğundan girişimler tarafından çok da ön planda tercih edilen diller arasında değil. Bunun kişiden kişiye değişebilecek sebepleri var tabii, üstesinden gelmek de pek tabii mümkün.

4- Python

İsminin yılandan gelmediğini belirteyim. Kurucusunun en sevdiği komedi şovu Monty Python'dan geliyor Python ismi. Açık kaynak kodlu bir dil olması ve Google'ın altyapı sistemlerinin bir kısmının üzerinde çalışması tercih edilme sebebi olabilir.

Lâkin Türkiye'de Python geliştirici sayısı diğerleriyle karşılaştırıldığında yok denecek kadar az. Başka şansınız kalmayana kadar alternatifleri değerlendirmenizi önermek durumundayım.

5- Ruby

Son zamanların en çok ortalarda dolaşan dillerinden bir tanesi. Girişimler tarafından çoğunlukla Ruby on Rails isimli altyapı ile birlikte kullanılıyor ve Github, Airbnb, Groupon gibi sitelerin altyapı sistemlerini koşturmasıyla ünlü.

Geliştirilme sebebini "yazılımcılara güzel bir dil ile kolaylık sağlamak" olarak açıklayabiliriz. Türkiye'de de gittikçe yaygınlaşan bir şekilde kullanılmakla beraber, silikon vadisi gibi bölgelerde ciddi şekilde popüler durumda.

6- PHP

Yine görece eski bir dil olan PHP, Microsoft güdümlü .NET ile çatışmasıyla dikkat çekiyor. O kadar büyük bir topluluğu var ki, dili geliştirmek için yapılan değişikliklerin sıraya konulması gerekiyor.

Facebook, Wordpress, Drupal, Flickr gibi yapıların arkasında PHP olduğunu belirtirken; popülerliğini uzun yıllardır

kaybetmeyen, herhangi bir ülkede tonla geliştiricisi bulunabilecek ama uzman bulmanın bir o kadar zor olduğunu da es geçmeyelim.

7- Swift

Kitabın yazıldığı tarihte sadece iOS uygulamaları geliştirmek için kullanılan, Apple tarafından üretilmiş bir programlama dili. Ama aldığımız haberler, kısa zaman içerisinde açık kaynak kodlu olacağı yönünde. Bu da diğer alanlarda da Swift görmeye başlayabileceğimiz anlamına geliyor.

Sonuç olarak hangisini seçeyim diye sorarsanız, durumunuzu tam olarak incelemeden size net bir cevap verememem. Bu alanda size de tavsiyem, mümkün olduğunca güvenilir bir kaynaktan bir yönlendirme almanız.

Proje Üzerinden Anlaşmak

Programlama dilini de seçtiğimize göre, sıra yazılımcıyla önümüzdeki süreç üzerine daha net konuşmaya geldi. İlerleyebilmek için bir anlaşma yapmamız gerekiyor.

Bir projenin yazılımını yaptırmak için önümüzde çok da fazla anlaşma seçeneği olduğunu söyleyemem, ama birinden biri girişiminiz içerisinde oluşturmak isteyeceğiniz yapıya uyacaktır. Anlaşma yöntemlerine gelirsek:

1- Proje Başına Ücret

En az açıklamaya ihtiyaç duyacak başlık sanıyorum bu. Projenin yazılım analizinin gücü, söz konusu yöntem çatısı altında girişimcinin elindeki en büyük silah olacaktır. Tam anlamıyla netleşmiş bir projeye fiyat verilmesi ve üzerinden pazarlık edilmesi çok kolaydır. Bu yüzden, eğer bu yöntemi seçecekseniz iyi hazırlandığınızdan emin olmanızı öneririm.

Proje başına ücretin bir diğer özelliği; girişimci için de, yazılımcı için de en yüksek riski taşıyan anlaşma türü olmasıdır. Girişimci açısından net bir para çıkışı karşılığında sonucu bilinmeyen bir alana girmek, yazılımcı açısından da net olarak geleceği belli olmayan bir para karşılığında belli bir zaman harcamak risk başlıkları olarak sıralanabilir.

Sürece ne kadar hakim olurlarsa olsunlar, yazılım ekipleri genel olarak proje tarihlerini tutturmak için kimi özelliklerden feragat etme eğilimindedirler. Bu feragat yazılımcılar için de geçerlidir. Buradaki en büyük sorun, bu feragat etme sürecinin işin başından belli olmamasıdır.

Bu yüzden, eğer bu yöntem denenecekse, işin başından sonuna kadar en net analizin yapıldığından emin olunması gerekir. Sonrasında oluşması muhtemel sorunların önüne geçebilmek bu sayede mümkün olabilir.

2- Düzenli Sabit Ücret

Eleman çalıştırmanın diğer adı. Burada altına gireceğiniz sorumlulukları farkediyor olmak önemli. Zira sigorta, yol, yemek gibi ek masrafların yanı sıra, işin zamana yayılması ve olası sıkıntıların sorumluluğunun doğrudan patrona yüklenmesi gibi sorunlarla karşılaşabileceğinizi belirteyim.

Girişimci için yüksek maliyet, yazılımcı için yüksek stres içerir. Sigorta, vergi, yol, yemek, tatil haricinde; iyi bir yazılımcının iyi çalışması için gereken ücret girişim bütçe standartlarının üstündedir. Orta ve alt seviyedeki yazılımcıları verimli çalıştırabilmek için ise, ileri seviye bir yazılımcıya veya bir profesyonel danışmana ihtiyaç vardır

3- Şirket Hisse Paylaşımı

En çok kullanılan yöntem bu sanıyorum. Yazılım sürecinin de iş geliştirme süreci gibi bütünün bir parçası olarak tanımlanmasıyla beraber, benim de en sevdiğim ve önümüzdeki bölümde detaylı olarak açıklayacağım bir yöntem.

Girişimlerin çoğunluğu yaşam döngülerine bir şirketleşme olmadan başlarlar. Bu durumda, yazılımcıyla yapılacak anlaşma, yazılımcıyı risk altına sokacaktır. Bu yüzden böyle bir durumda, girişimci, kendi üzerine düşen görevleri net olarak tanımlamalı ve yazılımcıya iletmelidir.

Bununla birlikte, vesting olarak adlandırılan bir hisse paylaştırma yöntemi daha vardır ki, bu Türkiye'de yasal temelleri tam oturmadığından çok tercih edilmemektedir. Yazılımcının bir süre boyunca çalışması karşılığında belli bir hisse payı alması durumuna vesting diyebiliriz.

Genelde yatırım almış girişimlerde, net ücretle birlikte kullanılır. Yazılımcıları işin bir parçası haline getirmek esastır. Hisse değerleri çok düşük olduğundan yazılımcılar arasında çok tercih edilmez.

4- Kar Ortaklığı

Bu maddeler içerisinde en az kullanılan diyebilirim. Zira ortada kar etmekle uzaktan yakından alakası olmayan yüzlerce girişimin bulunduğu bir ekosistemden bahsettiğimiz için, kar ortaklığını gerçeğe dönüştürmenin de çok bir mantıklı görünmediği aşikar.

Bir diğer yanıyla, hisse paylaşımı gibi düşünmek de mümkün. Girişimcinin ince düşünmesi gerekirken, yazılımcının kafası daha rahat olan bir hisse paylaşımı olarak açıklanabilir.

Yatırım süreçleri işin içine girdiğinde paylaşımın nasıl devam edeceğine dair net bir yol haritası çıkartmak gerekliliğine dikkat etmek lazım, ki aynı şey hisse paylaşımında da birebir geçerli.

Yüzde Paylaşımı

Yüzde meselesi üzerine hararetli tartışmalar döndüğüne şahit oluyorum. Herkesin konuya dair farklı bir perspektif sunabiliyor olması güzel, ben de aynısını iki farklı perspektifi birleştirerek yapacağım.

Bir girişimin optimum düzeyde gelişim gösterebilmesi için içinde bulunan aktörlerin bulundukları yerden memnun olmaları gerekiyor. Hem girişimci hem de yazılımcı perspektifinden süreçlerin nasıl işlediğine dair birkaç fikir sunayım:

Öncelikle halihazırda para kazanıp kazanmıyor olduğunuz konusu var. Eğer halihazırda para kazanabilir -veya para kaynağı yaratabilir- durumdaysanız onu bir kaynak olarak değerlendirmeniz ve yazılımcıyla doğrudan para üzerinden anlaşmanız mümkün.

Bu durumda etik olarak da en mantıklı olan, sürecin başından beri sizinle yan yana yürüyenleri unutmayıp onlara vesting yoluyla bir takım hisse payları vermeniz olacaktır.

Yazılımcı açısından bakıldığında, sizin geliştirmeyi planladığınız girişimin ilk bakışta diğer girişimlerden bir farkı olmayacaktır. Onda da kod yazmak gerekecek, onda da emek ve zaman ortaya koymak beklenecektir. Bu yüzden ilk olarak karşılaşacağınız yaklaşım, yazılan kodun para karşılığının oluşturulması üzerinedir.

Piyasa araştırması yapmak bir yazılımcının iş tanımında bulunmamaktadır. Bu yüzden konu fiyat konuşmaya geldiğinde muğlak bir tabloyla karşılaşmanız mümkün. Herhangi bir piyasa düzlemi içerisinde bulunma zorunluluğu olmadığından, yapılan işin tek bir merkezden değerlendirilip fiyatlandırılması en sık karşılaşılan durum olacaktır.

Böyle bir durumla karşı karşıya kalındığında, önceden belirli bir piyasa araştırması yapmanın faydasını ciddi şekilde görebilirsiniz.

Yazılım ve yazılımcı piyasasını doğru şekilde araştırabilirseniz, yeterince fazla kaynaktan fiyat teklifleri alabilirseniz, kafanızda ortalama bir fiyat yargısı oluşacaktır.

Bu yargı doğrultusunda, elinizdeki kaynaklarla da doğru orantılı bir değerlendirme yapıp yazılımcıya sunarsanız, komik rakamlardan bahsetmiyorsanız anlaşmakta bir sorun yaşamazsınız.

Bununla beraber, kritik başlıklardan bir tanesi nasıl bir hisse paylaşımı yapmanız gerektiği. Öncelikle bunun tamamen kişisel bir yaklaşım olacağını söyleyeyim.

Bu alanda size net olarak doğru bir yöntem gösteremem, zira böyle bir yöntem bulunmuyor.

Süreci en hızlı ve etik değerler içerisinde -ve kimsenin motivasyonuna zarar vermeden- sonuçlandırabilmek için, en başta bahsettiğim yazılım analizine olan ihtiyacınız burada da belirgin şekilde hissedilecektir.

Yeterli analiz yaptığınız durumda, yazılımcının harcaması gereken emek miktarını ölçebilir, bunu girişimin diğer başlıklarıyla karşılaştırabilirsiniz.

Projeniz henüz fikir aşamasındaysa, süreci tanımlayacak en makul yaklaşım eş paylaşıma gitmek olacaktır. Projenin başından itibaren parçası olmaya gönüllü aktör sayısı kadar eş hisse bölümü yapılması burada bana kalırsa en doğru yaklaşımdır.

Projenin fikir aşaması tamamlanmışsa, analizleri yapılmış ve geriye bir tek kodun yazılması kalmışsa, bu paylaşım eş paylaşım normalinin yarısı şeklinde organize edilebilir. Örneğin bir girişimci ve bir yazılımcı varsa, yazılımcının payı %25 seviyesinde şekillendirilebilir.

Projenin baştan mı tekrar mı yazılıyor olduğu yazılımcıyla alakalı değil, girişimin kendisiyle alakalı bir durumdur. Bu sebebten yazılımcıdan yazılı kodu devralıp yeniden yazması beklenecekse, sıfırdan yazılıyor olması durumuyla aynı koşullar geçerli olacaktır.

Bütün girişimlerin veya şirketlerin toplamda %100 hissesi olduğundan, emek yoğunluğu hisse değerini büyütmez. Emek yoğunluğunun hisse değerini değiştirebilmesinin tek yolu, proje içerisindeki aktörlerin bu yoğunluğu paylaşma derecelerinin değişmesidir.

Yani bütün işi yazılımcı yapacak ve başka herhangi bir aktöre herhangi bir iş düşmeyecekse, girişimcilere önerim yazılım öğrenmeleri olacaktır.

Bu alanda sağlıklı bir yaklaşım geliştirmek her zaman tek başına mümkün olmaz. Bu yüzden konuya hakim üçüncü bir gözden yardım almak ciddi şekilde işinizi kolaylaştırabilir.

Yasal Anlaşma

Yazılımcılarla yapılacak yasal anlaşmanın bütün şartlarına hakim olmaya çalışmanız yerine, bu alanda aktif olarak çalışan bir avukata başvurmanızı öneririm. Bununla birlikte, uzun süredir yaptığım tonla anlaşmadan öğrendiklerimi de sizinle paylaşmak istiyorum:

1- Cezalı Anlaşmalar

Şirketler arasında yapılan anlaşmaların büyük çoğunluğu, işin beklendiği gibi gelişmediği durumda tarafların zararı karşılayabilmek adına bir ceza çekmesi -genelde para ödemesi- üzerine bir madde bulundurur.

Bunun temel sebebi, şirketlerin kendi yapılanmalarını bozmadan aldıkları hizmetleri sorunsuzca ve sorumluluk altında kalmadan alabilmek istemeleridir.

