Neden “programcının” kitabı değil?

Programcılık yazılım geliştirmenin en amele tarafı. Buna “programcının kitabı” demek müzisyenler için yazılan bir kitaba “notacının kitabı” demek kadar sığ olurdu. Halbuki bana “2500 yılında programlar nasıl yazılacak?” diye sorsanız “yazılmayacak” derdim. Tabi bir insana oturup 500 yıl sonrasını sorup bundan mantıklı bir yanıt beklediğiniz için sizinle görüşmeyi keser, telefonlarınıza çıkmayı da bırakırdım. İlk bilgisayarların mandallarla, sonra delikli kartlarla kodlanması gibi bugün klavye tuşlarına vura vura yazılım üretmek de gelecekte kaybolacak zanaatlerimizden. TRT kameramanı ve spikeri Erzurum’da yaşlı bir dedenin yanına gidecek ve “dedeciğim bize yaptığın işi anlatır mısın?” diyecekler. Dede de “işte ben bu tuşlara vuruyorum ve bu ekranda yazılar çıkıyor, böyle böyle program yazıyoruz evladım” diyecek. Türkiye’de kalan son üç programcıdan bahsedecekler.

Programcılık yazılım üretmek için ihtiyacınız olan son bilgi ama ne yazık ki ilk öğrenileni. Zira günümüzde yazılım geliştirmek için programlama dışında alternatif yöntemimiz yok. 90’lardan beri bu konuda çözümler geliştirilmeye çalışılıyor. IDE (Integrated Development Environment, Bütünleşik Geliştirme Ortamı), RAD (Rapid Application Development, Çabuk Uygulama Geliştirme) gereçleri bu konuda daimi bir çözüm arayışı içindeler ama ne yazık ki halen programlamadan tamamen kurtulabilmiş değiliz.

Programlama da atla deve bir şey değil. Çamaşır makinesini çalıştırmak bile başlı başına bir programlama eylemi ama hiçkimse görmüyoruz ki CV’sinde “Çamaşır Makinesi - Üst seviye (Kurutma, Sıkma)” yazıyor olsun. Programlama bilgisayara yaptırmak istediklerimizi bilgisayarın anlayacağı dilden anlatma eyleminden başka bir şey değil. Bilgisayarın da bu kadar karmaşık bir dil kullanıp bunda manasızca ısrar etmesini de anne ya da baba tarafından Fransız olma ihtimaline bağlıyorum.

Yazılım üretiminin programcılıktan geçiyor olması en büyük laneti de. Zira ürünle üretim teknikleri arasında manalı bir biçim yok. Yazılımı programcıya yaptırmak film senaryosunu kameramana yazdırmak gibi. Bunun ne kadar korkutucu sonuçları olabildiğini Nuri Bilge Ceylan örneğinden biliyoruz. Programcılık analitik ve teknik bir düşünce yapısı gerektirirken, yazılım insanla etkileşimli bir problem çözümünü temsil ediyor. Haliyle programcının analitik bakış açısı ister istemez çözümlere de yansıyor. O yüzden “Devingen aygıtar bağdaştırıcıyla

eşleştirilemedi” diye insanı çileden çıkartan ama programcı için gayet manalı hata mesajlarıyla, programcı için kodlaması kolay ama kullanıcı için kullanımı zor senaryolarla karşılaşabiliyorsunuz. İlla ki programcının kafasına vurmanıza gerekiyor “insan kullanacak onu insan!” diye.

9 yaşımdayken bilgisayarı ve bilgisayar programlarını ilk gördüğümde beni büyüleyen tek şey “bilgisayara istediğini yaptırabilmek” oldu. Programlama bunun için bir araçtı sadece. Onu öğrenmem gerektiğini biliyor ama onun kendisiyle ilgilenmiyordum. Eğer bilgisayarın benim istediğim şeyi yapması için zıplamam, takla atmam gerekiyor olsaydı onları yapacaktım. Zaman içinde akışın ters yönde olmasına engel olamadığım oldu. Mesela kullanıcı için kullanımı zor ama kodlaması kolay yöntemleri tercih ettim.

