Onur Özcan

NASIL AGİLE OLUNMAZ

Yalnızca Scrum ile değil; mühendislik,
otomasyon ve insana yatırımla dönüşüm rehberi.

sharpware

sharpware.co

Onur Özcan

Ben Onur Özcan; bankacılıktan e-ticarete, Garanti'den eBay'e uzanan 15 yılı aşkın serüvenimde, yazılım dünyasının mutfağında pek çok proje ile boğuştum. Şimdilerde **Sharpware**'in kurucu ortağı ve Bahçeşehir Üniversitesi'nde öğretim görevlisi olarak, sektörde "çeviklik" adı altında pazarlanan hayallerin değil, bizzat sahada işe yarayan gerçek mühendislik pratiklerinin peşindeyim. Burada okuyacaklarınız; süslü unvanlar, ezberlenmiş kılavuzlar veya içi boş sertifikalar değil; kan, ter ve tecrübeyle harmanlanmış, "karpuz projelerden" ve bürokrasiden bıkmış profesyonellere yönelik filtresiz gerçeklerdir. Hazırsanız, görüneni değil, olması gerekeni konuşmaya başlayalım.

1. Köprüleri Kurmadan Çeviklik Olmaz

İşin sahibini işi yapanlara -gerçekten- yaklaştırmadan çevik olmayı beklemeyin.

İşin sahibi,

- *“Benim çok işim var.”,*
- *“Ne istediğimi söyledim ya, benden daha ne bekliyorsunuz?”*
- *“Bu sorumluluk bana mevcut sorumluluklarımın yanında ilave sorumluluk olarak verildi, ancak bu kadar vakit ayırabiliyorum.”*
- *“Siz yapın, ben sonra UAT’de bakarım”*

gibi ifadeler kullanıyorsa ve siz bu durumu düzeltmek için hiçbir şey yapmıyorsanız çevik falan olamazsınız.

2. Müşteri Masada Değilse, Çeviklik Yolda Kalır

Çoğunlukla IT ekipleri önceliklendirmenin ne olduğunun ve bunun öneminin zaten farkındadır. Gidip müşteriyi doğru düzgün önceliklendirme yapmaya ikna etmediğiniz, her Sprint Genel Müdür'den, GMY'den, Dünya Başkanı'ndan araya işler sokuşturduğunuz sürece **bu iş yine olmayacak.**

Ne Yapmalı?

Takımlara gereksiz bir ton eğitim vermek yerine şunlara odaklanın:

- **Müşteriye önceliklendirme ve odağın önemini** iyice anlatın.
- Takımların odağını sürekli değiştirmenin y**an etkilerini** gösterin.
- Önceliklendirme yapma ve odağı koruma konularında onları ikna etmeye çalışın.

Müşteriye çevikliği anlatırken, *"bak bir çevik olalım her istediğiniz hemen yapılacak, işler 8 kat hızlanacak"* şeklinde ifadeler kullanıp,

mavi boncuk dağıtmayın. Bunun başarılabilmesi için kendisinin de bir şeyleri değiştirmesi gerektiğini, çevikliğin sadece IT'nin bazı pratikleri uygulamasından ibaret olmadığını öğrenmelerini sağlayın.

3. Kirli Kod ve Noel Baba Beklentisi

Elinizde kirli bir kod tabanı varsa, kapasitenizin tamamını müşteri gereksinimlerine ayırmayın.

- Her iterasyon kod tabanınızı biraz daha temiz hale getirmek için çaba harcayın.
- Az az da olsa kod kalitesini ve mimariyi düzeltin.

Kötü kalitenin üzerine kat çıkmaya devam ederseniz **daha hızlı gidemeyeceksiniz**. Çevik falan da olamayacaksınız. **Scrum falan sizi kurtarmayacak.** Kalite problemlerinizi şeffaf bir şekilde ortaya koyup, düzeltmek için efor harcayın. **Siz uyurken bir gece Noel Baba gelip kötü kodlarınızı düzeltmeyecek.** Bu konuda bir mucize beklemeyin.

4. Hızın Anahtarı: Üretimde Otomasyon

Makinaya yaptırabileceğiniz işleri insanlara yaptırdığınız sürece değişime karşı hızlı adapte olamazsınız. İnsanlar makinalara göre yavaştır ve hata yapma (rework'e neden olup, maliyeti yükseltme) olasılığı yüksektir.

Hız istiyorsanız kaliteyi korumanız (yüksek teknik borç taşımamanız) ve fikirden çalışan çözüme ulaştığınız üretim sürecinizde uygun olan şeyleri insan yerine makinalara yaptırmanız gerekir.

- **Zor Olan:** Yazılım üretimi sürecinde otomatikleştirilmesi en zor olan kısım gereksinimlerle ilgili çalışmalardır. “Makina bana şu projenin gereksinimlerini üretiversin” demek en azından bugün çok mümkün değil. Bu kısım hala yoğun insan emeği gerektiriyor ve gereksinimlerle ilgili çalışmaların maliyetinden yırtma şansımız pek yok.
- **Mümkün Olan:** Ancak dağıtım ve test gibi aktiviteleri -tümü olmasa bile büyük bir bölümünü- otomatikleştirmek mümkün ve çok

da zor değil. Hatta kod üretimini bile belli koşullarda belli araçlarla otomatikleştirmek söz konusu.

O zaman ne yapıyoruz?

Hızlı gidebilmek için üretim sürecimizde otomasyonu arttırıyoruz. Gerçek çeviklik için muhtemelen bir ordu büyüklüğünde olan ve işleri tekrar tekrar test datası hazırlayıp ekranların sağına soluna tıklamak olan test ekiplerinizi **"otomasyon" konusunda eğiterek** işe başlayabilirsiniz.

5. "Light" Yöntemler ve Mühendislik Gerçeği

Çeviklik sadece Scrum gibi proje yönetimine odaklanan **light yöntemler** kullanılarak elde edilebilecek bir şey değil. Kendinizi ve takımlarınızı kandırmayın. Scrum, Kanban gibi yöntemleri **iyi mühendislikle beslemediğiniz**, işi yapan insanların yetkinliklerini geliştirmek için yatırım yapmadığınız sürece 2 haftalık Sprintlerinizin sonunda ancak *"yine doğru düzgün bir şey teslim edemedik"* dersiniz.

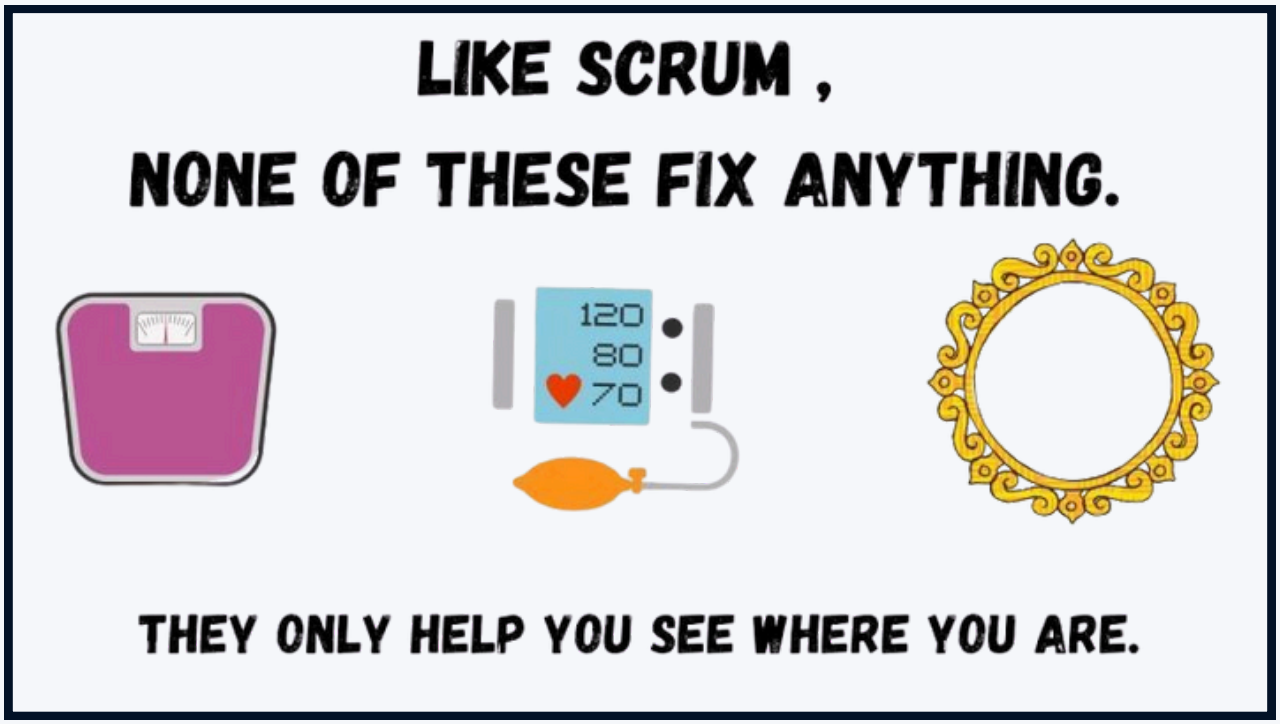
İnsanların gelişimlerine destek olun. Çalışanlarınız için **20-30 dolarlık birkaç kitap** ve/veya e-learning içeriklerine yatırım yapmaktan çekinmeyin. Bu paralar hiçbir şirketi batırmamış veya zengin etmemiştir. Şirketinizde 10 dakika turlasanız bunun kat ve katı israf görürsünüz, onları azaltıp, çalışanların gelişimine yatırım yapın. **İşi yapan insanların gelişimi** en az Scrum Master veya Product Owner'ın gelişimi kadar önemlidir.

Danışmanlar size bu gerçekleri söylemek yerine:

- *"Scrum master'ları geliştirmeye odaklanalım,"*
- *"Product owner'lar için bir kamp yapalım,"*
- *"Buradaki asıl problem takımlar arası dependency yönetimi"*

falan diyip 1-2 senenizi heba ederler. Şenliklerle, çarşaf çarşaf CEO mailleri ve *"yaşasın biz de agile oluyoruz"* temalı LinkedIn postları ile başlayan agile dönüşüm insiyatiflerinizin **mutsuz son ile bitmemesi** için benden söylemesi.