Bir girişim söz konusu olduğunda ise burada bahsi geçen garanti türünün bir mantığı olmayacaktır. Girişimin doğası gereği süreç içerisinde denemeler, yanılmalar, sinir bozuklukları ve ilişki yıpranmaları olacaktır. Bunu sözleşme içerisinde cezai bir duruma bağlamak sadece yazılımcıların değil diğer aktörlerin de anlaşmaktan kaçınmasını beraberinde getirir.

Bu yüzden önerim anlaşmalar içerisine cezai bir madde koymak yerine, ekibe girecek yazılımcı ile ikili ilişkilerinizi optimum düzeyde tutmak olacaktır. Bu durumda bir kısmı zaten yaşanmak durumunda olan olası sorunları yaşarken bile motivasyonunuz düşmeyebilir.

2- Yazılım Şirketleri Yasal Süreçleri Yavaşlatır

Pek çok kurumsal müşteri ve girişimle çalışmış bir yazılım şirketinin başında olduğum dönemin bana getirdiği dürüst yargı,

yazılım şirketlerinin yasal süreçleri yavaşlatıyor olduğudur. Zira yazılım şirketleri de diğer kurumsal firmalar gibi sözleşmelerinin her bir kelimesine dikkat etmek, her bir durumu içerdiğinden emin olmak gibi sorunlarla uğraşmak durumundadırlar.

Bu durumun bir girişim açısından acı verici olduğunun da farkındayım, bir süre bu acıyı veren tarafta bulunduğumu da itiraf etmek isterim. Öğrenme süreçleri böyle süreçler sonuçta.

3- E-Posta Yoluyla Yapılan Anlaşmanın Geçerliliği Vardır

Her şeyi ıslak imzaya bağlamak gerekmez. Şahsen yaptığım tonla gizlilik anlaşmasının birkaçı dışında hiç ıslak imzaya ihtiyaç duymadığımı söyleyebilirim. Anlaşmanın iki tarafında da defalarca kez bulunduğumdan sürecin nasıl işlediğini aşağı yukarı biliyorum ve tekrar ediyorum: Bir girişim için yapılacak anlaşmalarda esas olan anlaşmanın değil, karşılıklı ilişkilerin yaptırım gücüdür.

Yazılımcıyla anlaşmanız sürecinde aranızda geliştirdiğiniz ilişkiden daha değerli herhangi bir yazılı anlaşma olmayacaktır. Özellikle girişim gibi yeni iş hayatına başlayan yapılarda bu anlaşmaların geçerliliğini sağlayan taraflar arasındaki samimiyetten başkası değildir.

Bununla beraber, yazılı anlaşmaların karşılıklı niyet belirtisi şeklinde e-posta yoluyla yapılmasında hiçbir sakınca yoktur. Olabilecek en kötü durumla karşılaşmamanızı umarım, ama karşılaşırsanız bu e-postaların geçerliliği olduğunu da bilmenizde fayda var.

4- Sözleşmeden Beklentiler

Söz konusu sözleşmenin iki tarafı var. Girişimci bu sözleşmenin bir tarafı olduğu kadar, yazılımcı da bir diğer tarafı. Aynı yolda yürüyecek, aynı yöne bakacak insanlar arasında yapılacak sözleşmelerin her iki tarafı da gözetiyor olması gerekir.

Sizi dış dünyadan korumak üzerinden geliştirilmesi yerine, sözleşmelerinizin, sizin dış dünyayla bağınızı en güçlü hale nasıl getireceği üzerine odaklanması gerekir.

Bu bağlamda, sözleşmelerin tarafların birbirinden ne beklediğini net olarak içermesi gerekir. Ne kadar emek, ne kadar süre, ne kadar işgücü, ne kadar gizlilik, ne kadar para, bütün bunların net olarak belirtildiği anlaşmalar karşılıklı soru işaretlerine mahal bırakmayacak şekilde anlaşılabilir olurlar.

Girişimler söz konusu olduğunda, taraflar, anlaşma yapmak isteyip istemediklerine bireysel ilişkileri üzerinden karar vereceklerdir. Lâkin bununla beraber, bu bireysel ilişkileri herhangi bir başlığıyla zedelemeyecek, üstelik ileri taşıyacak bir yazılı anlaşmayla da pekiştirildiği takdirde, sürecin beklendiği gibi işlemesi tarafların alanlarındaki yetkinliğine emanet edilmiş olabilecektir.

Fikrin Çalınması

Geldik fasülyenin faydalarına. Son süreçte en çok tartışılan konulardan bir tanesi projelerin gizliliği.

Üstelik bir de yazılımcı gibi projeyi alıp gerçeğe dönüştürme potansiyeli olan birinden projeyi "koruma"nın nasıl mümkün olacağı yakın zamanda forumları ve sosyal medya gruplarını işgal etmeye devam ediyor.

Kesin bir yöntemi olmamakla birlikte, kişiden kişiye değişebilecek kimi adımlarla bu sorun başlığının ortadan kalkabileceğine inanıyorum. Bugüne kadar herhangi bir fikrimi çaldırmadığım için şanslı olduğumu düşünmem bir yana, eğer düzgün değerlendirilirse birkaç anahtar nokta göze çarpıyor. O noktaları sizinle paylaşmak istedim.

1- Güven - Sözleşme Dengesi

En iyi yöntemin her zaman güven temelli bir ilişki kurmak olduğunu birkaç kez söyledim, tekrar söyleyeyim. Elbette bunun anlamı doğaya pozitif enerji salıp bütün insanlığın sizin fikrinizin size ait olduğunu anlamasını beklemek değil. Bu noktada mümkün olduğunca güven ilişkisi sağlayacağınız kişilere dikkat etmeniz gerekecektir.

Sözleşmenin karşılıklı güven ilişkisiyle doğrudan bir ilgisi yoktur. Gizlilik sözleşmesinin temel amacı, tarafların yapıyor oldukları şeyleri yaparken kullandıkları bilgi ve araçları paylaşmamaları olmalıdır. Bu durum zaten fikrin çalınmasının önüne geçer.

Kritik nokta, süreci fikirle sınırlı tutmamaktır. Tamamen size ait olduğunu düşündüğünüz bir fikrin birkaç hafta içerisinde bir benzerinin çıkması ve bunun sizin herhangi bir bağlantınızla ilgili olmaması, günümüz koşullarında pek tabii mümkündür.

Herkes sizin gibi düşünebilir, konulara sizin açınızdan yaklaşabilir, sizin gibi proje geliştirebilir veya o projeleri gerçeğe

dönüştürmekle uğraşabilir. Öncelikle bunu kabul etmek lazım. Bunu kabul ettikten sonra, gizlilik sözleşmesinin mantığı daha da net oturacaktır. Kullanılan araçlar, süreçler ve süreç içerisinde üretilen fikirlerin gizliliğinin korunması, fikrin kendisinin gizli tutulmasını kapsıyor olsa da, onunla aynı şey değildir.

2- Göz Korkutma

Sözleşmelerin herhangi bir göz korkutma amacı taşıması, tarafların motivasyonunu ciddi şekilde düşürecektir. Kimse kendini tehdit altında hissetmeyi sevmez. Göz korkutma amacı taşıyan bütün sözleşmeler, karşılığında bir direnç görürler. Bu direncin zamana yayılması, sürecin işlemeye başlama zamanını öteler ve üretkenliğini azaltır.

Bu yüzden, sözleşmelerin aslında herkesçe malum olanı dünyaya ilan etme kağıtları olduğunu unutmamak gerekir. Özellikle yazılımcılarla ilişkilerde durum bu şekilde.

3- Fikrin Yasal Sahibi

Garip bir başlık, bir o kadar garip bir konu. Fikrin belli patent süreçlerinden geçmediği, belli yasalarca net bir şekilde koruma altına alınmadığı ve bu korumanın sınırları net bir şekilde çizilmediği sürece, o fikir yasal olarak size ait değildir. Yani bir fikrin yasal olarak size ait olduğunu, bir takım yasal süreçler haricinde dünyaya kanıtlamanın bir yöntemi yoktur.

Gizlilik sözleşmeleri bu yüzden kişiler -veya kurumlar- arasında yapılır. Aynı şekilde, tam da bu yüzden başka etkenler tarafından fikrinizin bambaşka noktalara çekildiğini görmeniz pek tabii mümkündür. Bu durum, birilerinin sizin fikrinizi açıklıyor veya satıyor olduğu anlamına gelmez. Birileri normal günlük hayatlarında bir an akıllarına gelmiş “sizin fikriniz”i gerçeğe dönüştürüyor olabilir.

Bu yüzden gizlilik süreçlerini fikri bir fanusa sokmak için değil, gerçekleştirme sürecini ilerletmek için kullanmak gerekir.

4- Piyasa Etkisi

Yazılımcılar ve teknoloji girişimcileri, aynı piyasanın aktörleridir. Bu durum, birbirleri üzerinde pozitif ve negatif etkide bulunabilecekleri anlamına gelir.

Gizlilik söz konusu olduğunda, hiçkimse piyasa içerisinde “ifşacı” veya “hırsız” olarak bilinmek istemez. Bu bilinirlik, otomatik olarak kişinin önündeki bütün iş ilişkilerini sıkıntıya sokacaktır. Bu yüzden içinde bulunduğunuz piyasada ne kadar sağlam bir yer edinirseniz, o kadar sağlam bir gizlilik süreci yaratabilirsiniz.

Bütün bunların ardından, fikrinizin çalınması değil de umursanmaması üzerine endişelenmenizin daha mantıklı olacağını düşündüğümü belirteyim.

Geliştirme Süreci

2

Yazılım, konuya görece uzak kimseler açısından "her şeyin yapılabileceği alan" olarak görünür.

Yazılım geliştirmeye yeni başlayanlar ise bunun tersini düşünürler.

Zaman Tahmini (Estimate)

Zaman tahmininde bulunmak, yazılım sürecine başlarkenki kritik noktalardan bir tanesidir. Bu kadar kritik olmakla beraber, genel itibariyle çok da iyi değerlendirilmez.

Kısaca tanımlamak gerekirse, yazılım açısından zaman tahmini, mümkün olduğunca gerçekçi bir şekilde eldeki işlerin ne kadar sürebileceğine dair destekli fikirler oluşturma süreci olarak tarif edilebilir.

Her zaman yazılımcının kendisi tarafından yapılmak durumunda olmasa da, genel olarak girişimler açısından durum bu şekildedir. Adımlarını ve adımların püf noktalarını paylaşmak isterim:

1- Gerekliliklerin Netleştirilmesi

Bütün zaman tahmini sürecinin ilk adımı, gerekliliklerin netleştirilmesi olmalıdır. Gerekliliklerin netleştirilmesinin en iyi yolu yazılım analizinden geçer. Yazılım analizi ne kadar doğru ve detaylı yapılmışsa, üzerinden zaman tahmini yapmak da bir o kadar kolay olacaktır.

Buradaki olası sorunlardan bir tanesi, yazılımcının o zamana kadar gerçekleştirmemiş olduğu bir işlem için zaman tahmini yapmakta zorlanacağı gerçeğidir. Yani, telefon ekranında harita gösterme yöntemlerini bilmeyen bir yazılımcı, harita göstermenin ne kadar süreceği hakkında destekli bir tahmin yapabilmek için daha çok veriye ihtiyaç duyar.

Buradaki veriyi sağlamak girişimcinin doğrudan görevi olmasa da, bir girişim içerisindeki her şey doğrudan veya dolaylı olarak girişimcinin görevi olduğundan, araştırma sürecine katılmak gerekebilir.

2- İşlemlerin Sıraya Sokulması

İşlemlerden kastım yazılım analizinde bahsettiğimiz en küçük parçalar. Analiz doğru yapıldığı takdirde, yapılacak işlemler hemen hemen belli ve kaynak da sınırlı olduğundan, işlemlerin bir sıraya sokulması ve tahmin yapılırken de bu sıraya uyulması gerekmektedir. Bu noktada işlemlerin ne kadar süreceği üzerine tahmin yürütmek gerekli değildir. Bu sürecin sebebi, işlemlerin

sıraya sokulup tahminlerin bütünlüklü olarak değerlendirilebilmesine olanak sağlamaktır.

3- Katılımcıların Belirlenmesi

Tahmin sürecinin katılımcıları önemlidir. Zaman tahminleri yapabilmek için her zaman tek bir yazılımcı yetmeyebilir. Bu tarz durumlarda girişimcinin de sürece dahil olması gerekir. Onun da yetmediği durumlarda projenin analizine ve zaman tahmini sürecine dışardan bakabilecek ve katkı sunabilecek profesyonel bir göze başvurulması faydalı olacaktır.

4- Tahminlerin Yapılması

Herbir küçük parça için ayrı ayrı zaman tahminleri yapılması ve bütünün zaman beklentileriyle uyumluluğunun kontrol edilmesi gerekmektedir. Bunu yapmanın en kolay yolu, küçük parçaları mümkün olduğunca detaylı bir hale getirmek olacaktır. Yeterince detay içeren parçalara zaman tahmini yapmak görece daha kolaydır.

5- Tampon Zamanlar

Tampon (buffer) zamanlar, süreç içerisinde karşılaşılabilecek olası sorunların alacağı zamanlara denir. Herbir işlem içerisinde bulunmak durumunda olmamakla birlikte, işlemin boyutu büyüdükçe tampon zaman gerekliliği artar.