Bu kitap eğer hiç fikriniz yoksa yazılım geliştirmeyle alakalı size doğru bir perspektif sunmayı, fikriniz varsa da bakış açınızda sizi de memnun edecek iyileştirmeler yapmayı, kısaca sizi daha iyi bir yazılımcı yapmayı hedefliyor. Bitirdiğinizde daha iyi bir programcı olmayacaksınız ama belki olduğunuzdan daha iyi bir programcı olmanıza gerek olmadığını da keşfedeceksiniz.

Yazılım nedir?

Böyle lise müfredatından çıkma bir başlık gördüğümde oradan koşarak kaçasım geliyor. Genel olarak yazılım geliştirmeyle ilgili kitaplarda “önce şunu bir tanımlayalım” kısmı beni en sıkan en bayan kısımdır.

Böyle bir tanıma ihtiyaç olma sebebi programlama, tasarım ve her tür teknik bilginin bize yazılımın ne olduğu konusunda algımızı bükebilme potansiyeli. Mesela bir programcı için yazılım “belli koşullarda bilgisayarın yürütmesi beklenen direktifler bütünü” olabilecekken, bir grafik tasarımcı için “butonlar, ekranlar ve yazıların etkileşimi”, bir kullanıcı için “kullanmayı sadece Muharrem’in bildiği başbelası” olabilir.

Bütün bu insanların akşam iş çıkışından sonra gittikleri grup terapi seansında uzlaşmaları gereken tek bir tanım vardır o da yazılımın bir “çözüm” olduğudur. Aynen bizler her şeyden önce insan olduğumuz gibi, yazılım da her şeyden önce bir çözümdür.

Bunu kafanıza ne kadar nakşetseniz az zira zaman içinde yazılım sizin için illa ki başka şeylere dönüşmeye başlayacak ve illa ki yazılım üretiminin bir noktasında bir açmazla karşılaşacaksınız. O zaman “dur ya biz burada bir çözüm üretiyoruz alo” diye bu çatışmadan sıyrılabilecek, onu aşabilecek, bana bir teşekkür mektubu yazabilecek ve belki içine bir yerlere 2300 euro’luk bir çek sıkıştırabileceksiniz. Bu söylediğimi bir sonraki paragrafta tamamen unutacaksınız ve uyandığınızda kendinizi daha hafiflemiş hissedeceksiniz.

Aslında sadece bu bilgi, yani yazılımın her şeyden önce bir çözüm olduğu, tek başına kitabın tamamını sizin de yazmanızı sağlayabilir. Sadece bu bilgiyi temel alarak yazılımla ve yazılım geliştirmeyle alakalı bütün açmazları ve bütün bilinmeyenleri çıkarabilirsiniz.

Basit gibi görünüyor ama yazılımın “çözüm” olduğu gerçeği bize beraberinde pek çok bilgi veriyor aslında. Mesela bir çözüm olması için en başta ortada bir problem olması gerektiği gerçeğiyle yüzleşiyoruz. Haliyle yazılım tasarım sürecinin ilk aşaması olan “problemin tanımı”yla karşılaşıyoruz. Aa bak böyle düşüne düşüne her şey aslında çorap söküğü gibi gidiyor.

Problemin tanımı

Problemi net ve eksiksiz tanımlayabilmek aslında sadece yazılımcılığın değil hayatımızdaki her eylemin en temel çıkış noktası olmalı. Mesela bir sokak kavgasında bile problemi net tanımlamak önemlidir. O yüzden Amerikan filmlerindeki kavga girizgahı genelde “Dostum senin problemin ne biliyor musun? O kocaman beyaz kıçın” şeklinde olur. Bu gelecekte olabilecek yanlış anlaşılmaları engeller. Mesela en baştan kocaman beyaz kıçı olmayan biri sözün muhatabı olmadığını anlayabilir ve “Hey dostum sorun yok çocuklarla biraz eğleniyorduk o kadar” deyip çözümü en baştan üretebilir.