6. Liderlik ve Halının Altındaki Engeller



Sevgili dostum Hamdi Küçük paylaşmıştı bu görseli daha önce.

Bir çevik dönüşümde takımlara eğitim aldırıp Scrum veya Kanban kullanmaya başlamalarını sağlayan şirketler dönüşümün %95'ini hallettiklerini zannediyorlar. **Bu yöntemler sizin yerinize problemlerinizi fixleyecek sihirli değnekler değiller.** Asıl iş bu yöntemleri kullanmaya başladıktan sonra farkına varılan engelleri ortadan kaldırabilmek veya minimize edebilmek.

Özellikle şirketlerin liderlerine çok fazla iş düşer bu aşamada. Takımlar veya bir Scrum Master size bir engelden bahsettiğinde bunu halının altına süpürürseniz, **Scrum'ın yapabileceği tek şey o engelin hala orada olduğunu size göstermeye devam etmektir.** Sevgili liderler, yöneticiler; **elinizi kirletmeden, takımların sorunları / engelleri ile ilgilenmeden Scrum'ın bir mucize yaratmasını beklemeyin.** Sadece masanızda oturup *"işte bizim çocuklar da eğitim aldılar, 2 haftalık Sprintler yapıyorlar"* dediğinizde çalışanlarınız da dönüşüme olan inancınızı ve samimiyetinizi sorgulamaya başlayacaktır.

Takımlarınıza kulak verin ve onlar için mücadele edin.

7. Gereksinimlerin Metodolojisi Olmaz

Her ne kadar "Agile Software Requirements" isimli kitaplar/makaleler olsa da bir Agile projenin gereksinimleri diğer yazılım geliştirme yaşam döngüsü modellerini takip eden projelerden **niteliksel olarak farklı değildir.**

Software Requirement'ın Agile'ı Waterfall'ı olmaz. Bununla birlikte, çevik ve geleneksel projeler, özellikle gereksinimlerle ilgili çalışmaların **zamanlaması ve derinliği** ile gereksinimlerin **dokümantasyon şekli** açısından elbette farklılıklar içerir.

Hangi SDLC modelini kullanırsanız kullanın tüm projelerde doğru işlevselliği doğru şekilde geliştirebilmek için işi yapacak geliştiricilerin önüne aynı bilgileri/detayları koymanız gerekir. İster **user story formatı** ile ifade edin, ister **use case tekniğini** kullanın, isterseniz beyaz bir tahtada görseller çizin **işi yapacak insanlar "ne" üreteceklerini iyi anlamalıdırlar**. Agile Coachlarınıza veya Scrum Masterlarınıza boş beleş eğitimler aldırana kadar tüm takım üyelerine **Yazılım Gereksinimleri, Görsel Analiz Modelleri (UML vb.), Prototipleme Araçları** gibi konularda eğitimler aldırın.

8. Kod Yazmadan Değişebilmek

Değişen müşteri gereksinimlerine hızlı bir şekilde adapte olabilmek istiyorsanız, **kolaylıkla değiştirilebilir uygulamalara** sahip olmalısınız. Müşterinizden gelen her gereksinim için kod yazıp, test yapmak zorunda kalıyorsanız çeviklik konusunda epeyce bir yolunuz var demektir.

Uygulamalarınızın en sık change alan bölümlerini herhangi bir kod geliştirmenize ve test yapmanıza gerek kalmadan değiştirmenize imkan veren yapılar tasarlayın. **Imperative yapılar yerine Domain-Specific Language / Rule Engine gibi Declarative yapılar** kullanmanız değişiklik taleplerini minimum maliyetle karşılamanızı sağlayarak çevikliğinizi arttırır.

He tabi bunları tasarlamak ve geliştirmek tecrübe ve emek ister. Bu yollar biraz daha **kan, ter, gözyaşı gerektirir.**

Bunlarla uğraşmak yerine;

> "Çok keyifli bir Retrospective tekniği denedik, tüm takım üyelerinin çocukluğuna indik. Meğer bizim Ahmet'in ilkokul öğretmeni eline cetvelle vurmuş ondan bu kadar agresifmiş, bu Sprint bunu öğrendik."

şeklinde de devam edebilirsiniz çevik dönüşümünüze.

9. Danışmanlık ve ”Hands-on” Gerçeği

Eğer çevik dönüşüm için bir danışmanlık desteği alıyorsanız **danışmanları oyunun içerisine - gerçekten- dahil edin.** Sadece takımlarınıza ve yönetiminize "ne" yapılması gerektiğini söyleyip, kendisi hiç bir işe elini sürmeyen danışmanlarla çevik dönüşüm falan yapamazsınız.

Sınıf eğitimlerinde size ideal kurgular üzerinden product backlog oluşturmayı, sıralamayı anlatan arkadaşların çoğu gerçek bir projede *"hadi gel beraber bir product backlog oluşturalım"* dediğinizde kalem oynatamazlar. Size sadece

kitabi doğruları söyleyen arkadaşlar çoğu zaman söylediklerini herhangi bir şirkete uygulamış durumda değiller.

> **Acı Gerçek:** Takımlarınızla birlikte -**hands on**- iş yapmayan, sadece size ahkam kesen danışmanların tek faydası şirket giderlerinizi arttırıp daha az vergi ödemenizi sağlamalarıdır.

Bir de hayatında bir kez dahi bir yazılım projesinin içerisinde bulunmamış, kariyerini İK, satış gibi fonksiyonlarda geçirmiş danışmanların yazılım takımlarınıza fayda sağlama olasılığı yoktur. Faydanın aksine, işi gerçekten yapan insanlara işin nasıl yapılması gerektiği konusunda bol miktarda ahkam kesip canlarını sıkarlar, üretkenliklerini düşürürler.

Takımlarınız, insanlarınız bu kadar değersiz değil. Önünüze danışman diye koyulan her insanı kabul etmeyin.

10. Sertifika Fetişizmi ve Gerçek Dönüşüm

Sevgili şirketler/teknoloji emekçileri, bir konuda sizi uyarmak istiyorum. Şirketinizdeki herkese;

- **PSM, PSPO, PSD, SPS,**
- **PAL, PAL-E, PSK, PSU,**
- **CSM, CSPO, PMI-ACP**

sertifikalarını aldırdığınızda **dönüşmüş olmuyorsunuz.** Linkedin'de boy boy paylaştığınız sertifikalar günün sonunda işi yapmıyor. **Kağıt parçalarına odaklandığınız kadar işinizi nasıl daha iyi yapacağınıza odaklansanız** projelerinizin başarılı olma şansı artar. Sertifikayı veren kurumlar basılı halini göndermeye bile tenezzül etmiyor, pdf üretip üzerine isminizi yazıyor, ne kadar değerli olduğunu buradan anlayabilirsiniz.

11. Dönüşümün ”Fake” Yüzü ve Sayıların Sessizliği

Bu platformda bol bol anlatılan çevik dönüşüm başarılarının içeriğini ben şöyle okuyorum: Genellikle ilgili şirketin üst yöneticilerinden biri tarafından sayfalarca yazılmış ve "öyle dönüştük, böyle dönüştük, bir dönüştük ki parmaklarını yersin" temalı metin içerisinde **herhangi bir sayı var mı?** Evet evet, bildiğin sayı.

Yaw güzel kardeşim anladık dönüştün de, bunun nasıl bir faydası oldu? Sen onu bir anlatsana bize.

- *"Çalışan memnuniyeti ve bağlılığı çok arttı"* demişsin mesela, ne kadar çok?

- Çok mu çok, az mı çok? Kime göre çok?

Yazsana oraya ölçtük, şu kadar artmış diye, biz de görelim çok mu. Ayrıca, madem çalışan memnuniyeti bu kadar arttı neden bu kadar açık pozisyon var şirketinizde? Her sene %30 büyümediğinizi hepimiz biliyoruz.

Ya da desene *"Çevik dönüşüm sonrası lead time'ı şuradan şuraya getirdik"* diye. Biz de alkışlayalım. Diyemiyorlar, neden diyemiyorlar? Çünkü **dönüşüm adı altında yapılan işlerin büyük bir bölümü fake!**

"Danışmanlık şirketine bir ton para verdik, karşılığında pek bir şey elde edemedik" diyemiyorlar, *"Çok güzel dönüştük, pek de güzel dönüştük."* diyorlar.

Siz devam böyle dönüşmeye, aferin!

12. Etkinliklerin Ötesinde Çeviklik

Sevgili takımlar sadece 2 haftalık Sprintler koşup, Günlük Scrum yaparak agile olamazsınız. Çeviklik bu kadar basit bir şey değil. Konu hakkında

yazılmış rasgele 2 tane kitap okursanız, çeviklik namına yaptığınız şeylerin **olması gerekenin %1'i bile olmadığını** görürsünüz. Ve sevgili agile coach'lar, sadece takımlarınızın Planlama, Review gibi etkinliklerine katılıp tuttuğunuz Excel dosyasına:

- *"Ali Ahmet'e laf soktu,"*
- *"Osman Ayşe'nin saçını çekti,"*
- *"Product Owner Saim toplantıya 8 dakika geç katıldı."*

şeklinde notlar tutarak takımlarınızı / şirketinizi çevikleştirmeniz mümkün değil. Takım üyelerine *"sen de haklısın, seni de anlıyorum"* diyip, dile getirdikleri problemler hakkında **hiç bir şey yapmıyorsanız bence boşa maaş alıyorsunuz**. Bu problemleri çözmek için gerekirse yönetiminizle çatışmalısınız. Yönetiminizle çatışacak gücünüz yoksa, bir genel müdür yardımcısına *"hayır, böyle olmamalı"* diyemiyorsanız zaten şirketinizde dönüşüm adına henüz hiç bir şey başaramamışsınız demektir.