Örnek olarak, telefon ekranında harita gösterme işi 2 saat sürecek diye öne sürülmüşse, işin ortalama 1.5 saat sürecek olması ve yarım saatin tampon zaman olması yüksek ihtimaldir.

Zaman tahminlerine göre hızlı ilerliyor olmak bir başarı değildir. Aksine, zaman tahminlerinin yanlış yapıldığını göstermektedir. Doğru yapılan zaman tahminleri, kendi tampon zamanlarını doğru tanımlayıp, işi zamana yaymayıp, bütün süreci ilerletmeye devam ederken işleri bir bir bitirmeye odaklanmış zaman tahminleridir ve genel olarak sürece tam otururlar.

Olası sorunlardan bir tanesi de, tahminlere göre yavaş ilerliyor olmaktır. Bunun tecrübeyle ilgisi olabileceği gibi, motivasyonla ilgisi olması da muhtemeldir. Böyle bir durumla karşılaşıldığında ekibi motive edecek dengelerin iyi kurulup işlediğinden tekrar tekrar emin olmak işe yarayabilir.

İnanmadığın bir şeyi yapmak zordur. Bu inanmadığın kişi olabilir, iş olabilir veya zaman olabilir. İnanılabilecek zaman tahminleri yapmanın, “bir hafta içerisinde 5 uygulama yapmamız lazım” ile farkı, ortaya bir sonuç çıkartabilmesi olacaktır.

Yazılımcıyla Aynı Dili Konuşmak

Yazılım, konuya görece uzak kimseler açısından "her şeyin yapılabileceği alan" olarak görünür. Yazılım geliştirmeye yeni başlayanlar bunun tersini düşünürler, onlara her şey imkansız gibi görünür. Belli bir tecrübe kazandıktan sonra, her şeyin mümkün olması üzerinden değil de, her şeyin arkaplanı nasıl acaba diye düşünerek ilerleme süreci başlar.

Deneyimli yazılımcılar için, ne yapılacağı değil, nasıl yapılacağı önemlidir. Bu bağlamda, yazılım "her şeyi yapabilir" olarak kabul edilse de, her şeyi yapabilmenin ciddi boyutlara ulaşabilen bedelleri olduğu da görülmüş olur.

Bu yüzden yazılımcınızla aynı dili konuşabilmenizin kilit noktası, ona her şeyi yapma sorumluluğunu yüklememekte yatar. İşine dair yapabileceğiniz olumlu veya olumsuz yorumların, deneyimli yazılımcılar haricindekileri genel itibariyle uzun vadede olumsuz etkilediğini söylemek isterim.

Bir girişim içerisinde bir yazılımcının pozisyonu, girişimcinin hayallerini gerçeğe dönüştürmek değildir. O, girişimcinin pozisyonudur.

Yazılımcının pozisyonu, kendisine tanımlanan işleri mümkün olan en hızlı ve güçlü yoldan, yine mümkünse en uygun bütçe sınırları içerisinde yapmaktır, onun dışında bir şey yüklemek onu yazılımcı kategorisinden çıkartır.

Burada bir yazılımcının girişimci olamayacağını söylemeye çalışmıyorum, aksine, eğer girişimci kimliği üzerinden düşünülecekse, kişinin süreç içerisinde yazılım yapıyor olması bir problem teşkil etmez. Burada kritik olan, kişinin belli roller içerisinde bulunduğu sırada, o rollerin net sınırları içerisinde kalmasının girişimin işine yarayacağıdır.

Kullanılabilecek dil açısından önemli noktalara gelince, yapılması gereken işe dair “kolay”, “şu kadarcık”, “bunda ne var” gibi sözler sarfetmenin yararınıza olacağını hiç sanmıyorum. Herhangi bir teknik altyapıdan geliyorsanız, bunları zaten sarfetmezsiniz, lâkin konuya teknik açıdan yaklaşmıyorsanız da ek bir dikkat göstermenizi öneririm.

Hiçkimse yaptığı işin küçümsenmesini istemez, yazılımcılar da istemez. Herhangi bir yazılım parçası, bir yazılımcı tarafından hızlı ve kolay bir şekilde üretilebilir, lâkin bunu size yazılımcının belirtebilmesi gerekir. Aksi durumda, yazılımcı tarafından yapılan işin olduğundan daha zor gösterilmesi sorunuyla karşılaşmak mümkündür.

Yazılımcılarla ortak dil tutturmak zordur. Kilit noktalarından bir tanesi de, insan ilişkilerinin her başlığında olduğu gibi, onunla aynı yola baş koyduğunuzu gösterebilmektir. Bir girişim içerisinde gerçekten bir yazılımcı kadar çalışabilirseniz, sizinle anlaşamayacak yazılımcı yoktur.

İyi anlaşabilmenin bir diğer görece az önemli kriteri de, gün içerisindeki zaman planlaması olacaktır. Kimi yazılımcılar gündüz, kimileri gece çalışmayı sever. Daha doğrusu sevmek bir yana, ancak o şekilde kendilerini var edip ortaya bir iş çıkarabildiklerini düşünürler.

Bu alanda tavsiye edebileceğim yaklaşım, yazılımcının takımın geri kalanına uyumunun bir şekilde sağlanması olacaktır. Bunun temel sebebi, takımın kendi içinde bir bütün olarak hareket etmesinin girişim için faydalı sonuçlar üreteceği gerçeği.

Sorumluluğun büyük ölçüde girişimci tarafından yükleniliyor olması, resmin yazılımcı tarafından da böyle göründüğünü her zaman göstermez. Özellikle çalışma saatleri arasında fark varsa, resmin bu şekilde görünmeyeceğinden emin olabilirsiniz. Bu noktaya getirebilmek için benim tavsiyem, sürecin nasıl gelişiyor olduğunu genel çerçevesiyle ve kimi zaman özele de inerek yazılımcınıza aktarmanız.

Bu sayede, gerçek anlamda bir ekip resmi yazılımcının da gözünde netleşecek ve bu durum ekibin ciddi biçimde ileri gitmesini sağlayacaktır.

Bir teknoloji girişiminde yazılımcının bulunduğu pozisyon kritiktir, lâkin bu durum, yazılımcı ne derse yapalım demek değildir. Yazılımcı da, girişimci gibi, bir ekibin parçası olduğu ve o ekiple beraber hareket ettiği sürece kendini ileri götürmeye devam edebilir. Bu yüzden, bu ekibin başında olan kişi veya kişilerin yazılımcıyla anlaşma süreçleri, diğer ekip arkadaşlarıyla anlaşma süreçlerinden çok farklı kurgulanmamalıdır.

Yazılımcı ile Birlikte Proje Yönetmek

Genel anlamda proje yönetiminden ilerleyen bölümlerde bahsedeceğim, burada bahsetmek istediğim, proje yönetim süreçlerinin bir yazılımcı ile nasıl paylaşılacağı. Kullanılan proje yönetim tekniği şu an için çok önemli değil, önemli olan sürecin dışardan bakıldığında nasıl kurgulanmış olduğu.

Yazılımcıyı "yönetilecek kişi" olmaktan ziyade "birlikte çalışılacak kişi" olarak görmeyi önereceğim. Yazılım süreçlerinin teknik olmayan bir girişimci tarafından yürütülebilmesi zaten pek mümkün görünmüyor, lâkin benzer herhangi bir durumda da yönetim sürecinin paylaşılmasının faydalı olacağını düşünüyorum.

Yazılım süreçlerini yazılımcıdan daha iyi kavramanız gerekmiyor, öncelikle onu netleştirelim. Bir girişimci olarak her şeye dair fikrinizin olması sizin için bir artı, lâkin bütün yazılım sürecine üstten bakmak sizi büyük ölçüde yavaşlatacağından, mümkün olduğunca yazılımcının kendi süreç yönetimlerini -hatta bazen sizinkini bile- yapmasına alan bırakmak güzel sonuçlar verebilir.

Girişimlerin geleneksel şirketlerden ayrıldığı noktaların başında, hızlı karar alıp hızlı uygulamaya geçebilmesi gelir. Bu hızın teknoloji tarafındaki açılımı kimi zaman, daha önce yapılmış ürün altyapısını çöpe atmak ve yerine başka bir yapı oluşturmak da olabilir.

Bu tarz durumlarda, kararların tepeden inme değil de, yazılımcının da işin içinde bulunduğu bir ekip tarafından veriliyor olması, yazılımcının süreç içerisindeki motivasyonu açısından oldukça önemlidir. Bu alanda işinize yarayabilecek birkaç önerim var:

1- Aynı Noktadan Baktığınızdan Emin Olun

Aynı noktadan bakmaktan kastım, sürecin nasıl gelişiyor olduğunun herkes tarafından net bir şekilde anlaşılır hale getirilmesi. Bu bazen ofiste görünür bir yere asılacak bir tablo ile olabilir, bazen çevrimiçi proje yönetim araçlarınızda görünür alanlara yerleştirilen çizelgelerle.

Yol haritanızın netleşmesi, ekipteki herkes için büyük önem taşır. Yazılımcı içinse, kendi yapması gereken proje ile kendi dışında gelişen başlıkların bütünlüğünü oluşturabilmek açısından ciddi kolaylık sağlayacaktır. Bu sayede, yazılımcı, süreç içerisinde geliştiriyor olduğu ürünün yaratabileceği sonuçlara dair kendini motive edecek bir alan yaratabilmiş olur.

2- İş Listesini Şeffaf Hale Getirin

Bir girişimde, herkesin birbirinin o anda ne yapıyor olduğunu bilmesine gerek yoktur, lâkin bunun kimi faydaları vardır. Örneğin, bir bütün olarak ekibin sürecin hangi bölümünde olduğunu net olarak ortaya koyabilmesi, yazılım gibi teknik ağırlıklı başlıkların bulundukları noktaları tam saptayabilmesi açısından kritiktir.

İş listesine dair efsanelerden bir tanesi, gizli tutulan işlerin aslında hiç yapılmıyor olduğudur. Bu kısmen doğru olabilir, zira bir girişimde bir işin gizliden yürütülüyor olması için gerçekten başka bir başlık altında düşünülüyor olması gerekir.

Şeffaflık hayatın her alanında olduğu gibi, bir girişimin hayat döngüsü içerisinde de kritiktir. Yazılımcınızdan da bunu beklediğinizi açıkça söylemenizi öneririm. Tam ve net olarak o gün nelerle uğraşıyor olduğunu biliyor olmanız, süreci daha iyi anlamanızı sağlayacaktır.

3- İşlerin Sahiplerini Tam Saptayın

Yazılım ve pazarlama arasında bir iş karışıklığı olmayacaktır, lâkin pazarlama ve satış alanları arasında bu muhtemeldir. Satış ve iş geliştirme alanlarında da pek çok başlıkta işin sahibini anlama sorunu ortaya çıkabilir.

Yazılım alanında her zaman olmasa bile, özellikle yazılım ekibi büyümeye başladıkça, atılması gereken her adımın kim veya kimler tarafından atılacağının tam ve net olarak saptanmış olması gerekir. Bu durum ciddi şekilde kafa rahatlatır, motive eder.

4- İşler Yapıldığında Herkesi Haberdar Edin

Özellikle yazılım tarafında bir işin parçalarının her birinin bitirilmesinin ufak çaplı bir kutlamayı hak ettiğini söylemek isterim. Yeni bir müşteri bağlamak gibi, yeni bir özelliğin geliştirilmesini tamamlamak da önemli bir iştir ve ekip içerisindeki herkesin durumdan haberdar olmaya hakkı vardır.

İşin biraz eğlencesi gibi görünüyor olsa da, proje yönetimi sürecinde herkesin aynı noktadan bakıyor olduğunu netleştirmeye ciddi şekilde yardımcı olur.

Kodları Nerede Tutmalıyım?

Ortak dilin tutturulmasının ardından, belirli bir yazılım sürecinin başladığını ve ortaya bir kodun çıktığını varsayalım. Elimizdeki sorunlardan bir tanesi, kodların tam olarak nerede barındırılacağı sorunu.

Bu soruya "yazılımcının bilgisayarı" veya "Dropbox" gibi bir cevap veriyorsanız, size "versiyon kontrol sistemi" denilen bir yapıdan bahsetmek isterim. Versiyon kontrol sistemleri, yazılan kodların (aslında sadece kodlar değil, ama konumuzun kapsamı bu) belirli bir düzende saklanmasını sağlayan yapılar.

Versiyon kontrolden kastım, bütün yapılan hareketlerin belirli bir düzende kaydedilip yine aynı düzen içerisinde istenildiği zaman geri alınabilmesini ve birden fazla yönde ilerleyebilmesini sağlamak. İlk bakışta biraz karmaşık gelebilir, ama üzerine düşününce böyle bir yapı oluşturmanın bir kod tabanı için ne kadar kritik olduğu anlaşılacaktır.

Kodların nerede tutulacağının önemi, sadece takip ediliyor olması değil elbette. Tek bir cihazda tutulan veriler, o cihazın yaşam döngüsüne bağlıdır ve onunla beraber her an yok olabilme riski taşırlar. Örneğin olası bir düşme, yanma, su kaçma durumunda süreç içerisinde geliştirilmiş tonla kodun yok yere tarihe karışması mümkündür.