Yazılımda erken aşamalarda verilen kararlar en kritik kararlardır. Atacağınız bir okun yönünü atmadan önce ayarlamak gibi size en çok faydayı erken aşamalarda yapacağınız tercihler sağlar. Aynı şekilde yanlış tespit ve tercihler de sonrasında telafisi en güç sıkıntılara yol açabilir. Okun yönünü attıktan sonra koşup yetişip değiştirme çabası kadar zorlar ve olur da başarırsanız aynı derecede heyecanı öldürür.

Problemin tanımı tüm ekibin uzlaşması gereken bir tanım olmalı. Zira yazılım geliştirme sürecindeki bütün tartışmalar eninde sonunda “ya bizim çözmek istediğimiz problem bu değil ki” argümanıyla çözümlenecektir. Açıkçası yazılım ekibini en çok meşakkaten doğru yapılmış problem tespiti kurtaracaktır.

İlk başta garip geliyor kulağa tabi. “Oyun geliştiriyorsak hangi problemi çözüyoruz?” denebilir. Halbuki orada da bir problem çözüyorsun. En temelinde kullanıcının hoşça vakit geçirme ve eğlenme ihtiyacını çözüyorsun. Dolayısıyla problemin tanımı aynı zamanda yazılımın vizyonunu da teşkil ediyor.

Herhangi bir vizyona sahip olmayan ve sadece “şunun gibi olsun” diye geliştirilmiş çok yazılım gördüm. Bunlardaki en büyük sıkıntı temel bir çözüm hedeflemediklerinden başarısız taklitlere dönüşüyor olmaları. Bu her taklit başarısız olur anlamına gelmiyor ama bir çözümü incelerken hangi problemi çözdüğünü anlamadan yapılan taklitler illa ki başarısız oluyor. Problemi ve onun nasıl çözüleceğini idrak etmek çok önemli.

Problemin kendisine odaklanmak yazılım geliştirmeyle ilgili en önemli algı değişikliğini yaratıyor: yazılıma özellik eklemek yerine tam tersine problemi çözmeye yardımcı olmayan aksine zorlaştıran özellikleri çıkarmanın ya da hariç tutmanın da gücüne uyandırıyor.

Problem tespitini bir örnekten yapalım. Her programlama dilini anlatmaya başlayan kaynakta muhakkak bir “Hello World” (“Merhaba Dünya”) uygulaması olur. Bunun esprinin kaynağı da Brian Kernighan ve Dennis Ritchie’nin yazdığı C programlama dilinin ilk resmi kaynağında verilen ilk örnek program olması ve herkesin bu örneği takip ediyor olması.

Programın yaptığı şey şudur:

```
Hello, World!
```

Evet programın yaptığı tek şey bu. Ekrana koca beyaz bir Hello World yazmak. Buna bakınca bu yazılımın herhangi bir problem çözmeye çalışmadığını ya da tek probleminin o koca beyaz çıktısı olduğunu söyleyebiliriz. Öte yandan programın aslında problem tanımı şöyledir:

“C dilini hiç bilmeyen kimselere dilin temel yapılarını kabaca gösteren, başka dilleri bilen insanların anlayabileceği bir kod yaz”

Haliyle programın kendisi bir çözüm üretmez ama kaynak kodu bir çözüm üretir. Okuru dille tanıştırır, “ekrana yazma nasıl yapılıyor” hakkında basit bir fikir verir. Kitabın geri kalanında göreceklerine alıştırır. Nasıl bir cehennemin ortasına düştüğünü haber verir. Yok merak etmeyin aslında C’den nefret etmiyorum ama zorluğunu da inkar etmiyorum.