13. SDLC ve Gereksinim Gerçeği

Çevik yöntemleri kullanırken gereksinimlerle ilgili yapmanız gereken çalışmalar kullanıcı gereksinimlerini **User Story formatı ile ifade etmekten ibaret değil.**

Hangi SDLC modelini kullanıyor olursanız olun, **yazılım gereksinimleri konusunda iyi iş çıkartmalısınız.** Gereksinimleri düzgün bir şekilde ele almadığınızda **rework maliyetleri** ile karşılaşma olasılığınız çok yüksek.

Scrum da kullansanız, V-model veya klasik waterfall ile de uygulama geliştirseniz kodu yazacak veya test edecek kişilerin önüne koymanız gereken **bilgiler / detaylar aynı.**

"Müşteri ne istediğini şu peçeteye 1 cümle ile yazdı, nasılsa Scrum yapıyoruz ya, Sprint'e bir başlayalım da Sprint'in içinde işi detaylandırırız" yaklaşımı ile geliştirdiğiniz uygulamadan hayır gelmez.

14. Danışmanlık Pastası ve ”0” Tecrübe

Çevikliğin popüler olması ile beraber pastadan payını almak isteyen bütün “büyük” danışmanlık şirketleri şimdilerde Çeviklik Danışmanlığı’na soyunuyor. Kitaplardan veya katıldığınız 2 günlük eğitimlerden öğrendiğiniz teorik bilgiler dışında danışmanlarınızın tecrübesi ne kadar? Kaç tane projede bu yaklaşımları -gerçekten- kullandınız mesela?

Cevap: 0

Diğer yandan 10 küsür senedir çeviklik danışmanlığı adı altına şirketlerin IT departmanlarına hiç bir değer katmamış “Çeviklik” danışmanlık şirketleri kendilerine yeni pazarlar yaratmak için İnsan Kaynakları, Pazarlama, Satış, Tedarik ve Satın Alma vb. fonksiyonlara **“Teknoloji ekiplerinizi çevikleştirdik, sizi de çevikleştirelim"** vaadi ile eğitim-danışmanlık hizmeti satmaya çalışıyor.

Siz hayatınızda kaç tane performans süreci kurup yönettiniz veya kaç tane şirkette satın alma

süreçlerinin içerisinde bulundunuz mesela?

Cevap: 0

Arkadaş bir durun ya...

15. Ali Bey ve ”Dutluk” Dönüşümü

Hikayemizin kahramanı Ali Bey... Ali Bey A şirketinden B şirketine transfer olur. B şirketinde bir an evvel kendini ispatlamak ve patronun gözüne girmek istemektedir. Bunun için B şirketindeki çalışanlar ile yaşadıkları problemler hakkında konuşup, yapısal birtakım değişiklikler yapmak yerine kolay yoldan gitmeyi tercih eder ve;

> *“Aa siz hala Agile’a geçmediniz mi, bu çağda Agile olmayan şirket mi kaldı, hemen Agile’a geçiyoruz”* der. Bunun için muhtemelen A şirketindeyken birlikte çalıştığı ve kendisine arka çıkacağı garanti olan bir danışmanlık şirketi ile anlaşır.

32 sınıf eğitim planlanır ve B şirketinin bütün takımları 3-4 ay içerisinde Scrum kullanmaya başlar. Retrospective'lerde takımlar sürekli aynı problemlerden şikayet etseler de, danışmanlık

şirketinin de gazıyla içerde **“Şahane dönüştük”** hikayeleri yüksek sesle anlatılır.

Takımlar *“Yaw sprint’e aldığımız bir PBI için 7500 satır stored prosedürün içerisinde değiştireceğimiz satırı bulana kadar sprint bitiyor.”* diye feryat etseler de Ali Bey bunları pek duymaz ve patrona “Ben yokken buralar hep dutluktu, ben geldim Agile olduk” diye hikayeler anlatarak koltuğunu 2 sene daha sağlama alır.

1-2 Linkedin postu, bir de konferans/meetup konuşması yaptı mı Ali Bey’den kralı yoktur.

16. Spotify Modeli mi, Müşteri Değeri mi?

Çevik dönüşümü illa organizasyonel dönüşüme bağlamak zorunda değilsiniz. Çeviklik namına size sadece Scrum anlatan danışmanlar 5-10 günün sonunda ellerinde malzeme kalmayınca konuyu hemen "**çevik organizasyon**"a getirir;

- Chapter,
- Tribe,
- Guild

ve buna benzer bir ton kavramla kafanızı karıştırır, bir şeyleri ıskaladığınız hissiyatı yaratırlar.

Bu meselenin özünde **müşteriyi merkeze koymak** vardır. Natamam çözümlerden öğrenmek vardır. "Benim dediğim doğru" diye diretmek yerine veriye bakıp, *"müşteri bunu istiyor, bunu üretelim"* diyebilmek vardır. Karşılanmamış her müşteri ihtiyacını kendine dert edinmek vardır. Bu "n" farklı organizasyonel yapılanma ile yapılabilir. İlla Spotify'ın modelini kopyalamanıza gerek yok.

> **Bir Telekom Operatörü Hikayesi:** Misal, adını zikretmeyeceğim bir telekom operatörü her fırsatta çıkıp tribe'lardan, chapter'lardan bahsedip, nasıl dönüştüğü konusunda ahkam kesiyor. Gelgelelim ben bu operatörün mobil uygulamasından son faturamı pdf olarak download edemiyorum.

Siz tribe'lara, chapter'lara kafa yormaya devam edin. Dönüşümü neden yaptığınızı unutup, **dönüşüm için dönüşüm** yapıyorsunuz. Olay dönüşümü müşteri için yapabilmekte... Dönüşüme başladığınız tarihten bugüne pazar payınız, toplam abone sayınız nereden nereye gelmiş ona da bir bakın...

17. Eğitimde ”Gamify” Tuzağı ve Ciddiyet

Çevik dönüşüm kapsamındaki eğitimlerinizi illa **"Gamify" etmenize gerek yok**. Evet biliyorum böyle isimler, içerikler havalı görünüyor ancak bu eğitimlere katılan insanların **"yetişkinler"** olduğunu düşünürsek onlara bir şeyler öğretmek için illa oyun oynatmanız gerekmez. Eğitimi alan insanların çoğunluğu **mühendislik formasyonuna sahip**, muhtemelen 2 üniversiteden mezun veya yüksek lisans derecesine sahip profesyoneller.

Doğru düzgün bir içerikle meseleyi anlamalarını sağlamak yerine 2 günlük eğitimin yarısında **geyik oyunlar** ile çalışanlarınızı anca eğlendirirsiniz.

Şimdi kalkıp bana ama hocam "uygulama" da önemli diyenler olur. Bu işin uygulaması saçmasapan **lego simülasyonları veya lastik top oyunları ile olmaz**. Çeviklik gibi derinliği olan bir konsepti "lego" simülasyonu gibi basit bir şekilde anlatarak hem **katılımcılarınızın zekasını küçümsüyorsunuz** hem de konuyu sığ gösteriyorsunuz. 2 günlük eğitimde bütün olayı timeboxed iterasyonlarla çalışmadan ibaret

göstermek bu konuyu çok hafife aldığınıza işaret eder.

18. Gereksinimler Bir Ekip İşidir

Agile yöntemleri kullanırken gereksinimlerin ortaya çıkarılması, analizi, ifade edilmesi ve gözden geçirilmesi; ürün sahibi, analist, kilit paydaşlar ve geliştiricilerin işbirliğine dayalı bir ekip sürecidir.

- Bazen PO olarak bir iş analisti konumlanabilirken, bazen PO'nun kullanıcı hikayelerini önceliklendirmesine ve detaylandırmasına yardımcı olmak için takım içerisinde bir iş analisti bulunabilir.

- Her iki durumda da diğer kilit paydaşlar ve takım üyeleri de "Refinement" (veya adına her ne diyorsanız) aktivitelerine dahil olmalı ve işi talep edenlerle ile işi yapacaklar arasında "ortak bir anlayış" oluşması sağlanmalıdır.

Halen yazılım geliştiriciler "analist veya PO yazsın önüme koysun", paydaşlar "UAT veya Review dışında bizden bir katılım beklemeyin" kafasındaysa daha çok yolunuz var demektir

Organizasyonunuzdaki insanlar *"birbirleri için doküman üretmek"* yerine *"birlikte çalışarak değer üretmeyi"* öğrenmeli.

19. Takımın Yakıtı: Amaç ve Etki

Bir takımın motivasyonunu düşürmenin en iyi yolu nedir?

Yaptıklarının (geliştirdikleri ürünün) müşterilerinizi nasıl etkilediğini görmelerini engelleyin. Aynı şekilde, geliştirdikleri ürünün şirketin vizyonu ve hedefleri ile nasıl bir ilişki içinde olduğunu bilmemeleri de takım için motivasyon kırıcıdır.

Üretilen değer ve amaç ile ilgili bu şeffaflık ve hizalanma sorunları, bir takımının kendisine organizasyonel hedeflere hizmet eden net hedefler belirlemesini zorlaştırır.

- Bu zorluk, sprint/iterasyon hedeflerine kadar sirayet eder.

- Sprint hedefleri olmadan takımlarınızda aciliyet hissi uyandıramazsınız ve onlara ilham veremezsiniz.

Kimin kaç saat efor harcadığından önce takımlarınızın yarattığı etkiyi onlara görünür kılın ve organizasyonunuzun hedeflerini tüm takımların bildiğinden emin olun.

20. Sabit Süre, Sabit Bütçe, Değişken Kapsam?

Müşteri / iş birimi (adına ne dersen) bir proje yaptıracak. Tam ne istediğini bilmiyor. (Yeri gelmişken şunu da söyleyelim: Müşteri tam ne istediğini bilmek zorunda değil)

Bir yerlerden de duymuş Agile diye bir şey var istediğin şekilde değiştirebiliyorsun gereksinimleri.

"*Heh, tamam işte, ne güzel, bu tam bana göre. Bizim proje de Agile olsun*" diyor.

Agile olsun ama gelgelelim süresi belli olsun, sabit olsun. Yeter mi yetmez. Parası / bütçesi de belli olsun, "*bana sonradan bu istediğin CR'dır, buna ek para, buna da ek süre*" deme diyor.