Bu tip durumlarla karşılaşmamak için, kodun öncelikle birden fazla noktada tutulması gerekir. Bu durumda da, kodun her bir kopyasının ne güncel haliyle tutulması sorunu ortaya çıkar. Versiyon kontrol sistemleri bu soruna, eşleme adını verebileceğimiz bir yöntemle çözüm bulurlar. Yani, bir noktada yazılan bir kodun, bütün kopyalarını aynı noktaya getiren birkaç işlemle güvence altına alınmasından bahsediyorum.

Kodun tarihe karışması sorunu, karşılaşılabilecek sorunlardan bir tanesi. Bununla birlikte, süreç içerisinde yapılması muhtemel yanlışların geri çevirilmesi sorunu da mevcut. Yani, müşterilerinize istemediğiniz şeyler göstermeye başlayan sisteminizde, sorunun tam olarak ne zaman ortaya çıktığını

anladıktan sonra, o söz konusu günden sonra yazılan kodun tam olarak neyi değiştirdiğini görebilmek önemlidir.

Bu sayede versiyon kontrol sistemleri, sadece kodun tutulduğu yeri dağıtmakla kalmaz, aynı zamanda süreç içerisindeki geliştirmeleri de kayıt altına alarak yapılması muhtemel hatalardan da hızlıca dönülebilmesini sağlar.

Versiyon kontrol sistemlerinin temel işleyişini anlamak önemlidir. Bunun ilk sebebi, her şeyden önce, muhteşem sistemler olmaları ve onları öğrenmenin size hiçbir şey kaybettirmeyeceği. Bunun dışında, önümüzdeki bölümlerde göreceğimiz sürdürülebilir bir yazılım üretilmesi sürecinde, kimi kontrol mekanizmalarında işinize yarayacağını söyleyebilirim. Örneğin “commit” ismini verdiğimiz yapılan işin tanımını versiyon kontrolüne anlatma parçaları, sürdürülebilirlik süreçlerinizi iyi yönde etkileyebilir. Önümüzdeki bölümlerde detaylandıracağım.

Versiyon kontrol sistemlerinin en meşhuru, Git isminde bir yapı. Github.org adresinden en yaygın kullanım alanını görebilmeniz mümkün, ama Git başlı başına kendine özgü bir yapı. İnternet adresi git-scm.com olan Git, Linux işletim sistemini geliştiren Linus Torvalds tarafından geliştirilmiş ve tamamen açık kaynak kodlu olarak dünyanın kullanımına sunulmuş.

Bununla birlikte, TFS, SVN, CVS, Mercurial, Bazaar, Monotone gibi versiyon kontrol sistemleri de mevcut olmakla birlikte, burada hepsinin üstünde ayrıca durmayacağım.

Girişimler için açık ara en çok tercih edilen versiyon kontrol sistemi Git olduğundan, onun üzerinde ilerlemenin mantıklı olacağını düşünüyorum. Hem bir geliştirme yaparken, hem de yapılan geliştirmeleri denetlerken oldukça kolay bir mekanizma sunuyor.

Versiyon kontrol sistemleri herhangi bir grafik arayüzü içermeyen kod parçalarıdır. Lâkin her veri gibi, burada içerilen verinin de grafiksel anlatımını oluşturmak mümkündür. Github ve Bitbucket gibi yapılar bunu sizin için yaparlar.

Bir kere tam entegre olduktan sonra (ki bir yazılımcının iki dakikasını alır) bütün kod tabanınızı oradan takip edebilme şansına erişmiş olursunuz. Önümüzdeki bölümlerde, öncelikle bu kodun kime ait olması gerektiği, ardından da nasıl yönetilmesi ve incelenmesi gerektiğine dair fikirlerimi paylaşacağım.

Bütün versiyon kontrol sistemlerini öğrenmek yerine, sizin için en uygununun hangisi olacağına yazılımcınızla veya profesyonel biriyle birlikte karar verip onun üzerine kısa bir araştırma yapmanızın faydalı olacağını belirteyim.

Kodların Kullanım Hakları

Kodların kullanım hakları tanımı çoğunuza yabancı gelebilir, bana da öyle gelmesini isterdim açıkçası. Yıllarca yazılım işi yaptım, yazılım şirketi yönettim ama geçenlerde karşılaştığım şey daha her şeyi görmediğimi tekrar tekrar hatırlattı bana.

Bir yazılımcının anlaşmış olduğu bir ekibi uzun vadede kendine bağlayabilmek için kodları onlarla paylaşmıyor olması ve eğer istenilirse satabileceği üzerine düşünüyor olması, yıllar boyunca yazılan kodun hiç lafını yapmamış biri olarak garip hissetmeme sebep oldu.

Bunun üzerine özellikle böyle bir bölüm oluşturmaya karar verdim. Burada yazılanların şahsi düşüncelerimden öteye geçebilmesi için de, konu hakkında bulabildiğim makaleleri derleyerek daha objektif bir görüş oluşturdum.

Sonuç itibariyle, bir girişim içerisinde geliştirilen herhangi bir şeyin, bu kod olabilir, yöntem olabilir, müşteri listesi olabilir, hepsinin, tam anlamıyla girişimin kendisine ait olması gerekiyor. Bunun temel sebebi, girişimin hayat döngüsünü herhangi bir şekilde etkileyebilecek herhangi bir materyalin sağlığının şansa bırakılamayacağı gerçeği.

Herkes süreçleri kendine bağlamayı ve sadece kendi üzerinden yürütülebilecek başlıklar olması fikrini sever. Bunda yanlış bir şey de yok, herkesin hayatının bir alanında kendini vazgeçilmez hissetmesinin kimi faydaları vardır.

Gelgelelim, bir girişim söz konusu olduğunda, kişilerin vazgeçilmez olmasından daha önemli bir şey varsa, o da girişimin sürekliliğinin mutlaka sağlanıyor olması gerekliliğidir.

Yani girişimin yaşamını zora sokacak herhangi bir başlığın orada bulunuyor olması, ciddi biçimde bir tehdit içerir.

Kişiler arasındaki ilişkiler değişebilir. Bu yüzden en iyi arkadaşınız dahi olsa girişiminiz söz konusu olduğunda elindeki kartları tam ve açık olarak masaya koymasını istemek durumundasınız. Aksi durumda elindeki kartların süreç içerisinde size nasıl zararlar verebileceğini düşünmek durumunda kalabilirsiniz.

Elbette ters durum her zaman gerçekleşecek diye bir şey yok, lâkin garantiye almak gerekiyor. Olası herhangi bir anlaşmazlık durumunda dahi girişimin herhangi bir sorunla karşılaşmadan hayatına devam edebiliyor olması lazım.

Bu yüzden, yazılan kodun veya herhangi bir materyalin, girişime ait olduğunun net bir şekilde yazılı olarak belirtilmesi gerekiyor. Bu belirtildikten sonra, kimsenin herhangi bir durumda uykularının kaçmasının gerekeceğini sanmam.

Bununla birlikte, her şekilde bir yedeğin bulundurulmasını önereceğim. Bunu öneriyor olmamın sebebi, 1996'da Omega Engineering firmasından atılan bir yazılımcı. İsmi Timothy Lloyd ve 11 senedir çalıştığı firmadan atıldıktan sonra intikam almak istiyor. 3 hafta sonra, yazdığı 6 satırlık bir kodu çalıştırıyor ve bütün Omega sistemlerini çökertiyor. Verdiği toplam zarar 12 milyon dolar civarında oluyor.

Bunun bir girişimle ne alakası var diye düşünebilirsiniz. Şöyle açıklayayım. Eğer belli operasyonlarınızı belli kişilere emanet ediyorsanız, o kişilerin yapıyor oldukları şeylere dair net fikirler ve yedekler bulundurmanız gerekir.

Yedeklerden kastım, herhangi bir sıkıntı durumunda bir önceki yedek alınma zamanına hiç vakit kaybetmeden dönebilmenizi sağlayan "Farklı Kaydet" seçeneği. Tabii ki kodlar için biraz daha karmaşık süreçler, lâkin yedek almanın ne olduğunu biraz araştırdığınızda o kadar da zor gelmeyecektir. En azından yapmadığınızda olabilecekleri düşününce, zor olup olmamasını pek de önemseyeceğinizi sanmam.

Yazılım geliştirme süreçleri bu tip istisnalara sahne olsa da, durum genel olarak böyle değildir. Gerçekten sizinle birlikte bir şeyler yapmaya öyle ya da böyle ikna ettiğiniz bir yazılımcı, yazdığı kodları sizinle paylaşmakta herhangi bir sıkıntı çekmeyecektir.

Ayrıca, bu güzelliğinin karşılığında bir yazılımcı, girişimin diğer alanlarındaki başlıklarda da kendisine karşı şeffaf olunmasını hak eder. Periyodik olarak süreçten haberdar edilen aktörler, sürecin neresinde bulunduklarına dair oluşturdukları bilinçlerini yenileyebilirler. Bu da bir girişim için kritik başlıklardan biridir.

Proje Yönetim Teknikleri

Proje yönetim teknikleri çok çeşitli olabilir. Hemen her türlü soruna bir proje yönetimi tekniği icat edilmiş olduğundan aralarında kaybolmak işten bile olmayabilir. Bununla beraber, girişimler söz konusu olduğunda bu durum biraz daha farklıdır.

Büyük şirketlerde süreçlerin bürokratik işlemesini normal karşılayabiliriz, lâkin bunun Google gibi, Twitter gibi istisnaları da yok değil. Bu yüzden girişimimizin bütün süreçlerinde bize faydalı olabilecek teknikler arasından seçim yapmamız gerekiyor.

Girişimler genelde küçük ekiplerle işe başladıklarından, geleneksel proje yönetim tekniklerinden ziyade girişimin ihtiyacını karşılayabilecek teknikler üzerinde yoğunlaşılması akıllıca olacaktır.

Elbette, küçük ekiplerle çalışıyor olmak, bütün süreç boyunca küçük bir ekibiniz olacağı anlamına gelmez. Bu yüzden, gerek küçük gerek büyük ekiplerde denenmiş ve işe yaradığı defalarca kez kanıtlanmış sistemler üzerinde durmak gerekir.

Geleneksel teknikler derken, birinin söyleyip diğerlerinin yaptığı, düz mantık organizasyon şemasından bahsediyorum. Kararların alınmasının bir bürokratik sürece dahil olduğu, haliyle yavaş olduğu, süreç içerisinde istenildiği kadar değişiklik ve geliştirme yapılamayan, başı sonu net olarak tanımlı olan ama o tanımlardan herkesin bihaber olduğu proje yönetim teknikleri yani.

Bütün geleneksel teknikleri dışlıyor olmak istemesem de, bir şekilde aralarından bir tanesini seçeceğimiz için, diğerlerini her türlü dışlamış olacağız. Yıllardır bu süreçlerin birkaç tanesinin

aktif olarak içerisinde bulundum. Sonuç olarak, bir girişim açısından geleneksel yaklaşımın nasıl sorunlar yaratabileceğine birinci elden şahit oldum.

Bu sorunların başında hantallık geliyor. Girişimlerin hareket etmekte yavaş kalmaları, sonlarını hazırlayan en önemli etken olarak göze çarpıyor. Bu demek, bir şekilde bütün ekibi yönetim sürecine dahil edecek, bütün fikirleri belli açılardan değerlendirmeye alabilecek, bütün enerjiyi ortada toplayıp patlama yaratabilecek bir tekniğe ihtiyaç var.

Bu durumda, bir girişim için gerek yazılım alanında olsun, gerek iş geliştirme gibi diğer alanlarda, en uygun proje yönetim tekniği Agile (Çevik) yönetim olarak görünüyor. Bunun temel sebebi, muhtemelen haberdar olduğunuz “yalın girişim” süreçleriyle agile sistemlerin tam uyum içerisinde yürütülebilirliği. Bir sonraki bölümde nasıl uygulanabileceğini göreceğiz.

Yönetim tekniklerindeki değişimlerin bir girişimin hayat döngüsünü nasıl etkileyebileceğine dair düşündüğümüzde, karşımıza pek sık karşılaşılmayan bir sonuç çıkıyor: Proje yönetim tekniklerini kuralına göre uygulayan girişimlerle “biraz ondan biraz bundan” uygulayan girişimler arasında büyük bir üretkenlik açısı farkediliyor.

Bu üretkenlik farkının temel sebebi, insanların oturup bu proje tekniklerini geliştirirken bir takım mantık dizileri kullanmış olmaları. Yani eğer bir yönetim tekniği üzerine düşülecekse, onu oluşturanların oluşturma sebeplerini öğrenmek ve süreç içerisinde göz önünde bulundurmak gerekiyor.

Bu sayede, bahsi geçen mantık dizilerinin neden verili teknik içerisinde verili sırayla kullanılmış olduğunu daha iyi anlamak ve bu sayede daha üretken bir çalışma ortamı oluşturmak mümkün.

Bir girişimde, süreçlerin belli bir süreklilik içerisinde gelişiyor olması esas meselelerden bir tanesidir. Eğer tanımlıyor olduğunuz süreçler bir süreklilik içermiyorsa, ortaya çıkan sonuçlar da birbirinden kopuk olacaktır.

Bununla beraber, süreç içerisinde karşılaşılabilecek sorunlarla başa çıkabilme yöntemleri de çoğunlukla bu proje yönetim teknikleri tarafından içerildiğinden, teknikleri tam olarak anlamak önemlidir.