9 yaşımda ilk bilgisayar ve programla tanıştığımı söylemiştim. Haliyle hayatımda yazdığım ilk program 48KB hafızaya sahip o ufak Sinclair ZX Spectrum+’ın o koca beyaz ekranını şöyle doldurmuştu:

```
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
SEDAT
scroll?
```

Bunun çözdüğü problemin bir kitabı daha kalın göstermek olduğunu düşünebilirsiniz. Haklı da olursunuz. Öte yandan 9 yaşındaki Sedat için bunun çözdüğü problem, koca bir ego tatmini dışında, “döngü” kavramıyla tanıştırması idi. Zira programın kaynak kodu şundan ibaretti:

```
10 PRINT “SEDAT”
20 GO TO 10
```

Yani Türkçe anlatmak gerekirse

```
10 EKRANA “SEDAT” YAZ
20 10’A GİT
```

Haliyle anında bir programlama dilinin temel mekanikleriyle tanıştırmış oldu. Yönergeler, etiketler, döngüler, yazı veri tipi, sonsuz döngü, sonsuz döngüye girmiş bir programdan nasıl çıkılır, çıkmanın yolu bulunamazsa bilgisayar nasıl resetlenir, küfürler. Hepsi bu program sayesinde.

Şimdi anında BASIC programından örnek verdim diye bunu bir programlama kitabına dönüştüreceğimi sananlar çok yanılıyorlar. Hayır burada veri tiplerinden döngülerden bahsetmeyeceğim. Problemden bahsedeceğim, o koca beyaz problemden.

Problemi doğru tanımlamıyor olmak bize ne kaybettirir? Sadece zaman ama öyle böyle değil. 1993 yılında GenDiş adında Diş Hekimleri için muayenehane otomasyon programı yazmaya başlamıştık. Elimizde bir problem tanımı yoktu. Birileri bize “abi dişçilere program lazımmış” dedi. Ben de “tamam daha fazla konuşma şapşal” deyip parmağımı dudaklarına götürdüm. Programı yazmamız 2.5 yıl sürdü. Sonunda döneminin en gelişkin DOS tabanlı mouse destekli grafik kullanıcı arabirimine sahip, CD çalarlı, hesap makinalı, ekran koruyuculu, her tür yerel ağ ile uyumlu çalışan çoklu kullanıcı ve chat desteği, sihirbaz tabanlı kurulum tecrübesi, hatta aynı anda paralel çalışabilen komut satırına bile sahip olan fantastik bir yazılım elde ettik. Ama koskoca programda diş hekimleriyle ilgili sadece üç pencere vardı: hasta kartı, borç alacak takibi ve tedaviler.

Tedaviler penceresi sanat eseri gibiydi. Diş çekmek için kerpeteni alıp gerçekten dişin üstüne götürüp çektiğinizde işlem kayıtlarına “diş çekme” kaydı ekliyordu. Aynı şekilde köprü için elinizle ağız şekli üzerinde mouse’la köprü çiziyordunuz.

Ama hiçbirimiz bir diş hekimiyle konuşup çözmeye çalıştığımız problemi net tespit etmeye çalışmamıştık. Eğer bir diş hekimine gidip konuşsaydık bize en yaygın yaptığı işlemin hasta bilgilerini ve ödemelerini takip etmek olduğunu diğerlerini kağıtta da yapabildiğini söylerdi. Hatta belki kağıtta kullandığı formatı kullanmamız işine gelirdi. Elbette diş hekiminin, yani hedef kitlenin yazılımla ilgili kendi fikirleri de olur. Aslında eklediğimiz çoğu şey diş hekimlerinin bizden talep ettiği şeyler olmasına rağmen çok azı diş hekimlerinin temel problemlerini hedefliyordu.

Eğer biz tasarımımızı sadece diş hekiminin birinin fantazilerine bakarak değil tam olarak hangi problemi çözmeye çalıştığımıza odaklanarak belirleseydik programın ilk sürümünü sadece hasta kartı ve borç alacak takibiyle de 2.5 yıl yerine 2 ayda çıkarabilirdik. Hatta o dönemin popüler muhasebe paketleri gibi geri kalan işlevleri “paketler” olarak sunmamız mümkün olurdu. Bu bizi oldukça maliyetten kurtarırdı, programı çok ucuz satmamız sağlayabilirdi.