Yani, projenin kapsamı değişime tabi ama süresi ve parası değişime tabi değil.

Sevgili müşteri(ler) / iş birim(ler)i, sen bu ikisini de sabitlersen, projede değişim olunca bunun sonucu ne olur? Ben size söyleyeyim, **işi yapacak firma / takım fazla mesai yapsın, gecesini gündüzüne katsın, parasını da almasın.**

Özetle, **çeviklikten anladığınız:** “Çevik olalım dediysem, işi yapacak takım / firma sen çevik ol demek istedim!”

Aslında, **çeviklikten anlamanız gereken:** “Projede değişim her zaman olabilir. Çevik olmak istiyorsanız işi yapacak takımın değişimin maliyetini minimuma indirmek için doğru üretim pratiklerini kullanmasına, buna efor harcamasına izin vermeniz gerekir. Değişimin maliyeti kendi kendine düşmez, bunun için takımın çaba harcaması gerekir, kaliteli üretim yapması gerekir. Bunun yanında çeviklik için takımın üretim pratiklerini değiştirmesi yetmez. İşin sahibi olarak sizin de yaklaşımınızı değiştirmeniz gerekir. Doğru üretim pratikleri ile değişimin maliyetini ne kadar düşürmeye çalışırsanız çalışın değişim hiçbir zaman sıfır maliyetle gerçekleşecek bir şey değildir. Gereksinimlerde değişebilmeyi istiyorsanız bunun zaman ve para maliyetine de

katlanmalısınız. Ya da istediğiniz değişimin gerçekten gerekli olup olmadığını / faydasını daha iyi sorgulamayı öğrenmelisiniz. Bazen *“madem maliyeti bu kadar hepsini yapmayalım (descoping), yarısını yapalım ”* demeyi öğrenmelisiniz.”

21. İş Sonuçları İçin IT’nin Ötesine Geçmek

İyi çalışan ve sürekli teslimat yapan bir IT tek başına başarılı iş sonuçları için yeterli değildir. Başka bir deyişle, sürekli teslimat ve Agile/DevOps vb. tüm insiyatifler, iş sonuçlarını iyileştirmiyorsa önemsizdir.

İş sonuçları, yalnızca IT içerisindeki geliştirme ve operasyon ekiplerinin birlikte daha iyi çalışmasını sağlayarak gelişmez. Tek başına IT ekiplerini çevikleştirerek organizasyonu çevikleştiremezseniz.

Çevikliğin işe yaraması ve iş sonuçlarını iyileştirmesi için tüm birimlerin birlikte daha iyi çalışması gerekir. Her birimin tek başına düzgün çalışması gereklidir ancak yeterli değildir. **Birimler arasındaki etkileşimleri geliştirmek, organizasyonel çevikliği artırır.**

22. Uçtan Uca Çevik Değer Zinciri Yönetimi

İş fonksiyonları ve BT iyi bir şekilde işbirliği yaptığında, ürün veya çözüm konseptinden nakde (veya şirket içinde ürünün/çözümün benimsenmesine) kadar ödüllendirici bir yolculukla sonuçlanır.

Bu yolculuk, **pazardan ve değişen teknoloji ortamından kaynaklanan zorluklar**la karşı karşıya kalabilir, ancak gerçekten bir Çevik dönüşüm geçirdiyseniz iç faktörler nedeniyle hayal kırıklıkları ve gecikmelerden uzak olacaktır.

Bu yolculukta doğasında var olan zorluklarla başa çıkmak için oldukça iyi anlaşılmış yöntemlere sahibiz. Ancak Çevik dönüşüm diye adlandırılan insiyatiflerin büyük bir çoğunluğu maalesef bu yöntemlerin pek çoğunu es geçiyor.

- Yalın ürün keşif teknikleri yolculuğun ilk kilometrelerinde size yardımcı olur.

- Benzer şekilde sürekli teslimat ve DevOps, yolculuğun son kilometrelerinde size yardımcı

olur.

- Mühendislik pratiklerini de içeren çevik yazılım geliştirme, aradaki kilometreleri katetmek için neredeyse ana akım haline geldi.

Görüldüğü gibi işin **doğal zorluklarını aşmak için bir arada kullanılması gereken çok fazla yöntem/pratik var.** Bu nedenle çevik dönüşümü Scrum uygulamaktan ibaret zannetmek büyük bir hata. Gerçek bir dönüşüm çok daha fazlasını gerektiriyor ve maalesef bu pek çok kurum için zannedildiği gibi 3-5 sınıf eğitimi ve bir süre alınacak danışmanlık ile 6 ayda gerçekleşebilecek bir şey değil.

Dahası işin doğal zorlukları dışında, organizasyonel zorluklar da var. Bu organizasyonel zorluklar, yolculuğun her kilometresine nüfuz ederek ve tüm dönüşüm çabalarının başarısını tehdit edebilir.

23. Çevik Dönüşümde Mühendislik Prensipleri

Bazı Çevik(Agile) fanatikleri (çoğunlukla konu hakkında son derece yüzeysel bilgisi olan arkadaşlar), önceki adımlara dayanan ardışık bir süreç fikrine şiddetle karşı çıktıkları için waterfall gibi modellere saldırmayı severler.

Zannediyorlar ki, **çeviklik olayın içine balıklama atlamak** ve bir şeyleri direk kodlamaya başlamak anlamına geliyor.

Çevik yöntemleri kullanırken de halen "define before design" ve "design before code" dediğimiz prensipleri işletmelisiniz.

Çeviklik gereksinimler veya tasarımla ilgili faaliyetleri üstünkörü yapmayı değil; bunları sadece daha küçük adımlarla yapmayı savunuyor.

Sevgili hocam Yaşar Safkan'ın ekteki blog postunda dediği gibi, Çevik olmaya çalışan sevgili takımlara verilecek tavsiye şudur ki:

"Bu ahval ve şeraitte dahi, birinci vazifen, kodunu tasarımsız yazmamak, kodun mimari yapısını iç ve dış tüm baskılara karşı korumaktır. Muhtaç olduğun kudret, ellerinin altındaki klavyede mevcuttur."

24. Çeviklik İllüzyonu

- Testi projenin sonuna bırakarak,
- Entegrasyonu *"ayın sonunda yaparız ya"* diyerek,
- Kodlama başlayınca gereksinimlerle ilgili çalışmaları durdurup, geri bildirim almaktan vazgeçerek

çevik olamazsınız.

25. Sürekli Kod Gözden Geçirme Kültürü

Takım üyelerini birbirinden izole etmek tehlikelidir.

Daha fazla çıktı vermek uğruna geliştiricilerinizin birbirlerinden tamamen yalıtılmış olarak kod yazmasına izin vermeyin.

Takımdaki herkes, diğer arkadaşlarının yazdığı kodları okumak için zaman ayırırlarsa, kodun okunabilir, anlaşılır olduğundan ve kaliteyi düşüren keyfi kod yazmadıklarından emin olabilirler.

Kodu ne kadar sık okursanız o kadar iyidir. Devam eden bu kod gözden geçirmeleri, yalnızca kodun anlaşılır olmasına yardımcı olmakla kalmaz, aynı zamanda **hataların da tespit edilmesini sağlar.**

26. Hata Aramaktan ”Öğrenen Organizasyon” Yapısına

Takımınız bir sorunla karşılaştığında ilk yaptığı şey nedir?

Bu durumlarda soruna neden olan kişiyi, "suçluyu" bulmak ilk önceliğiniz olmalı, değil mi?

Cevap: Hayır!

Dönüşüm için sarf ettiğiniz tüm çabalar sonucunda halen soruna kimin neden olduğunu belirlemek ilk önceliğiniz ise çevikliği anlamamışsınız demektir. Her zaman eldeki sorunu çözmek en önemli önceliktir.

Takımlarınızın bu tarz durumlarda verdiği ilk tepkinin ne olduğunu dikkatlice inceleyin. Sorunun çözümüne hiç bir katkı sağlamayan **"günah keçisini" mi arıyorsunuz**? Yoksa eldeki problemi bir an evvel nasıl çözebileceğinizi mi tartışıyorsunuz? Elbette, problemi çözdükten sonra problemin tekrar yaşanmaması için neler yapılabileceğini de masaya yatırmalısınız.

Gerçek çeviklik sadece hazır yöntemleri uygulayarak elde edilmez. Takım üyelerinin tutum ve davranış değişikliği ile dönüşebilirsiniz. Zor olan kısım burasıdır.

27. Duvarları Değil, Kafaları Değiştirin: Agile ve Şekilcilik

Agile Manifesto'nun tanımladığı değerleri ofisin duvarına yazıp bütün çalışanlara imzalatınca dönüşmüş olmuyorsunuz. Önemli olan ofisin duvarlarının değil, insanların kafasının o değerlerle, ilkelerle doldurulmasıdır.

Başta yönetiminiz o imzaladığı değerlere uygun davranıyor mu acaba? Veya çalışanlar altını imzaladıkları o değerlere uygun davranmadığında ne yapıyorsunuz? *"Bunu imzalamıştın, sözleşmeye*

aykırı davrandın, eve gelip buzdolabını alacağız" mı diyorsunuz?

Şu şekilciliği bir kenara bırakın artık!

Farklı sektörlerden bir sürü şirketin içerisine girmiş biri olarak benim gözlemim şudur:

"Bir şirket ofisinin duvarlarına büyük harflerle ne yazıyorsa o olay o şirkette yoktur."

Mesela duvarda kocaman **"TRUST"** yazan şirketlerde emin olun kimse birbirine güvenmiyordur, herkes arkasını kollamaya odaklıdır. Duvarında kocaman **"TRANSPARENCY"** yazan şirketlerde mutlaka gizli ajandalar vardır.

Duvarları değil, kafaları değiştirin.

28. Önce Anla, Sonra Değiştir: Kod Bataklığı Tehlikesi

Kod üzerinde değişiklik yapmadan önce mevcut kodun ne iş yaptığını anlayın.