Bunların ardından, artık Agile proje yönetim tekniğinin bir hayranı olduğumu ve mümkün olduğunca herkese bu tekniği öneriyor olduğumu anlamışsınızdır.

Şimdi işin asıl eğlenceli kısmına, yani Agile tekniklerin girişiminizle nasıl etkileşimde bulunabileceğine ve süreçlerinize nasıl entegre edilebileceğine geliyoruz.

Sonraki iki bölüm tamamen bu başlığa ayrılmış durumda.

Agile (Çevik) Proje Yönetimi

Agile proje yönetimi metodolojisi, isminden anlaşılacağı gibi ortaya çevik bir süreç ve sonuç çıkartabilmek üzerine kurulu bir takım kurallar bütünüdür.

Bu kuralların temelleri yazılım projeleri için geliştirilmiş olsa da, girişiminizin genelinde belirli prensiplerini kullanarak ciddi fayda üretebilmeniz mümkün olacaktır.

Agile metodolojinin bütününü anlatmaya bir bölüm yetmeyecek, bu yüzden kısaca temel kurallarından ve süreçlerinden bahsedeyim:

1- Karar Süreçlerine Katılım

Her şeyden önce, işin geliştirilmesine katkı sunan bütün ekip üyelerinin, teknik karar verme süreçlerine aktif olarak katılması esas olarak belirlenmiştir.

Bunun temel sebebi, herkesin süreç içerisinde geliştirme yaparken bir takım sorunlarla karşılaşıyor ve yine bir takım çözümler üretiyor olmasıdır. Bu çözümlerin ortaklaştırılmasının ortaya büyük bir enerji çıkartacağı düşünülerek böyle bir prensip edinilmiştir.

2- Gerekliliklerin Değişmesi

Proje esnasında gerekliliklerin ve projeden beklentilerin değişiyor olması mümkündür. Bununla birlikte, Agile metodoloji içerisinde gereklilikler değişiyor olsa bile planlanmış zaman içerisinde planlanan iş tanımlarının değiştirilmemesi esas olarak belirlenmiştir.

Bunun sebebi, süreç sonunda oluşturulacak sonuçlar üzerinden karar mekanizmaları üretmenin çok daha kolay olmasıdır. Zaman skalası sabit tutulduğu sürece, işler planlandığı gibi

işlemeye devam edecek, birbirleri içine geçmek durumunda kalsa bile bir takım sonuçlar ortaya çıkabilecektir.

Kritik olan, ortaya sonuçların çıkabilmesidir, süreçlerin kendisi değil. Bu yüzden olabilecek bütün odağın sonuçları beklendiği gibi çıkartabilmek üzerinde olduğunu hatırlatmakta fayda var.

3- Küçük Adımlarla İlerlemek

Önceki bölümlerde bahsettiğimiz bir konu. Projenin mümkün olduğunca kendi başlarına tanımlanabilecek küçük parçalara bölünüyor olması yapılacak işi ciddi şekilde kolaylaştıracaktır.

Bununla birlikte, bu tanımlı küçük parçaların, adına “Sprint” denilen zaman yapılarına uyarlanması da geliştirme işini kolaylaştıracak bir başlıktır.

Parçaların kendilerini anlamlandırabileceği en küçük hallerinde olduğundan emin olduktan sonra, geliştirme sürecinin çok daha hızlı organize olabilidiğini göreceksiniz.

4- Adımların Sıralaması

Temel prensiplerden bir tanesi de, bir önceki maddede bahsi geçen adımların bir sıraya sokulmasıdır. Bu sokulan sıranın önemi, bir parça bitmeden diğer bir parçaya başlanmayacağının tanımlanmasıdır.

Bu prensibin sebebi, süreçlerin çıktılarının projenin genel gidişatı üzerine net bir fikir verebilmesini sağlamaktır. Buradan çıkan sonuç, projenin gidişatını net bir şekilde anlayabileceğiniz “Burndown Chart” denilen bir grafiği oluşturur. Bu grafik, size süreç içerisinde olması gereken ve mevcutta bulunan durumları net bir şekilde görebildiğiniz tek bir alan yaratır.

5- 80/20 Kuralı

Pareto kuralı olarak da adlandırabileceğimiz bu kural, sonuçların %80’inin süreçlerin %20’sinden çıktığını anlatır. Sayılar arada bir değişebilir durumda olsa da, ortalama olarak aynı kalırlar.

Bu kural sadece proje yönetimi açısından geçerli olmakla kalmaz, aynı zamanda hayatın pek çok alanında çeşitlenebilir. Örneğin Microsoft, Word kullanıcılarının çok büyük bir çoğunluğu yaptıkları bütün işi uygulamanın sadece %8’ini kullanarak yaptıklarını açıklamıştır.

Burada proje yönetim tekniğimiz açısından önemli olan, %20lik çalışma zamanı iyi değerlendirildiğinde, zamanın geri kalan %80lik kısmında çalışıyor olarak görünmeye gerek olmadığıdır. Kimsenin kendini kandırmasına gerek yok, gerçekten üretken olunarak işletilen bir süreç her türlü daha iyi sonuç verecektir. Zaman takibi yapmak konusuna ayrıca değineceğim.

6- Test Süreçleri

Test, bir yazılım projesinde en önemli başlıklardan bir tanesidir. İlla otomatik testler oluşturulmasından ve her seferinde kodların

bu testlerden geçirilmesinden bahsetmiyorum -ki keşke her proje böyle olabilse-.

İnsan gücü ile yapılan testler bile olsa, bütün yapılan işlerin tamamlandığını belirten son hareketin testi yapanlar tarafından gerçekleştirilmesi gerektiğini söylüyorum.

Dediğim gibi, Agile metodolojiye dair konuşulabilecek yazılabilecek çok fazla şey var. Proje yönetiminde ilerlemek isteyen herkesin bu metodolojiyi en azından bütün temellerini sağlama alarak öğrenmesi gerektiğini düşünüyorum.

Scrum

Scrum, agile metodolojinin en kilit parçasıdır. Kısa süreli, mümkünse ayakta ve tamamen konuya odaklı bir toplantı olarak adlandırabiliriz.

Scrum ismi, Okyanusya bölgesinde yaygın şekilde oynanan, Amerikan Futbolu benzeri bir oyun olan Rugby'den gelir.

Rugby'de takımların sahanın belli bölümlerinde toplanarak bir sonraki hareketlerini planlayıp ardından harekete geçtiği ufak süreçlerin adı olan scrum, Agile metodolojiye de bu vesileyle girmiştir.

Scrum'ın nasıl işlediğini anlayabilmek için, içerisinde bulunan birkaç kilit terimi inceleyelim:

1- Product Owner

Sürecin orada bulunmasını sağlayan kişidir. Geliştiriliyor olan ürünün tam tanımını, başını, sonunu, vizyonunu bilir. Bununla birlikte sürecin olması gerektiği yöne doğru gidiyor olduğundan emin olmakla yükümlüdür.

2- Scrum Master

Scrum süreçlerinin olması gerektiği gibi işliyor olduğundan emin olması gereken kişidir. Scrum sürecinin nasıl geliştiğini, bu süreçler kullanılarak nasıl başarıya ulaşılabileceğini bilir ve tüm takımı ona uydurmaya çalışır.

Sadece takımı sisteme uydurmaya çalışmakla kalmayıp, sistemi takıma uyumlu şekilde işler hale getirmek gibi bir göreve de sahiptir.

3- Scrum Meeting

Genelde sabahları işe başlarken uygulanan, mümkün olduğunca 15 dakikayı geçmeyen, bütün takımın aynı alanda ve ayakta durarak yaptığı toplantıya denir.

Herkesin sırayla konuşması ve 3 tane soruya tam ve net cevaplar vermesi esastır:

1- Dün ne yaptım?

2- Bugün ne yapacağım?

3- Bugün yapacak olduğum işle ilgili karşılaşabileceğim sorunlar neler?

Bu 3 soruya herkes sırayla ve hızlıca cevap vermekle yükümlüdür. Eğer 3. Sorunun cevabı kişinin kendi başına çözemeyeceği bir soruna işaret ediyorsa, takım içerisinden birileri o sorunu çözmekte yardım edebilir. Lâkin sorunun çözüleceği değil, tanımlanacağı yer olduğundan, çözümün süreci yavaşlatmaması önemlidir.

4- Sprint

Herkes kendi alanında yapması gerekenleri tam olarak netleştirdikten sonra, ilk toplantının ardından bir sprint planlanır. Sprint terimi, kısa mesafe koşusu anlamına gelir. Bu şekilde adlandırılmasının sebebi, takım içerisindeki herkesin belirli bir süre içerisinde (ortalama 2 haftalık bir süreç) yapacakları her şeyin tam ve net olarak tanımlanıyor olmasıdır.

Tanımlamalar tam olarak yapıldıktan sonra, her gün yapılan scrum toplantılarının sonucu olarak mevcut tanımlı sprint'in neresine gelindiği netleştirilmeye çalışılır.

5- Backlog

Ürünün sıfır noktasından ortaya çıkışına kadar geçen süreçte hangi işlerin yapılacağı yazılan listeye backlog denir. Herkesin kişi olarak işlerinin tanımlı olması gerekmez, lâkin işlerin kategorileri bellidir. Yani yazılım ekibinin yapması gereken işler, önyüz geliştirmek için yapılacak işler gibi işbölümleri nettir.

Herbir scrum toplantısı sırasında, takım üyeleri backlog içerisinden o gün yapacakları işleri üstlerine alırlar. Bu sayede sürecin işliyor olduğu net olarak görüntülenebilmiş olur.

6- Burndown Chart

Sürecin nasıl işlemesinin beklendiği ile, nasıl işlediğinin aynı grafik içerisinde gösterilmesiyle burndown grafiği elde edilir.

Bu grafiğin bulunmasının temel sebebi, sürecin tam olarak istenildiği gibi mi, istenilenden yavaş mı, yoksa hızlı mı olduğunun tek bir noktadan görüntülenebilmesini sağlamaktır.

Scrum süreçleri detaylı olarak düzenlenebilecek ve geliştirilebilecek süreçlerdir. Küçük projelerde rahatlıkla uygulanabileceği gibi, çok büyük ekiplerin de uygulamasında bir sorun olmamaktadır.

Bir girişim için Agile metodoloji ve scrum yönteminin kullanılması kritiktir, zira aksi halde süreçlerin takibinin geleneksel yollarla yapılması gerekecek ve bu durumda

scrum’ın ortaya çıkmasına sebep olan sorunlarla karşılaşılacaktır.

Yazılım süreçleriyle iş geliştirme süreçlerinin içiçe geçmesini sağlayan Agile metodolojinin bir güzelliği de, herkesin sürecin gelişiminden aynı derecede haberdar olabilmesidir. Bu özellikle küçük takımlarda çok kritik bir başlık olduğundan, uygulanmasını öneririm.

Scrum master her yerde bulunabilen bir şey değildir, lâkin öğrenilmek istendiğinde öğretebilecek kaynaklar her yerden temin edilebilir.

Sürdürülebilir Yazılım

Kendi başına ayrı bir kitap oluşturabilecek bir başlık da, sürdürülebilir yazılım başlığı. Sürdürülebilir yazılımın tanımı, yazılımcı(lar) değişse bile tekrar yazılmak durumunda olmayan yazılım olarak yapılabilir.

Yazılımlar, teknolojinin doğası gereği belirli periyotlarla yeniden yazılmak durumunda kalırlar, lâkin bu durum sürdürülebilir yazılım üretmenin önemini etkilemez.

Sürdürülebilir yazılımın bir girişim için öneminden bahsedecek olursam, eğer geliştirdiğiniz yazılım, ortalama bir rakip yazılımdan daha kısa süre yaşayabilirse, sadece baştan doğru önlemleri almadığınız için kaybedeceğiniz zaman ve paranın canınızı bir hayli sıkacağını söyleyebilirim. Önemli olduğunu düşündüğüm birkaç sürdürülebilirlik başlığını listeleyeyim:

1- Kod Kalitesi

Önümüzdeki bölümlerde daha detaylı inceleyeceğim, lâkin burada da ismini geçirmek gerekiyor. Kod kalitesi, yazılan kodun makineler tarafından olduğu kadar, insanlar tarafından da anlaşılabilir olması anlamına geliyor.

İnsanlar için anlaşılabilir kod yazmak, bir yazılımcıyı iyi yazılımcı yapan özelliklerin başında geliyor. Bunun temel sebebi, hiçbir yazılımcının proje belirli bir büyüklüğe ulaştıktan sonra sürecin bütününe tek başına hakim olamayacağı gerçeği.

Yani sadece başka yazılımcılar tarafından değil, üzerinden belirli bir süre geçtikten sonra yazan yazılımcının kendisi için de anlaşılır bir kod tabanı oluşturulması gerekiyor. Bu süreçler düşünülerek geliştirilen yazılımlar, uzun süreler boyunca ayakta kalabiliyor.

Bununla birlikte önemli bir nokta şu: Yazılımın çalışıyor olması, kodun doğru yazıldığı anlamına gelmez. Çoğu banka altyapısı bugün şans eseri çalışıyor durumdadır.

Bunun gibi, pekçok açıdan insanların teknolojiye hakim olmaya başladığını düşünürsek, hızlı gelişen bir girişimin pekçok teknik saldırıya maruz kalacağı da bir gerçek. Bunlara gerekli zamanlarda gerekli müdahalelerin yapılabilmesi için kod kalitesi çok önemli bir başlık.