GenDiş yazılımı bittiğinde 3 adet sattı. Ben bunun üzerine GenDiş’te geliştirdiğimiz kullanıcı arabirimini koruyarak geri kalan her şeyi sökerek sadece hasta kartı ve borç alacak takibi yapılan bir hasta takip programı yaptım. İsmini Hasta 1.0 koydum ve fiyatını 5 dolar olarak belirledim. PC World Türkiye’ye incelemeleri için gönderdim. Program için ufak bir paragraf tanıtım yazısı çıktı. Onun üzerine Adapazarı’ndan Zeynel adında bir doktor beni aradı ve programı satın aldı. Ondan sonra arayan soran olmadı.

Kısacası satış rakamlarına bakarak 15 dakikalık eforla 2.5 yılda elde ettiğimiz satış rakamının üçte birini elde ettim. Eğer 2.5 yılda kodladığımız kütüphaneler olmasaydı da öyle bir şey yazmam 2 aydan fazla sürmezdi. 2 ayda problemi doğru tanımlanmış bir yazılım paketi çıkardığını düşünürsen 2.5 yılda farklı meslek dallarına yönelik, farklı problemleri çözen yaklaşık 10 yazılım paketi çıkarabiliyorsun aslında.

Bizim problemi anlamaz, tecrübesiz, şovmen yaklaşımımız çok süslü bir yazılım ve bir sene sonra batmış bir şirket elde etmemizi sağlamıştı.

O yüzden hangi problemin çözüldüğünü net masaya yatırmanın ve o manzarayla yüzleşmenin önemi çok kritik. Bu örnekte olduğu gibi “kullanıcıyı dinlemek” de tek başına yeterli değil. Kullanıcı size problemini doğru anlatmayabilir. Mesela Türkiye’de yazılım geliştirme standartları açısından meşhur bir söz vardır:

> “Yazıtipini biraz büyütelim”

Niye? Çünkü muhasebe departmanındaki Ahmet Abi okuyamıyor. Kullanıcı burada “yazıtipi büyüklüğü”nün doğrudan işletim sistemi ayarlarından değiştirilebilir bir şey olduğunu bilmediğinden “biraz büyütelim” talebiyle geliyor. Yazılımcının burada yapması gereken talepten problemi ayrıştırabilmek. Bu örnekte Ahmet Bey’in ileri derece miyop olduğunu anlamak ve doğru çözümü üretmek: Ayarlar -> Ekran -> Yazıtipi -> Büyük. Ya da Hastane -> Doktor -> Gözlük.

Bu çözüm ne zaman işe yaramaz? Yazılımcı, aslında üzerine vazife olmadığı halde programın yazı tipini varsayılan kullanmak yerine kendisi belirlediyse işletim sistemi onun boyutunu değiştiremez. Yani programcı olmayan bir problemi çözmeye çalışarak topuğuna da sıkmış olur.

O yüzden geliştirme ekibinin uzlaşacağı ilk şey gelecekte en az tartışacağı bir problem tanımı, bir gelecek vizyonu olmalıdır. Biri öbürünü otomatik olarak oluşturur.

Vizyon

Vizyon genel anlamda bir gelecek öngörüsüdür. Haliyle vizyon aslında bir durumu betimler. Mesela John Lennon’ın “herkesin barış içinde yaşadığı bir dünya” hayali bir vizyondur. Vizyon aynı zamanda bir problemin başarıyla çözülme testidir. Dolayısıyla bir problemi net olarak yazdığınızda bir vizyonu da aynı anda yazmış olursunuz aslında.

Tecrübe

Performans

“Opsiyonel” denen başbelası

“En iyi kod hiç yazılmamış olandır”

“Algoritmalar”

“Bir algoritmayı koda dökmek”

“Comment”ler

- Don’t repeat what code is already saying
- Use it sparingly and always when needed
- Use it when code fails to explain it
- ekşi (bkz: kod yorumu/@ssg)