Evet üzerinizde zaman baskısı var ve kod üzerinde hızlıca yaptığınız bir düzenleme (*"şuraya bir if eklersek*

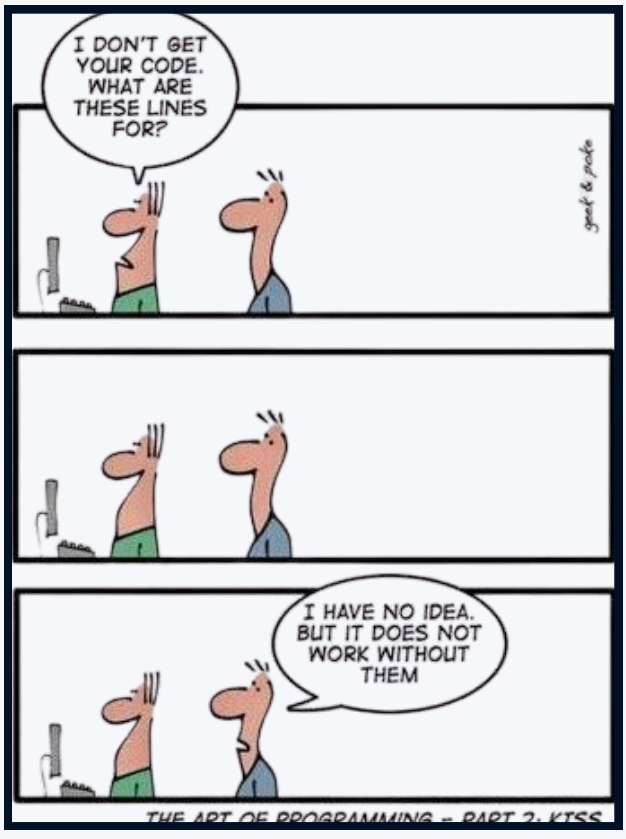
şuraya bir if eklersek halloluyor abi" veya *"şu flaggedCells değişkenine 1 ekledik mi çözeriz bu işi")* eldeki hatayı fix edebilir. Ancak, kod anlaşılmadan yapılan bu hızlı fixler bir süre sonra **kodu bataklığa çevirir.**

Gerçek bir çevik takımdaki geliştiriciler bir sonraki adıma geçecek ve bu if'in veya +1'in neden gerekli olduğunu ve daha da önemlisi başka nelerin bu değişiklikten etkilendiğini anlamaya çalışacaktır.

Takımınız bunu yapmadığında kod tabanınız sizi bir bataklık gibi içine çekmeye başlar ve ciddi vakit kayıplarına / kalite problemlerine neden olur.

Sonra bana şöyle gelmeyin:

"Scrum'a ilk başladığımızda projemiz yeniydi ve Sprint'lerin sonunda bir sürü iş teslim ediyorduk, 1 senenin sonunda velocity yarı yarıya düştü, niye böyle oldu?"



29. İsimler ve Gerçekler

Takımlara "**squad**", domainlere "**tribe**" veya "**garage**", uzmanlık alanlarına "**chapter**" dediğinizde daha çevik olduğunuzu mu düşünüyorsunuz?

Müşteri aynı müşteri, çalışan aynı çalışan, yönetici aynı yönetici... Kafaların içinde, yetkinliklerde değişen en ufak bir şey yok ama maşallah iş ambalaja gelince beylik lafların bini bir para.

O sizin "transformasyon" dediğiniz şey ünvanları, isimleri değiştirmekle olmaz; insanları/kafaları değiştirerek, geliştirerek olur.

Bakın Scrum'ı ortaya koyan iki kişiden biri olan ve entegre sağlık hizmetleri bilgi sistemlerinin lider sağlayıcısı olan PatientKeeper, Inc.'in eski CTO'su Jeff Sutherland, şirketteki ürün sahiplerinin (Product Owner) gerekli niteliklerini ve yetkilerini nasıl açıklıyor:

> "[Bir ürün sahibi] bir alan uzmanı, tercihen haftada birkaç gün Boston'ın önde gelen

> hastanelerinden birinde pratisyen hekim olmalıdır... Bir yazılım mühendisliği uzmanı olmalı, tercihen kendisi bazı uygulamalar yazmış olmalıdır...
>
> Kullanıcı hikayeleri (user stories), use cases ve genel olarak yazılım spesifikasyonları ve özellikle sağlık hizmetleri konusunda uzman olmalıdır... Gereksinimleri ortaya çıkarmak müşterilerimiz ve satış ekibimiz ile gerçekten yakın çalışmalı ve yeni işlevlerin prototiplerini test etmek için uzman doktorları işe almalıdır. [Ürün sahiplerimiz], genellikle geliştiriciler ve diğer ürün sahiplerinden başka birinin yardımına ihtiyaç duymaz. Bu pozisyon için yaptığımız ilk birkaç işe alım bunu sağlayamadı. Tekrarlanan eğitim, koçluk ve doğru kişiyi işe almak, bunun gerçekleşmesini sağladı."

Demek ki neymiş?

Birine "sen Product Owner oldun" demekle iş bitmiyormuş. Sen dersin de, bakalım o dediğin kişi gerçekten onun içini doldurabiliyor mu? Eskisine göre bir şeyleri farklı yapıyor mu? O rolün içini

doldurmak için kendisini geliştiriyor mu?

Aynı şekilde krediler müdürlüğü yerine artık "**krediler garajı**" demekle, "**dijital varlıklar tribe'ı kurduk**" demekle bu işler olmuyor maalesef. Keşke o kadar kolay olsaydı.

Sahi ne diyordu rahmetli Barış Abi?

> *"Altın çöpe düşse değerin kaybeder mi?*
> *Tenekeyi parlatsan hiç çeyrek altın eder mi?"*

30. Yönetim Gerçekten İstiyor mu?: Değişimin Anahtarı

Herhangi bir değişikliğe girişmeden önce yönetiminizden onay ve destek almalısınız.

Neden?

Çünkü çalışanlar kendilerine söyleneni yapmaya eğilimlidirler, ta ki bu çalışanlar aslında başka bir şey yaptıklarında ödüllendirileceklerini anlayana kadar.

Çalışanlara ne yapacaklarını söyleyenler

yöneticilerdir. Çalışan ödüllerini belirleyen kişiler yöneticilerdir. Değişim ancak yöneticiler *"Evet, bunu yapmalısın"* dediğinde başarılı olur ve daha sonra değişimin gerektirdiklerini yapan çalışanları ödüllendirir.

Bu nedenle, sizin ve ekibinizin çevikliği başarılı bir şekilde benimsemesi için önce yönetiminizi bunu desteklemeye veya en azından engellememeye ve ekip üyelerini bunu denediklerinden dolayı cezalandırmamaya ikna etmelisiniz.

Kendiniz bir yöneticiyseniz, yönetim zincirinizin çevik yaklaşıma karşı çıkmamasını sağlamalısınız.

> **Burada kilit nokta şu:**
>
> *Çevikliği destekler gibi görünen yöneticilerin çoğu kendi alışkanlıklarını da değiştirmeleri gerektiği gerçeği ile karşı karşıya kaldıklarında çoğunlukla bu değişimden vazgeçerler, hatta ellerine geçen her fırsatta bunun altını kazmaya başlarlar.*

Şimdi tekrar düşünün bakalım. Yönetiminiz gerçekten

çevikliği destekliyor mu? Bu fikirleri gerçekten satın aldı mı?

Bu soruya yanıtınız **"Dönüşümünüz başarılı olacak mı?"** sorusunun da yanıtını bulmanızı sağlayacak.

31. En İyisi Olmak Yetmez

> *"Takımlarınızdaki kişiler bildiklerini paylaşmayıp, bilgiyi kendine saklasın. Sonuçta takımdaki en bilgili kişi olmak onlar için büyük bir avantaj. En iyisi kendileri olduğu sürece diğer kaybedenleri göz ardı edebilirler."*

Yanlış!

Ekibinizde farklı yeteneklere, deneyime ve becerilere sahip insanlar var. Her insanın farklı güçlü yönleri ve uzmanlığı var. Farklı yeteneklerin ve geçmişlerin bu karışımı, öğrenme için ideal bir ortam yaratır.

Bir takımda, kişisel olarak çok şey biliyor olmanız yeterli değildir. Ekibinizin diğer üyeleri de bu kadar bilgili değilse, ekip olabileceği kadar etkili değildir:

iyi eğitimli ve birbirinden öğrenen bir ekip daha iyi bir ekiptir.

Sizin veya ekibinizdeki bilgili birinin ekibin geri kalanının hızlanmasına yardımcı olabileceği alanları bulun. Bu, konuların projelerinize nasıl uygulanacağını tartışabilmeniz için ek bir avantaj sağlar.

Adına ister **"lunch&learn"**, ister **"BBS"**, ister **"öğren molası"** deyin, bu takım üyelerinin birbirlerinden öğreneceği oturumlarının faydası büyüktür.

32. Sadece Toplantı Yöneticisi Değil: Scrum Master ve Etkinlik

Scrum Guide diyor ki;

> *"Scrum Master, Scrum Takımı'nın etkinliğinden (effectiveness) sorumludur. Bunu, Scrum Takımı'nın pratiklerini Scrum çerçevesi dahilinde iyileştirmesini sağlayarak yapar."*

Yani bu kişi takımının nasıl daha efektif olabileceğine kafa yormalı, takımı daha üretken

kılabilecek pratikleri **araştırmalı, öğrenmeli** ve bu pratikleri takımının uygulamaya koyabilmesi için takıma **öğretmeli.**

Takımını daha üretken hale getirebilecek pratikleri bilmeyen, öğrenmeyen ve bunları öğretmeyen Scrum Master **işini yapmayan Scrum Master**'dır.

Olay sadece *"arkadaşlar Daily'nin timebox'ı 15 dakika"* veya *"Bugün yeni bir retrospective tekniği deneyeceğiz"* demekten ibaret değil.

Scrum Master'lık öyle herkesin yapabileceği bir şey de değil.

"Bizim ekipte çok istekli yeni mezun bir arkadaş vardı, onu Scrum Master seçtik" şeklinde karar verilebilecek bir rol hiç değil.