2- Kodun Geliştirilebilirliği

Kodun sadece kaliteli yazılmış olması yetmiyor, üstüne bir de geliştirilebilir olması gerekiyor. Dışarıdan bakıldığında çok kolay anlaşılabilecek bir durum değil, kısaca şöyle anlatayım: Bir girişim gelişme sürecinde kullanıcıların geribildirimlerine adapte olabilmesiyle kendini var eder.

Bu da, sürekli olarak bir şeylerin değişeceği, yeni şeylerin ekleneceği ve kimi özelliklerin çıkartılacağı anlamına gelir.

Sadece kağıt üzerinde yazılan kadar iş yapabilen ve sadece o kadar iş yapabilecek bir yazılımın geliştirilmesi, geribildirimler gelmeye başladıktan sonra zora girecektir. Bu süreçle karşılaşıldığında zora girmemek için, yazılımın ileriki süreçlerde geliştirilebilir olduğundan emin olmak gerekir.

3- Kuvvetler Ayrılığı İlkesi

Fazlasıyla teknik bir konu olmakla beraber, ilk maddede belirttiğim durum yeterince düzgün oluşturulursa herkes tarafından anlaşılır bir hale gelebilir. Burada kastım, kod içerisindeki herbir fonksiyonun yapması gereken tek ve en küçük işlemi gerçekleştiriyor olduğundan emin olunmasıdır.

Kuvvetler ayrılığı olarak adlandırmamın sebebi, yazılan herbir fonksiyonun kendine ait bir kuvvet teşkil ettiğini düşünürsek, bu kuvvetlerin belirli bir sırayla doğru olarak işletilmesi kadar, kendi başlarına da bir anlam ifade edebilen küçük parçalar halinde organize edilmesi gerekliliği.

4- Sürekli Denetim

Girişim sadece girişimciye bırakılamayacağı gibi, yazılım süreçleri de sadece yazılımcıya bırakılmaz. Bırakılırsa ortaya ne çıkacağından emin olunamaz.

Yazılımın, yazan kişinin kalitesi ve ona duyulan güven iade edilerek, bir denetim sürecine sokulması gerekir.

Bir yazılımın sürdürülebilir olup olmadığının denetlenmesini bu alandaki profesyoneller yapabileceği gibi, girişimciler kendileri de yapabilirler.

Bunu yapabilmek için “sürdürülebilirlik takibi” adını verdiğim bir süreç tanımladım. Bu süreç tanımı yeni değil, lâkin girişimler

için uyarlanması gerektiğinden mümkün olduğunca sade bir hale getirmek gerekti.

Sürdürülebilirlik takibinin 4 tane ana başlığı var: Code Review, Quality Assurance, İş Takibi ve Zaman Takibi. Bunları teker teker inceleyelim.

Kod İncelemesi (Code Review)

Code Review, ya da Türkçe adıyla kod incelemesi, kısaca yazılan kodun yazan haricinde biri veya birileri tarafından objektif bir gözle gözden geçirilmesi demek.

En iyi yazılmış kodun, makinelerle beraber insanların da anlayabileceği şekilde yazılmış olması gerektiğini daha önce de söylemiştik.

Bu durum elbette kodlama bilmeyen birinin bütün olan biteni anlamasını sağlamalı anlamına gelmiyor.

Lâkin kod incelemesi yapabilmek için kodlama bilmeye gerek bırakmayacak birkaç ipucunu sizinle paylaşmak isterim, zira eğer kodlama bilmiyorsanız aksi durumda işiniz biraz zor olabilir:

1- Objektif Göz

Yazılımcıların kendi kodlarını inceleyemeyecek olmalarının sebebi, kod inceleme sürecinin objektif bir göze ihtiyaç duymasıdır. Süreç uzun vadeli olsaydı, yazılımcıların kendi kodlarını incelemelerinde bir sıkıntı olmazdı, lâkin bir girişim içerisinde zaman tanımları pek de geniş olmayacağından, sürecin hızlıca çözüme kavuşturulmasına ihtiyaç olacaktır.

Bu noktada, her şeyden önce, kodu yazan kişi veya kişilere dair herhangi bir sorun başlığı üretmiyor olduğunuzu belirtmek isterim. Yazdığı kodun kodlama bilmeyen birileri tarafından “incelenmesi” düşüncesi, bir yazılımcı için hoş bir durum değildir, lâkin buradaki niyet yazılımcıyı yargılamak değil, aksine sürece hep birlikte hakim olunduğundan emin olmak.

Bu yüzden kimseye ayıp olacağını düşünmeyin. Gerekli açıklamalar yapıldıktan sonra herkes bu sürecin gerekliliği konusunda hemfikir olacaktır.

2- Kod İçindeki Yorumlar

Kod içerisinde bulunan, yazılan kodun açıklamasını yapmak için kullanılan kod harici parçalara “yorum (comment)” denir. Yorumlar, ayrı bir satır, ayrı bir blok veya bir satır içerisindeki son bölüm şeklinde kod içerisinde yer edinebilirler.

İyi yorumlar, önümüzdeki bölümlerde göreceğimiz “dokümantasyon” oluşturmayı destekleyebilecek şekilde yazılmış yorumlardır.

Yazılımcının yazdığı kod parçasının tam olarak işlevini anladığı ve bunu dış dünyaya anlattığı parçaları iyi düzenlemesi, sürecin doğru şekilde işleyebiliyor olduğunun en basit kanıtlarından bir tanesidir.

İyi yazılmış kod, içinde bolca yorum barındırır. Bunun sebebi, herhangi bir yazılımcı değişimi sonunda veya kodun yazılması üzerinden belli bir zaman geçtikten sonra, birilerinin o kodun neden yazılıyor olduğunu anlamak durumunda olmasıdır. Bu durum, öyle ya da böyle bir şekilde gerçekleşeceğinden, ona iyi hazırlanmış bir yazılımcı, işini iyi yapıyordur denilebilir.

Elbette tek kriter değil, lâkin önemli bir kriter olduğu kesin.

3- Kilit Noktalar

Kodun tümünü inceleyemezsiniz. Aksini düşünüyorsanız deneyin, lâkin inceleyemeyeceğinize bahse girebilirim. Bunun sebebi, kodun yazılması işleminin çok büyük bir kısmının yardımcı alanları oluşturmak için yapılıyor olmasıdır.

Yani, ortada asıl işi yapan bir kod parçası vardır ve geri kalan bütün kodlar onun o işi rahatça yapabilmesi için organize edilmişlerdir.

Bu noktada yapmanız gereken, o asıl işi yapan kod parçasının nerelerde bulunduğunu yazılımcınıza sormak ve incelemelerinizi onlar üzerinden yürütmektir.

Chimes her şeyi doğru yapamaz, kanımca yapmakla yükümlü de tutulmamalıdır. Bu sebepten, eğer kilit noktaları doğru ve olması gerektiği doğrultuda yapan bir yazılımcınız varsa, bu, o kodun emin ellerde olduğuna dair rahat bir nefes alabileceğiniz anlamına gelir.

Elbette bunun bir garantisi yok, lâkin bir şekilde ortada bir güven ilişkisi geliştirme ihtiyacı var. Özellikle söz konusu bir girişimse.

4- Standartlara Uyum

Kod yazmanın belli bir standardı yoktur. Herkesin kendine özgü standartları, isimlendirme kuralları, yorum yaklaşımları olabilir. Bunun olması gayet normaldir.

Lâkin burada kritik olan, yazılımcının kendisi ve ekip içerisinde bir stabiliteye sahip olması gerekliliğidir. Yani, ortada kullanılan

bir yöntem varsa, o yöntem kullanılmaya devam etmelidir. Aksi durumda kodun incelenmesi ve geliştirilmesi zorlaşacak, süreç istenilmeyen noktalara evrilebilecektir.

5- “Commit” Düzeni

Bir önemli nokta da, versiyon kontrol sistemlerinde kısaca bahsettiğim “commit”ler hakkında. Kullanılan versiyon kontrol sistemi içerisinde yazılımcınız tarafından oluşturulan pekçok “commit” görmeniz mümkün olacaktır.

Buradaki kritik nokta, yapılan “commit”lerin belli bir düzende ve aynı kapsam düzenini koruyarak yapıldığından emin olmaktır. Olması gereken, her işin kendi içerisinde bir “commit” çıktısı olması ve bunun doğru zamanda yapılıyor olmasıdır.

Kalite Güvencesi (Q&A)

Q&A, yazılımda kullanılan bir terim olarak “Kalite Güvencesi (Quality Assurance)” kavramının kısaltmasıdır. Kısaca, yazılımın kaliteli olmasını güvence altına alan yapıların denetimine verilen isim olarak düşünülebilir.

Bu alanda uzmanlaşmış bir mühendislik dalının da bulunduğunu düşünürsek, bir bölüm içerisinde bütün olan biteni anlatmak pek de kolay değil. Lâkin kapsamımız girişimciler olduğundan süreç biraz daha kolay olacak.

Her şeyden önce, bir yazılımcının kalitesini belirleyen bir önemli nokta da, yapılan işe yoğunlaşma durumu olarak karşımıza çıkıyor. Yani, bir yazılımcı ne kadar iyi olursa olsun, ne kadar kaliteli olursa olsun, o anda yaptığı işe yoğunlaşmıyorsa (veya daha önce aynısını yapmadıysa), ortaya çıkan sonuç onun kalitesiyle yarışabilecek bir düzeye gelemeyecektir.

Bu sebepten, bir kalite değerlendirmesi yapılacaksa, o değerlendirmenin yazılımcı için değil, yazılan kod için yapılıyor olduğunu belirtmek isterim. Bu yüzden bu kalite değerlendirmesini de kişisel anlamamak ve o şekilde davranmamak gerekir. Sonuçta ilişkileri yıpratmak için değil, işi geliştirmek için uğraşıyoruz.

Birkaç başlıkta bu alanda dikkat edilmesi gerekenleri sıralamak isterim:

1- Gerekliliklerin Tanımı

Her zamanki gibi, her şeyden önce yapılması gereken işlerin tam ve net olarak tanımlandığından emin olunması gerekir. Sadece girişimcinin değil, bütün takımın bu tanımlamanın içerisinde tutulması önemlidir.

Ardından yapılacak şey, mümkün olduğunca işlerin sınırlarının belirlenmesi olacaktır. Bunu proje başlangıcı bölümünde anlattığım yöntemleri kullanarak rahatlıkla yapabilmeniz mümkündür. Tek dikkat edilmesi gereken nokta, herhangi bir denetim yapılacaksa, bu denetimin belirli kriterler eşliğinde yapılması gerekliliği ve o kriterlerin ancak sürecin bütünü değerlendirilerek oluşturulabileceğidir.

2- Günlük Takip

Scrum bölümünde de bahsettiğimiz gibi, günlük yapılan toplantıların proje geliştirme sürecindeki önemi tartışılamayacak kadar büyüktür.

Bu toplantılardan yazılım süreci için elde edilmesi gereken temel sonuç, o gün içerisinde yapılacak işlerin tam ve net olarak herkes tarafından anlaşıldığından emin olunmasıdır. Eğer yazılım sürecinde “o gün” yeterince iyi anlaşılırsa, o gün yapılacakların süreç içerisinde kontrol mekanizmalarının oturtulması da bir o kadar kolay olacaktır.

Kalite güvencesini kontrol altına almak için, özellikle bir girişimde herkesin o gün kodun hangi parçasının yazıldığını bilmesi gerekir. Bu sayede işin doğru şekilde işlediğinden emin olmakla beraber, sürekli kodu okumak zorunluluğundan da kurtulunmuş olur.

3- Düzenli Kontrol

Günlük olmasa bile, belli bir düzen içerisinde yapılacak kontroller ile kodun olması gereken doğrultuda ilerleyip ilerlemediği anlaşılabilir.

Kalite güvencesi kontrolünün normal testlerden farkı, sürecin işliyor olmasından ziyade istenildiği gibi gelişiyor olduğunun da denetlenmesidir. Bunu yaparken ihtiyaç duyulabilecek kimi araçlar mevcut olsa da, onlar bir girişim için fazla profesyonel kalabilirler. Bu yüzden kodun denetimini mümkün olduğunca araya zaman koyarak yapılan standart kontrolleriyle yapmak faydalı olacaktır.

4- Girişimcinin Rolü

Yazılım süreçleri, düzenli kod kalitesi ve proje bütünlüğü kalitesi kontrolleriyle hızlandırılabilir ve güçlendirilebilir. Burada girişimcinin rolü yazılan kodun veya onu yazanın sorgulanması değil, aksine projenin istenilen niteliklere sahip olduğundan emin olması ve eksik noktaları doldurmaya çalışması olacaktır.

Bu bölümün biraz karmaşık olduğunun farkındayım. Bunun sebebi, Q&A işinin pek çok başlığının olması ve yazılım süreci içerisinde pek çok farklı anlamda kullanılmaya açık olması.

Hepsinden daha kısa şekilde anlatmak gerekirse, ortada asıl yapılıyor olan, kodlamanın belli standartlara uygun olarak yapılıyor olup olmadığının denetlenmesidir. Bunun için kod standardı oluşturabilecek bir departmanınız olmadığından da,

standartların yine yazılımcı tarafından oluşturulması gerekecektir.