33. Scrum Fanatizmi: Kutsal Kitap mı, Araç mı?

Türkiye'de "Agile Coach" ünvanına sahip kişilerin birçoğu koyu bir Scrum fanatiği.

Scrum'ı ölesiye savunuyorlar, Scrum Guide'a

kutsal bir kitap gibi bakıyorlar. Guide'da yer alan cümlelerin anlamları üzerine falan saatlerce tartışıyorlar. Konu hakkında meetuplar düzenleyip, Scrum Master rolü veya Product Backlog yönetimi hakkında konuşuyorlar.

Scrum Guide'ı yazmış olan 2 kişinin bile bu metin üzerine bu kadar kafa yorduklarını düşünmüyorum.

(XP gibi) alternatif herhangi bir Agile yöntem ile çalışmamışken bu arkadaşların nasıl bu kadar Scrum fanatiği olabildiklerini de çok merak ediyorum.

Herhangi bir alanda bir şeyi bu kadar tutkuyla savunabilmek için o alandaki diğer bütün alternatifleri öğrenmiş, hatta denemiş olmak gerekir.

> **Ömrünüz boyunca tek bir otomobil modeli kullanıp, bunun diğer tüm otomobillerden "daha iyi" olduğunu nasıl iddia edebilirsiniz?**

Bir konuda "fikir" sahibi olmadan önce "bilgi" sahibi olmak gerekiyor.

Hepi topu 14 sayfa kılavuzu olan Scrum'ı bu kadar abartmak yerine Agile dünyasında başka ne var ne yok, biraz derinlere inin derim. Hatta sadece Agile dünyasındaki şeylerden de öte **yazılım mühendisliğinin iyi uygulamalarını öğrenmek için çaba sarf edin.**

34. Scrum Sadece Bir Çerçeve

Scrum sadece basit bir çerçevedir.

Takımlarınızdaki insanlar, Scrum'ın getirdiği rollerin, etkinliklerin ve eserlerin arkasındaki **esas güçtür.**

> Takımınızdaki insanlar işbirliği yapmıyor, birbirlerine yardım etmiyor, destek vermiyor ve kendilerini geliştirmek için zorlamıyorlarsa, müşterilerinize hizmet etme şeklinizi geliştirmek için çok önemli bir fırsatı kaçırıyorsunuz demektir.

35. Kırmızı Bayraklar: Şikayet ve Suçlama Kültürü

Takımlarınızda **şikayet** ve **suçlama**yı kırmızı bayraklar olarak görmelisiniz.

Sürekli şikayet eden insanlar/takımlar sorunların çözülmediği, sadece tartışıldığı bir olumsuzluk ve çaresizlik kültürü oluşturur.

Sohbeti, takımınızın tartıştıkları engel veya soruna nasıl sahip çıkabileceğine yönlendirin.

Bu, uzun vadede çok daha üretkendir ve ekibinizin kendi kendini organize etme ve karmaşık problem çözme gibi önemli becerileri uygulamasına yardımcı olur. Takımlarınızın sorunlarına sahip çıkmalarına ve sorunlarını yaratıcı bir şekilde çözmenin yollarını bulmalarına yardımcı olun.

> **Şikayet etmek, bir takımı çözüme yaklaştırmaz.**
>
> Konuşmanın bir şikayet oturumuna dönüştüğünü görürseniz, takımınızı tekrar problem çözmeye yönlendirin.

36. Hafızaya Güvenmeyin!

Ürün geliştirme sürecinde aldığınız kararları ve bu kararların arkasındaki nedenleri **kayıt altında tutun.** Hepimizin bildiği gibi insan belleği güvenilmezdir.

> Çeviklik, *"sadece kendi aranızda konuşun, hiç bir şeyi dokümante etmeyin"* gibi bir şey söylemiyor.

Takımınızın tuttuğu bir günlük (journal), herkesin okuyup yazabildiği bir Wiki, çok sevimli olmasa da bir e-posta izi veya bir issue tracking uygulaması gibi opsiyonların tümü kabul edilebilir; **yeter ki seçtiğiniz yöntem çok ağır veya külfetli olmasın.**

37. Bazen 1 Görsel 1000 Kelimeye Bedeldir

Bilgisayar bilimcileri aynı cümleyi "Bazen 1 görsel 1024 kelimeye bedeldir." şeklinde kurar. O yüzden serinin 37. post'u sadece bir görsel.

İşte size "Nasıl Agile Olunmaz?"ın 1024 kelimeye bedel özeti:

38. Scrum + JIRA Yeterli Değil

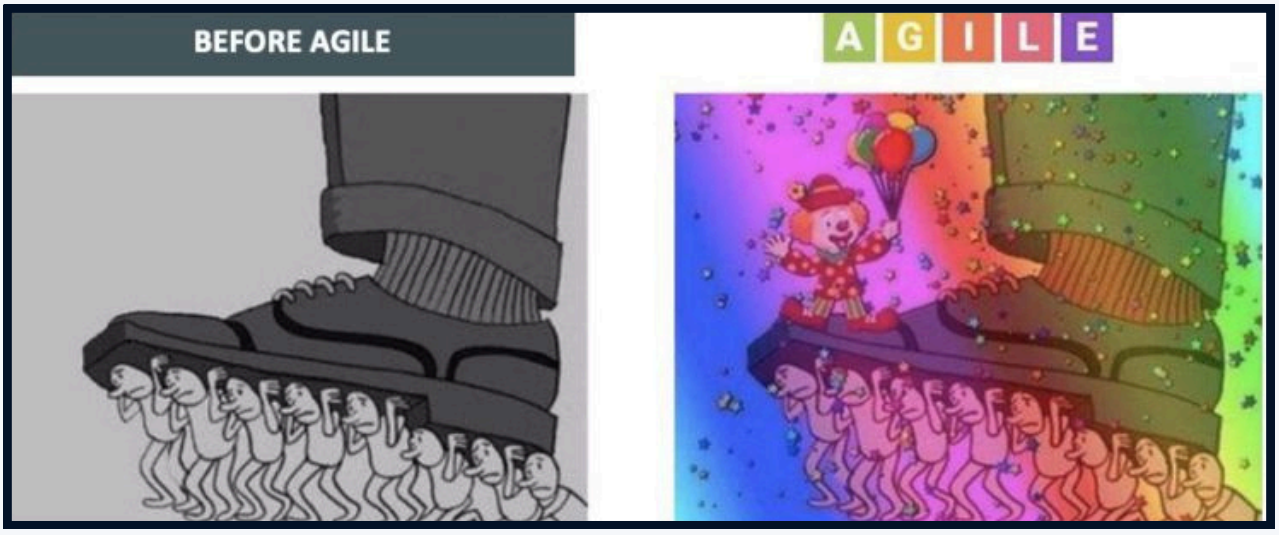
Sadece "Scrum + JIRA" ile çevik olamazsınız.

Bunların ikisini hiç bilmeyen/kullanmamış bir takımda uygulamaya koymak 2-3 günde yapılabilecek bir şey. (Hee tabi bunu da abartıp 1 senede anca bu noktaya gelen şirketler de yok değil.)

Bu kadar basit olsaydı **Salesforce**'un yaptığı gibi, **American Airlines**'ın yaptığı gibi gerçek çevik dönüşümler en az 3-4 sene sürmezdi, değil mi? Takımların teknik yetkinliklerini iyileştirmeden, legacy uygulamalarınızı adam akıllı refactor etmeden, yönetimin kafa yapısını değiştirmeden "Scrum + JIRA" ile elde edebileceğiniz tek şey şudur:

Bu işi yapamadığınızı 2 haftada 1 görmek (belki de görmemek) olur.

39. Renkli Post-itler ve Gerçekler: Göz Boyama Sanatı



Çevik dönüşümden geçtiğini zanneden pek çok kurumdaki "gerçek durum" görseldekinden ibarettir.

Ortalığı renkli post-itlerle donatarak, "şöyle böyle dönüştük" diye havalı sunumlar yaparak **olmuyor bu işler.**

- Çalışanların seslerinin duyulmadığı,
- Kararların hiyerarşinin en tepesinden alındığı,
- Değeri üretenden çok, değeri üreteni yönetene değer verildiği,

ortamlarınızı değiştirmediğiniz sürece anca ortalığı rengarenk gibi gösterip **göz boyarsınız.**

40. Moda Diye Dönüşmeyin: Amaç ve Sonuç Nerede?

Rakibiniz bir dönüşüm projesi başlattığı için veya bir danışmanlık şirketi size bu dönüşümü yapmanız gerektiğini söylediği için başlattığınız dönüşüm projesinden **hayır gelmez.**

Öncelikle şu soruların yanıtlarını düşünmek gerekir:

- Çevik dönüşüm ile neyi amaçlıyorsunuz?
- Neyi iyileştirmeye çalışıyorsunuz?
- Bunu nasıl ölçüyorsunuz?
- Daha iyiye gittiğinizi neye bakarak anlayacaksınız?
- Çalışanlarınız/müşterilerinizin karşılanmamış ihtiyaçları/engelleri neler?

Bu mecrada *"Çevik Dönüşüme başladık, uçuyoruz, kaçıyoruz"* postlarını çok gördüm de, *"Çevik Dönüşüm ile şunu elde ettik"* diye yazanı hiç görmedim nedense.

En basitinden benim gördüğüm/bildiğim bütün dönüşümlerde başlarken çalışanlar ve/veya

müşteriye uygulanan NPS ile dönüşüm belli bir noktaya geldikten sonra uygulanan NPS arasında ya hiç bir fark yok, ya da değerler daha kötüye gidiyor.

Şirketlerin C-level'larının havalı demeçlerini bir kenara koyup, gerçek çalışanlara / müşterilere bir soralım bakalım.

1 hafta açık kalacak ankete katılın, kurumunuzun gerçekleştirdiği dönüşümü değerlendirin.

41. İdeal Product Owner Profili: Yazılımı Bilmek Şart mı?

Eğitimlerde Scrum'ın rollerinden bahsettiğimde bana hep sorulur:

> "*Hocam Product Owner (Ürün Sahibi) rolü için uygun profil nedir?*" diye.

Ben de hep şöyle cevap veririm:

*"Bir yazılım ürününden bahsediyorsak, o ürünün Ürün Sahibi **yazılım işinden anlamalı.** Elbette domain bilgisi, market bilgisi de çok önemli ama geliştirmeye çalıştığımız ürün "yazılım" olduğuna*

göre bunun sahipliğini üstlenecek kişi de yazılımdan anlamalı."

Hatta yöntemi ortaya koyan iki kişiden biri olan Jeff Sutherland aynı soruyu şu şekilde yanıtlamaktadır diye örnek veririm:

"[Bir ürün sahibi] bir alan uzmanı, tercihen haftada birkaç gün Boston'ın önde gelen hastanelerinden birinde pratisyen hekim olmalıdır... bir yazılım mühendisliği uzmanı olmalı, tercihen kendisi bazı uygulamalar yazmış olmalıdır...

Kullanıcı hikayeleri (user stories), use cases ve genel olarak yazılım spesifikasyonları ve özellikle sağlık hizmetlerinde yazılım gereksinimleri konusunda uzman olmalıdır... Gereksinimleri ortaya çıkarmak müşterilerimiz ve satış ekibimiz ile gerçekten yakın çalışmalı ve yeni işlevlerin prototiplerini test etmek için uzman doktorları işe almalıdır.

[Ürün sahiplerimiz], genellikle geliştiriciler ve diğer ürün sahiplerinden başka birinin yardımına ihtiyaç duymaz. Bu pozisyon için yaptığımız ilk birkaç işe alım bunu sağlayamadı. Tekrarlanan eğitim, koçluk ve doğru kişiyi işe almak, bunun gerçekleşmesini sağladı."

O zaman iş birimlerinden domain uzmanlıklarını nedeniyle PO yaptığımız arkadaşlarımız için ne yapıyoruz?

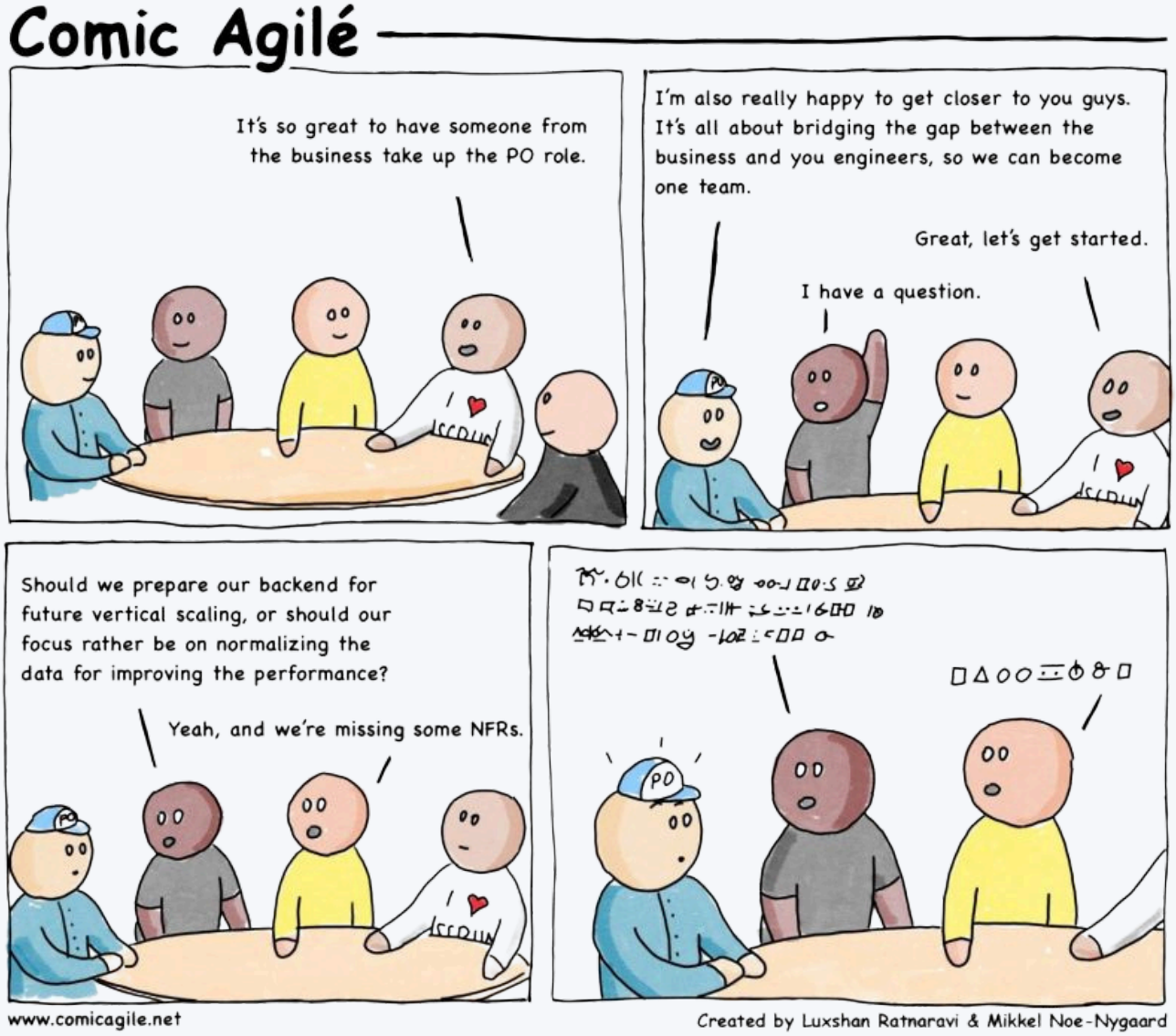
Cevap: Onları eğitiyoruz.

Peki bu kolay mı?

Cevap: Hiç değil. Size bu işin kolay olduğunu söyleyenlere de inanmayın.

Bu yol uzun, bu yol çok çaba gerektiyor. Ama sonu gerçekten daha iyi bir yere çıkıyor.

Bu yolda gerektiği gibi yürüyün, aksi halde sizin durumunuzu en iyi Comic Agilé'ın şu görseli özetler.



42. Kod Zamanla Çürür: Teknik Borç ve Refactoring

"Teknik borç" çoğunlukla kodlama aşamasında yeterince iyi iş çıkartmamış olmanızın sonucu değildir.

(Tabi ki bu durumdan da kaynaklanan teknik borç vardır. "Doğru" olanı yapmak yerine "hızlı" olanı yapmayı tercih ederek teknik borç yaratabilirsiniz.)

Çoğunlukla "teknik borç" geliştirdiğiniz uygulama hakkında daha fazla bilgi edindikçe, daha fazla öğrendikçe "refactoring" yapmamanızın maliyetidir.

Robert C. Martin'in dediği gibi:

> *"Kod zamanla çürür."*

İlave gereksinimler ortaya çıktıkça ve uygulama genişledikçe bu çürümeyi engellemek gerekir.

 **Çare refactoring!**

Tabi en başta bunun gerekli olduğunu bilen ve buna alan açan, zaman ayıran bir **yönetim zihniyeti** lazım.

Sizin şirkette o zihniyetten var mıdır, bilemem.

43. Ağaç Dikerek Çevik Olunur mu?

Bir caps falan mı diye durdum bir an bu fotoğrafı görünce...

Neden bir şirket böyle bir şey yapar diye de uzun uzun düşündüm. Kim verdi mesela bu kararı?

Ağaç dikilmesine elbette söyleyecek hiç bir şey yok, dünyanın en güzel hareketi. Yalnız ormanın adı neden **Agile Ormanı**?

- Birincisi; "agile" kelimesi bir sıfat, "çevik ormanı" diye bir şey olmaz.
- İkincisi; o sene emekli olan çok değerli bir

> çalışanın ismini vermek dururken veya bir iş kazası geçirmiş çalışanın adına bu işi yapabilecekken neden "Agile Ormanı"? Mavi yaka bir çalışanızın yeni doğan çocuğunun adına olsa mesela daha hoş değil mi?

Ayrıca nedir bu ormanı agile yapan diye sorsam. Ağaçlar değişime daha mı hızlı tepki veriyor sizin ormanda?

Bir başkası da çıkıp Waterfall Korusu, Spiral Göleti, V-Model Vadisi mi yapsın buna cevap olarak? Ya da biz de şirket olarak Scientific Management Millet Bahçesi'ne sponsor mu olalım?

Hep söylüyorum, *"şirketler sahip olamadıkları şeyleri duvarlarına yazarlar"*.

> Şimdi de bunun yanına ekliyorum, ***"şirketler sahip olamadıkları şeylerin ormanını yaparlar".***

Sevgili yöneticiler, karar vericiler biraz düşünseniz mi acaba bu işleri yaparken?

Ağaç dikerek çevik olmak gibi bir beklentiniz varsa ben size söyleyeyim o işler öyle olmuyor maalesef.

44. Her Şeyin Başına ”Agile” Koyunca Oluyor mu?

Gökrem Tekir hocam ile dün akşam dertleştik biraz. Her şeyin başına "Agile" getirilmesinden o da çok sıkılmış.

> "*Böyle her lafın başına 'Agile' koyunca oluyor mu bu işler Onur?"* dedi.
>
> *"Olmuyor elbette hocam"* dedim.

Gerçekten değer üretmeye odaklanmak yerine laf salatası ile günü kurtarıp, "Agile" lafı ile prim yapma olayı ne zaman son bulacak merak ve şaşkınlık içerisinde bekliyoruz hep birlikte.

Hocam üşenmemiş bir de listesini yapmış bu lafların:

Agile Danışman
Agile Koç
Agile İnşaat Yönetimi
Agile Proje Yönetimi
Agile Proje Yönetimi Ofisi
Agile İnsan Kaynakları
Agile Ekip (Takım)
Agile Proje Yöneticisi
Agile Dönüşüm
Agile Metodoloji
Agile Planlama
Agile Okul Ödevi Yapmak
Agile Ateşi
Agile Pazarlama
Agile Satın Alma
Agile Eğitim ....

Bir önceki gönderide söylediğim gibi bir de "Agile Ormanı" var tabi...