Yani yapmanız gereken, kendi kurallarını belirlemesi için gerekli alanı sağladığınız yazılımcınızın kendi belirlediği kurallara uyup uymadığını denetlemek, hepsi bu.

İş Takibi

İş takibi, günlük scrum toplantılarında kararlaştırılan işlerin yapılıp yapılmadığının kontrol edilmesine denir.

Burada önemli olan, scrum içerisinde tam ve net olarak tanımlanmış bir işe sahip olunması ve onun yükümlülüğünün alınmasıdır.

Kısaca tanımlamak gerekirse, bir yazılımcı, o gün yaptığı işi tam ve net olarak tanımlayamıyorsa, o gün aşağı yukarı hiçbir şey yapmamıştır diyebiliriz.

Bu biraz sert görünebilir, lâkin yapılan iş kod yazmak olduğunda, her şeyin belirli bir düzen içerisinde ilerlemesi ve sonuçlandırılması gerekir. Bu görev de yazılımcıya ait olduğundan, sürecin çıktılarının sorumluluğu da ona aittir.

Öte yandan, bir yazılımcının o gün içerisinde hiçbir şey yapmamış olması, sadece kendisiyle alakalı olmayabilir. Hatta girişim projelerinde daha sık karşılaşılan bir durum olarak, işin tam ve net tanımı kendisine aktarılmadığından veya yanlış aktarıldığından dolayı bu sorunla karşılaşmış olması da mümkündür.

Bu gibi sonuçlarla karşılaşmamak için, sürdürülebilir yazılım geliştirme sürecinin bir parçası olarak tanımladığımız iş takibini olması gerektiği gibi yönetebilmek gerekir. Bu yönetim, sürecin öncesini ve sonrasını da kapsar. Birkaç ipucunu sıralamak isterim:

1- İş Tanımları

Yazılım söz konusu olduğunda, bunu tekrar tekrar söylemek durumundayım, yapılması beklenen her işin tam ve net olarak tanımlanmasına ihtiyaç vardır. Eğer doğru iş tanımlamaları

yapılmazsa, ortaya doğru iş çıkmasını beklemek pek de mantıklı olmayacaktır.

Bununla birlikte, yazılımcıların ortada iş tanımı olmadan da harikalar yaratması olasıdır. Lâkin buradaki mesele, tanımlı bir iş olmadığı durumda yaratılan harikaların projenin bütününe fayda sağlayıp sağlamadığının net olarak anlaşılmasıdır.

Yani, eğer ihtiyaç duyulan alanda ilerletilmesi gereken kritik başlıklar yerinde sayıyor ve onlar yerine daha az önemli başlıklarda ilerleme kaydediliyorsa, bunun süreç açısından nasıl bir sorun yaratabileceği üzerine net bir fikir üretmek gerekebilir.

Her zaman sorun olarak değil elbette, lâkin tam olarak bulunulan yerin her şekilde bilinmesi, bir girişim açısından nerede olunduğunu tam anlamayı sağlayacaktır.

Bu süreçler içerisinde yapılacak işin tanımı ne kadar tam ve netse, işin yapılıp yapılamadığı da o kadar net olur. İş takibinin ne kadar sıkı olabileceğini belirleyen, girişimcinin karakterinden ziyade, süreç içerisindeki iş tanımlarının ne kadar tam ve net yapılıyor olduğudur.

2- Son Bırakılan Yer

Scrum toplantılarının amaçlarından bir tanesi de, herkesin tam olarak nerede kaldığının net bir şekilde görünür hale gelmesini sağlamaktır. Bu amaç doğrultusunda gerçekleştirilen günlük scrum toplantılarından ortaya çıkacak sonuçlar, herkesin aynı sayfaya bakıyor olduğundan emin olmayı sağlar.

Son bırakılan yerin önemli olmasının sebebi, işin gerçek anlamda ilerliyor olup olmadığını anlamanın gerekmesidir.

Günlük hayatta bir girişimcinin dikkatini dağıtacak tonla şey varsa, bir yazılımcının dikkatini dağıtacak da en az o kadar şey olacaktır.

Bu dikkat dağınıklığının yazılan koda yansımaması kişisel disiplinle ilgili olduğu kadar, iş yönetiminin ne kadar güçlü olduğuyla da ilgilidir.

Eğer aynı yerde duran bir sürece sahipseniz, o sürece müdahale edilebilecek olanakları araştırmaya başlayabilirsiniz. Bu bağlamda da iş takibi önemlidir.

3- Bütünün Parçaları

Yazılım geliştirme süreçleri, birbirini takip eden küçük parçaları birleştirerek bir bütünlük yaratma süreçleridir demiştik. Parçaların bölünmesi ve bütünü oluşturması süreçlerinin öneminden de bahsettik.

Burada ihtiyacımız olan, bütünün herbir parçasının ayrı ayrı takip edilebilmesini sağlamaktır. Bunu yapmanın en kolay yolu, burndown grafiği dediğimiz, agile yöntem bölümünde bahsi geçen grafik olacaktır.

Burndown grafiği, gerek iş, gerek zaman takibi konusunda sık sık başvurmamız gereken bir grafik olarak baş köşede yerini almalıdır.

İş takibini güçlü biçimde yapabilmek, bir girişimin iş disiplinini ve iç kültürünü yaratabilmesi açısından çok önemlidir.

Süreci mümkün olduğunca parçalara ayırarak takip etmek ise, söz konusu sürecin çok daha kolay yönetilebilmesine olanak tanıyacaktır.

Zaman Takibi

Sürdürülebilirlik takibinin en temel parçalarından bir tanesi de, zaman takibidir. İş takibi gibi, doğru yapılmadığı takdirde ölümcül olmasa da, doğru yapıldığında işi ciddi oranda iyileştirecek ve hızlandıracak bir başlıktır.

Doğru kullanılan zamana dair pek çok kitap, tonla üretkenlik yöntemi sunacaktır. Bunun için geliştirilmiş uygulamalardan, video seminerlere kadar pek çok kaynak mevcuttur. Bunlar için verebileceğim en iyi tavsiye, her geçen gün yeni bir yöntem çıkan günümüzde, bir yöntemi belirleyip ona sadık kalmak olacaktır. Aksi takdirde, harcanılan zaman da emek de her seferinde boşa gidebilir.

Doğru kullanılan zaman, illa uzun bir zaman parçası olmak durumunda değildir. Lâkin söz konusu yazılım olduğunda, kesinlikle planlı bir zaman parçası olmalıdır.

Örnek olarak, bir günlük olarak tanımlanmış bir işin 1-2 saatte bitirilebiliyor olması, tek başına bir yazılımcının ne kadar verimli çalıştığını göstermez. Bununla birlikte, söz konusu yazılımcının veya zaman tahminini yapan sorumlu kişinin sürece tam anlamıyla hakim olmadığını da gösterir.

Aynı durum 1-2 saatlik olarak tanımlanmış bir işin 1-2 gün sürmesi için de geçerlidir. Zaman tahmini yapılırken arada tampon zamanların da kullanılabileceğinden bahsetmiştik. Örnek olarak 2 saatlik görünen bir işin zaman tahminini 4 saat olarak yapmış olmakta bir sakınca olmayacaktır.

Lâkin, 4 saatlik olarak tahmini yapılan ve normal koşullarda 2 saatlik olacağı öngörülen bir işin 1-2 gün sürmesi, zaman

tahmini yapanın bilgi veya tecrübe eksikliğini gösterir. Tersi için de aynı durum geçerli.

Burada, bir yazılımcının kalitesine dair bir metrik daha ortaya çıkmış oluyor. Yapıp yapamayacağı şeyleri biliyor olmak, yapabileceklerinin ne kadar zaman alabileceğini tahmin edebiliyor olmak, ciddi şekilde süreçleri ileri götüreceğinden, yazılımcının kalitesi için büyük bir artı puan olarak yazılabilir.

Elbette hiçkimse bütün süreçlerde her şeyi beklenildiği gibi yapamayacaktır, lâkin zaten takip süreçlerinin temel amacı budur. Asıl mesele yazılımcının üstüne çökmek değil, üretmeye çalıştığı şeyi en rahat şekilde üretebilmesini sağlamaktır.

Bunu da belirttikten sonra, Proje Geliştirme Süreci bölümünün de sonuna gelmiş bulunuyoruz. Yazılımcı bulmaktan fikri analiz etmeye kadar geldiğimiz ilk bölümün ardından, ikinci bölümde proje yönetimi ve takibine dair mümkün olduğunca çok ipucu içermeye çalıştım. Girişim süreçlerinizde faydası olacağını umuyorum.

Sıra, proje geliştirildikten sonra dikkat edilmesi gerekenlere geldi. Önümüzdeki bölüm, proje geliştirildikten sonra doğru yapıldığından emin olunması gereken dokümantasyon ile başlıyor. Ardından yeni bir yazılımcıya projenin nasıl devredilebileceğini, sonrasında da proje bakım anlaşmalarını ve anlaşmazlık durumlarının nasıl giderilebileceğini anlatıyor.

Bir girişimin yaşam döngüsü içerisinde sürekli karşılaşıyor olduğu pek çok sorun başlığı var, yazılım geliştirme de bunlardan bir tanesi. Yazılım geliştirme başlığının sorun olmaktan çıkmasını sağlayabilecek şey ise, onun ciddiye alınıp doğru bir planlama ve uygulamayla yürütülmesi olacaktır.

Doğru bir planlama ve yürütülmenin ardından yine de sorunlarla karşılaşılabilir, lâkin bu karşılaşılan sorunlar çok daha ufak çaplı olacağından çok daha kolay çözüme ulaştırılabilir.

Her şeyden önce, birlikte çalışacağınız yazılımcılarla aranızda bir güven ilişkisi oluşturmanın en temel nokta olduğunu tekrar söylemek isterim. Bir girişimde ortada güven ilişikisi yoksa, dünyanın en iyi kadrolarıyla da çalışıyor olsanız sonuç yine de beklediğiniz gibi olmayacaktır.

Beklediğiniz her zaman en iyisidir diye söylemiyorum, lâkin bir girişimci olarak girişiminizden beklediklerini ve onun erişmesini istediğiniz noktaların iyi noktalar olacağını düşünüyorum. Bu yüzden her ne olursa olsun güven ilişkisi temelinden sapmamanızı öneririm.

Ardından gelecek adımlar, süreç içerisinde öğrenilebilecek ve oturtulabilecek adımlardır. Hepsini daha ilk günden biliyor olmanız gerekmez, yapmanız gereken, adım atmaya devam etmektir.

Proje Sonrası

3

Yazılım süreçleri zor süreçlerdir. Hele bir de girişim süreçleriyle birleştiğinde, çok daha zor hale gelebilir.

Girişimin ve yazılımın ayrı ayrı pek çok badire atlatmak durumunda olduğunu düşünürsek, bunları birleştirebilmek bir hayli zor bir iştir aslında.

Dokümantasyon

Proje geliştirildikten sonra ortaya büyük ölçüde çıkmış olması gereken en önemli doküman, projenin dokümantasyonudur. Dokümantasyonun önemli olmasını sağlayan, hiçbir projenin yap-at şeklinde gelişemeyeceği gerçeğidir.

Teknoloji bu hızda gelişirken hiç geliştirilmeyecek bir proje oluşturduysanız, elbette dokümantasyona ihtiyacınız olmayabilir, lâkin eğer üzerinde herhangi bir şekilde bir geliştirme yapılacaksa, dokümantasyona kesinlikle ihtiyacınız vardır.

Doğru geliştirilen dokümantasyon yazılımcıya çok büyük bir ek iş çıkartmayacaktır. Kaliteli bir kod tabanına sahip olan projelerin dokümantasyonunu geliştirmek hiç de zor değildir.

Dokümantasyon geliştirirken faydalı olabilecek birkaç ipucunu sıralamak isterim:

1- Kodun Kendisi

Kodun incelenmesi sürecinde yorumların ne kadar önemli olduğundan bahsetmiştim. Yorumların düzgün ve düzenli şekilde yazılmasının en temel faydalarından bir tanesi de dokümantasyon oluşturmayı kolaylaştırmasıdır.

Hatta sadece kolaylaştırmakla kalmaz, tamamen otomatik üretilmesini bile sağlayabilir.

Sonuç olarak bir yazılımın dokümantasyonunu geliştiriyor olduğumuzdan, kodun belirli bölümlerinin tam olarak içerildiğinden emin olmamız gerekiyor. Bunun da en kolay yolu, kodun içerisindeki yorumları sonradan dokümantasyona çevrilebilecek şekilde organize etmek olacaktır.

Öte yandan, kodun kendisi herhangi bir yorum içermiyorsa veya içerdiği yorumlar yetersiz durumdaysa, dokümantasyon oluşturmak tam bir angarya iş olacaktır.

Bunun sebebi de, sürecin zaten yazılmış olan kodun tekrar baştan bütün incelikleriyle gözden geçirilmesini ve dokümantasyon için tekrar organize edilmesini kapsamak zorunda olmasıdır.

Kod yazılırken sonuç itibariyle bir dokümantasyon ortaya çıkartılması gerektiğini yazılımcının işin en başından bilmesi gereklidir. Aksi durumda sürecin daha hızlı ilerlemesi mümkün olacağı gibi, bir sonraki adıma geçmek çok daha zorlayıcı olacaktır.

2- Commit Düzeni

Versiyon kontrol sistemlerinden bahsederken, herbir tanımlı işin ayrı bir commit olarak organize edilmesi gerektiğini söylemiştim. Geldiğimiz noktada, dokümantasyon oluştururken ciddi şekilde faydasını görebileceğiniz bir yapı olarak versiyon kontrol sistemleri tekrar karşımıza çıkıyor.

Düzenli biçimde organize edilirse, versiyon kontrol sistemlerine girilen commit'ler projenin dokümantasyonuna iyi bir katkı sağlayacaktır. Kod içerisine yazılan yorumlarla birleştirildiğinde, neyin ne için yapılıyor olduğu tam olarak ortaya çıkacak ve bir sonraki geliştirme adımında bu bilgiler rahatlıkla kullanılabilecektır.

Mesele projeyi olabildiğince güçlü şekilde bir sonraki geliştirme adımına hazırlamak olduğundan, uygulanan her bir işlemin açıklamasının bulunması faydalıdır.

Bir sonraki geliştirme adımında orada bulunan yazılımcılar, bir önceki işlemleri ne kadar hızlı anlarlarsa, projeye o kadar hızlı adapte olurlar.

3- Dış Belirleyenler

Her şeyin birbirine bir şekilde bağlı olduğu günümüzde, bir projenin de herhangi bir dış kaynak kullanmadan varolmasını düşünmek pek de mantıklı değil.

En azından sosyal medya hesaplarını kullanarak giriş yapılması bile dış bir belirleyenin sürecin içine sokulması anlamına gelecektir.

Sürecin içine sokulan bütün dış belirleyenlerin büyük ihtimalle kendilerine ait dokümantasyonları olacaktır, lâkin sizin yapınız içerisinde bulundukları yerlerin ve bulunma amaçlarının net bir şekilde dokümantasyonunuza aktarılması gerekecektir.

4- Tekrara Düşmemek

Kod yazarken de, dokümantasyon oluştururken de geçerli olan bir prensip var, ismi "DRY". "Don't Repeat Yourself (Kendini

Tekrar Etme)”in kısaltması. Bu prensibi temel aldığımızda geliştirdiğimiz kodun ve dokümantasyonun sade ve net bir şekilde işlevini anlatıyor olacağını söyleyebilirim.

Elbette dokümantasyon geliştirmek başlı başına bir alan olarak üzerinde uzun saatler çalışmayı da gerektirebilir.

Özellikle geliştirdiğiniz sistemin dışarıya açık bir Uygulama Programlama Arayüzü (API) varsa, üzerinde bir süre çalışmak durumunda kalacağınız aşikar.

Burada dikkat edilmesi gereken başlık, kodun yazılması ve dokümantasyonun oluşturulması sürecini bir bütün olarak kabul edip o bütün içerisinde tekrara düşmekten kaçınmak olmalı.

Bu sayede süreçler çok daha rahat ve acısız geçebilir.

Yeni Yazılımcı

Yazılım süreçleri zor süreçlerdir. Hele bir de girişim süreçleriyle birleştiğinde, çok daha zor hale gelebilir. Girişimin ve yazılımın ayrı ayrı pek çok badire atlatmak durumunda olduğunu düşünürsek, bunları birleştirebilmek bir hayli zor bir iştir aslında.

Haliyle her zaman her şey istenildiği gibi gitmez. Aynı yazılımcıyla bir ömür boyu çalışamayabilirsiniz, ki bu durum hızlı gelişen bir piyasada gayet normaldir. Doğal olarak bu durumla karşılaşıldığında neler yapılacağına dair net fikirler geliştirmemiz gerekiyor.

Aslına bakarsanız, bu kitap bir yanıyla bu sürece hazırlık yapıyor. Fikrin geliştirilme sürecinden başlayarak, uygulamaların yazılma süreçlerini de kapsayan bir “dokümante etme” sevdasının temel sebebi bu.

Dokümantasyon yalnızca yazılımcı değişikliği için değil elbette, lâkin bunu da ciddi şekilde destekliyor olduğunu yadsımamak gerekiyor. Elimizde yapılan işe dair ne kadar detaylı veri olursa, sonradan geliştirmelerle desteklemek o kadar kolay olacaktır.

Yeni yazılımcıya geçişin sancılı olabileceğini düşündüğüm adımlarını sırayla inceleyip her biri hakkında kimi ipuçları vermek isterim:

1- “Devam Edemiyoruz”

Mevcut yazılımcı işini muhteşem bir şekilde yapmış ve halen de yapmaya devam ediyor olabilir. Lâkin bu, onunla muhteşem ilişkiler geliştirebildiğiniz anlamına her zaman gelmiyor. Bir dönemde ister istemez, karşılıklı sebeplerden kaynaklı devam edememe durumuyla karşılaşmanız mümkün.

Bu tip bir durumdan mümkün olduğunca sancısız çıkabilmek için, tekrar tekrar söylüyorum, ortada yapılan işin bir dokümantasyonunun olması şart. Ardından, yapılan işlerin bütününün girişimin kendisine ait olduğunu da iş başlarken yazılım tarafına net olarak ilettiğinizi varsayıyorum, zira

yapmanız gerekenin bu olmasının nedenlerini geçtiğimiz bölümlerde anlattım.

Aynı zamanda, bütün ayrılık süreçlerinin sancılı süreçler olduğunu varsayarsak, yapmanız gereken, karşılıklı minimum hasarla olay yerinden ayrılmak olacaktır. Bunu yapabilmenin en iyi yöntemi de, sürecin hak edilen karşılığını net bir şekilde hak edene ödemektir. İlla para ödemesi yapmaktan bahsetmiyorum, lâkin ortada bir hak ediş varsa, onun karşılığının bir şekilde oluşturulabilmesi önemli.

2- "Yeni Biri Lazım"

Projenizin geliştirmesinde kilit rol alan yazılımcınızla yollarınızı ayırdınız. Şimdi karşılaşacağınız sorun, yeni bir yazılımcı bulma sorunu olacak. Bu noktada kritik başlıklardan bir tanesi, programlama dili seçerken bahsettiğim başlık olan "mümkün olduğunca fazla yazılımcısı olan bir dil seçme" kuralı.

Bu kurala uymadıysanız, örneğin Türkiye koşullarını değerlendirmeye katmadan "Go" veya "Python" kullandıysanız, işinizin biraz zor olacağını söyleyeyim. Ama yazılımcı kaynağında sorun çıkartmayacak bir dil seçimi yaptıysanız, süreç sizden yana işleyecektir.

3- "Ama O Kadar Yazdık"

Yazılımcılar kod yazmak isterler. Daha açık şekilde ifade etmek gerekirse, yazılımcılar kodu kendileri yazmak isterler. Başkalarının yazdığı koda koşullar ne olursa olsun belirli bir direnç oluşacağını ve mümkün olduğunca yeniden yazma yönüne kayılacağını belirteyim.

Bu noktada imdadımıza yine dokümantasyon yetişiyor. Yazılan koddan teknik olmayan biri ne kadar anlayabilirse, teknik olan biri de ilk bakışta o kadar anlayabilir, bunu unutmamak lazım.

Eğer doğru şekilde yazılmış bir kod tabanına sahipseniz, yeni gelecek yazılımcılarınızın sürece adapte olmakta sıkıntı yaşamayacaklarından emin olabilirsiniz.

Bir diğer başlık, bütün yazılımcıların kodun bütünüyle kendileri tarafından yazılmadığını bilmeleridir. Herkes zamanında başkaları tarafından yazılmış yan eklentiler veya sistemler kullanır. Bunları kullanmalarının sebebi de, başkaları tarafından yazılmış bu kod parçalarının işlerini net olarak görebileceğini düşünmeleridir.

Bu durum bir önceki yazılımcınızın yazdığı kod için de geçerli olduktan sonra, korkmanız gereken herhangi bir şey yok. Aksi durumda, gelsin maliyetler!

Belirtmem gereken bir diğer konu da, yeni gelecek yazılımcıların bir oryantasyon sürecine ihityaç duyacaklarıdır. Yazılımcının kalitesi ne olursa olsun, kendisinden önce yazılan koda alışması için bir sürecin geçmesi gerekir. Bu bir kalite değerlendirmesi başlığı değildir, gereklidir.

Bakım Anlaşması

Her proje her zaman yeni gelişmelere ihtiyaç duymayabilir. Nihayetinde duyacaktır, ama bu zaman ile ilgili bir durumdur, yani hemen olmak zorunda değildir. Böyle durumlarda, sürecin beklediğiniz gibi işliyor olmasını güvence altına almak gibi bir meseleyle karşı karşıya kalırsınız.

Bu noktada, mevcut yazılımcınızla ya da yeni bir yazılımcıyla bakım anlaşması yapmanız gerekir. Bakım anlaşması demek, yazılımcının sürecin beklendiği gibi işliyor olduğundan emin olması ve süreç içerisinde karşılaşılması muhtemel sorunlara çözüm üretmesi olarak anlaşılabilir.

Belirtmem gereken bir başlık da, bakım anlaşmasının yeni geliştirmeler içermek durumunda olmamasıdır. Hatta mümkünse yeni geliştirmeler içermemelidir. Bir girişimin iç dinamikleri düşünüldüğünde bu durum zor görünebilir, lâkin bakım anlaşmasını bu çerçevede tutmak uzun vadede yeni anlaşmaların da önünü açacağından faydalı olacaktır.

Bakım anlaşmasının doğru yapılması, sürecin düzgün işlemesi ve bir sorun çıkmaması açısından çok kritiktir. Bu yüzden, anlaşma içerisinde yapılması gereken bütün tanımlamalar tam ve net olarak yapılmalıdır.

Örneğin, yazılmış bir sistemin nasıl bir periyotla yedekleneceği, olası bir sunucu sorununda neler yapılacağı, olası herhangi bir sorun çıktığında ne kadar zaman içerisinde müdahale edileceği, bu müdahalenin kaç saatle sınırlandırılacağı ve belirlenen zaman aşıldığında nasıl bir politika izleneceği belirtilmezse, soruna müdahale etmek gerektiğinde herkes birbirini suçlamaya başlayacaktır.

Bu tip bir sorunla karşılaşmamak için, bakım anlaşması süreçlerinde profesyonel bir destek alınması faydalı olacaktır. Süreç ne kadar kesin işlerse, o kadar kesin sonuçlar verecektir.

Özellikle yazılım süreçlerinde, kesin olmayan her başlık tehlike arz eder. Bu her zaman bir sorun oluşturmayabileceği gibi, sorun oluşturduğu durumda birilerinin kesin olarak çözüme gidebilmesi gerekir. Bu yüzden bu bahsi geçen çözümü kimin, ne zaman ve hangi koşullarda üreteceğinin net bir şekilde saptanmış olması gerekir.

İşler iyi gidiyorken, kimse bakım anlaşmasını düşünmeyecektir. Bu da yapılan yanlışların başındadır. Özellikle bir girişim için, yani kurumsal yapıları tam oturmamış bir yapılanma için sürecin sonunda karşılaşılabilecek olası sorunların öngörülmesi önemlidir.

Olası sorunların işin başından itibaren değerlendirilmesi ve karşılaşılabilecek durumların analiz edilmesi gerekir. Ardından süreç tekrar iyi gitmeye devam edebilir.

Diyerek, sona geldik. Başta söylediğim gibi, bu kitap mümkün olduğunca bir referans kaynağı olmak ve adımları sıralayabilmek amacıyla yazıldı. Bütün başlıklarında en derin bilgilere erişebileceğiniz bir kaynak olarak organize olmaktan ziyade, o başlıkların neler içerdiğini net bir şekilde gösterebilme derdini taşıyor.

Bütün bu başlıklara hakim olduğunuz durumda hiçbir sorunla karşılaşmayacaksınız diyemem. Lâkin burada diyebileceğim, karşılaşmanız muhtemel olan sorunlar karşısında almanız gereken tavra dair bir yol gösterici olarak bu kitabı kullanabileceğiniz.

Ayrıca, kitap okumanın bir angarya olarak görülmeye başladığı günümüzde bu cümleye kadar gelebildiyseniz, memnun oldum, oturup bir kahve içmek isterim :)

Sonsöz

Kitabı ücretsiz olarak yayımlamamı sağlayan arkadaşlarıma tekrar teşekkür ederken, sizlerden bu ücretsiz kitabı okuyup beğendiyseniz çevrenizle paylaşmanızı rica edeceğim.

Türkiye girişimcilik ekosistemi olarak henüz ortaya yeterince ses getirecek işler çıkartamamış olabiliriz. Lâkin bu durum, asla bunu yapamayacağımız anlamına gelmiyor.

Yapmamız gereken, el birliği ile ülkemizden çıkabilecek girişimlere mümkün olduğunca destek olabilmek ve onları ileri taşımak için elimizden geleni ardımıza koymamak. Şahsım adına naçizane elimden gelebilecek herhangi bir desteği esirgemeyeceğimi buradan belirtmek isterim.

Son bölümde de dediğim gibi, eğer kitabı bu noktaya kadar okuduysanız, tebrik ve teşekkürlerimi sunar, bir kahve içmeye beklerim.

Ulaş Can Cengiz, Ağustos 2015, İstanbul

ulas@ulascengiz.com,

Twitter - @ulsc

Facebook - fb/ulascancengiz

LinkedIn - in/ulascengiz