Bir de benim çok sevdiğim "Agile Software Requirements" var. Waterfall Software Requirements diye bir şey yok ama nedense... Sanki bir projede agile yaklaşımı kullanınca işi yapacak insanların önüne koymamız gereken bilgiler farklıymış gibi gereksinimleri de "agile" diye etiketliyorlar bazı arkadaşlar.
Ama bunların hiç biri şu linkteki "Agile Sex" kavramının önüne geçemez: [Link] 🔗

Listeyi daha da uzatmak mümkün elbette. Sizin duyduğunuz "Agile ....." 'lar varsa yorumlarda paylaşırsanız listemizi genişletiriz, atladığımız bir şey kalmasın aman ha.

Ne yapıyor olursanız olun esas olan değer üretip paydaş memnuniyeti sağlayıp sağlayamadığınızdır. Bu kendi içinde yaptığınız şeyi hızlı yapmayı, kaliteli yapmayı veya doğru şeyi yapmayı zaten içerir. Bunu elde etmenin de pek farklı yolları olabilir.

Laflara değil de aksiyonlara odaklandığımızda

ve gerçekten neyi ne şekilde iyileştirdiğimizi somut bir şekilde ortaya koyduğumuzda bu işler daha iyi olacak Gökrem Tekir hocam! 44 nolu post'a katkıların için sonsuz teşekkürler.

45. Kimsenin Bir Şey Bilmediği Yerde: ”Biz Agile Olduk”

Son dönemde şirketlerin sürekli "Biz agile olduk" konulu paylaşımlarını görüyoruz Youtube, Linkedin vb. platformlarda.

Durumu özetleyen video ektedir.

Feyyaz'ın da dediği gibi:

> *"Kimsenin hiç bir şey bilmediği yerde bir insan her şeyi bilebilir, bu kadar yani."*

46. Şekil Var, Öz Yok: Türkiye’de Agile Yanılgısı

Türkiye'de "Agile"'ı kafasında çok başka yerlere koyan o kadar çok kurum var ki...

- Ne olduğunu, kurumda neleri değiştireceğini tam anlamadan sadece "havalı" görünmek adına yapılan dönüşümler,
- "Rakip yaptıysa biz de mutlaka yapmalıyız, neyimiz eksik" yaklaşımı,
- Olayın sadece şeklinin var olduğu (posterler, videolar, şarkılar... vb), özünün kaybolduğu ortamlar...

Sonunda pişman olacağınız işler yapmayın.

Niyetiniz -gerçekten- bir şeyleri değiştirmekse, bizim gibi danışmanlardan çok takımlarınızı dinleyerek başlayın.

> Ellerini kirleten insanlar neyin nasıl olması gerektiğini de çok iyi biliyor.

47. Grooming mi, Refinement mı?: İsimlere Takılıp Özü Unutmak

Vay efendim "Development Team" demeyecekmişiz, Scrum Guide'a gelen son güncellemeyle onun ismi "Developers" olarak değişmiş.

Grooming denmezmiş, onun ismi Refinement'mış. Ürün vizyonu olmuyormuş, "Product Goal" demek lazımmış.

İsimlere, olayın şekline neden bu kadar takılıyorsunuz sevgili takımlar / şirketler?

Olayın şeklini tartışmaktan, özünü unuttunuz!

Garaj, alan, tribe lafları havalarda uçuşuyor da, müşterileriniz halen "doğru ürünü üretmediğinizi" veya "ürettiğiniz şeyi doğru ve hızlı üretmediğinizi" basbas bağırıyor.

Takımlarınız da "burada çalışan memnuniyeti yok" diye basbas bağırıyor.

Sahi siz yaptığınız bu dönüşümlerle gerçekten kimi memnun ediyorsunuz?

İsimlere çok takılıyorsunuz, neden isimlere bu kadar çok takılıyorsunuz?

48. ”Daha Hızlı” Ama Ne Kadar?

Müşteri: "Onur Bey merhaba, biz sizden çevik dönüşüm danışmanlığı almak istiyoruz."

Ben: "Bu konuda size yardımcı olabiliriz. Neden böyle bir dönüşüme ihtiyacınız olduğunu düşünüyorsunuz?"

Müşteri: "Daha hızlı yazılım geliştirmek istiyoruz."
Ben: "Peki mevcutta nasıl bir hızla yazılım geliştiriyorsunuz?"

> **Müşteri:** "Mevcuttaaaaa?!?, o konuda bir şey ölçmüyoruz, o nedenle size bir cevap veremiyorum."

Ben: "Peki, çevik dönüşümün başka hangi problemlerinize derman olmasını bekliyorsunuz?"

Müşteri: "Daha yüksek müşteri ve çalışan memnuniyeti elde etmemizi sağlarsa çok iyi olur."

Ben: "Müşterileriniz ve çalışanlarınız bugün hallerinden ne kadar memnun?"

> **Müşteri:** "Pek memnun değiller ama bu konuda da size net bir cevap verememem."

Ben: "Peki gerçekleştireceğimiz çevik dönüşümün bu alanlarda bir iyileşme sağlayıp sağlamadığını nasıl anlayacağız? Hatta problemlerinizi çözecek şeyin 'çevik dönüşüm' olduğu kararını nasıl verdiniz?"

Müşteri: "Ama Onur Bey, biz sizden çevik dönüşüm danışmanlığı almak istiyoruz."

Ben: "Bu konuda size seve seve yardımcı olabiliriz. Ancak gerçekten neden böyle bir dönüşüme ihtiyacınız olduğunu düşünüyorsunuz?"

49. Ahkam Kesmek Kolay

Hayatında 1 tane ürünün sorumluluğunu almamış agile coach'lar "Product Ownership", "Scrum Master'lık" ile ilgili ahkam kesiyor ve kurumsal dünyanın gerçeklerinden haberleri yokmuş gibi insanları işlerini düzgün yapamadıkları için eleştiriyor.

Siz önce içinde bulunduğunuz ülkedeki bir şirkette bir yazılım projesinin içerisine **-gerçekten-** dahil olup o insanların yaşadığı stresi, yönetimden paydaşlardan gelen baskıyı bir yaşayın.

Hiç bir dayanağı olmadan takımın önüne koyulmuş bir deadline'a yetişebilmek için aylarca geceli gündüzlü mesai yapmak zorunda kalın bakalım adınızı söyleyecek haliniz kalacak mı?

İnsanları eleştirmek kolay da, insanların o davranışları sergilemesine neden olan ortamı ve yönetim zihniyetini değiştirmek o kadar kolay değil maalesef.

Makul olmayan insanların makul olmayan beklentilerini, hatta fantezilerini tatmin etmeye çalışan insanlar sizin **"havalı ama bir işe yaramayan"** söylemlerinize pek de muhtaç değil bence.

Yapın bir takımda product owner'lık, scrum master'lık biz de görelim bakalım söylediklerinizin ne kadarını -gerçekten- yapabiliyorsunuz.

50. Tersine Mühendislik Çilesi

Agile veya yazılım gereksinimleri hakkında bir eğitim verdiğimde katılımcılara sorarım:

> *"Geliştirmiş olduğunuz bir uygulama üzerinde nasıl değişiklik yapacağınızı, yeni talep edilen bir fonksiyonu nasıl ekleyeceğinizi anlamak için mevcut uygulamanın kodunu okuyup bilgiyi koddan sökmek için tersine mühendislik yapmak zorunda kaldınız mı?"*

Bu soru karşısında neredeyse eğitimdeki herkes elini kaldırır.

Sonra tersine mühendislik ile elde ettikleri bilgileri ileride başvurmak üzere dokümante edip etmediklerini sorarım. Sadece birkaç el yukarı kalkar.

"Neden?" diye sorarım?

Hemen her eğitimde birisi *"Ama hocam çalışan yazılım, kapsamlı dokümantasyon'dan daha değerlidir demiyor mu Agile Manifesto"* diyerek kendini savunur.

Benim ise şu açıklamayı yapmam gerekir:

> *"Agile Manifesto sadece yazılım üretin, dokümantasyon üretmeyin, dokümantasyon*

> *kötü bir şeydir demiyor. Ortada çalışan yazılım yokken sayfalarca üretilmiş bir dokümantasyon müşteri için değerli değildir diyor."*

Elde ettiğiniz bilgiyi kayıt altına almamanız, gelecekte başka bir değişiklik yapmak için uygulamanın aynı bölümüne giren birinin koddan bilgiyi sökmek için tersine mühendislik sürecini tekrar etmesi gerektiği anlamına gelir.

Eee, nerede kaldı çeviklik? Hani hızlıca değişime adapte oluyorduk? Yavaşladık işte!

Ayrıca, tersine mühendislik yoluyla koddan bilgiyi sökmek insanlar için sıkıcıdır. Bu işi tekrar tekrar yapmak da aşırı verimsizdir.

Öğrendiklerinizi yazarsanız, bu bilgiler sizin veya başka birinin ona geri dönmesi gerektiğinde kullanılabilir. Bu, siz üzerinde çalışırken kötü belgelenmiş bir uygulama hakkında aşamalı olarak bilgi biriktirmenin bir yoludur. Agile Manifesto'nun eş yazarlarından Ward Cunningham'ın yazılım takımlarının kolektif bir şekilde dokümantasyon yapmasına izin veren Wiki (What I Know Is) fikrini

geliştirmiş olması da bunun kanıtıdır.

Tersine mühendislik sonucunda öğrendiklerinizi yazmak, bu bilgileri öğrenmek için harcamanız gereken zamandan daha az zaman alır. Tabi geliştirdiğiniz uygulamada tersine mühendislik yapmanıza gerek kalmayacak şekilde dokümantasyon yapmak en iyisidir.

Uygulamanız ile ilgili edindiğiniz bilgileri kaydetmemeye elbette karar verebilirsiniz, ancak bu ancak siz de dahil hiç kimsenin sistemin o kısmıyla bir daha çalışmak zorunda olmayacağından emin olduğunuzda anlamlıdır. Hepimiz biliyoruz ki bu çok nadiren söz konusu olabilecek bir durumdur.

Geleceği tahmin etme konusunda yetenekli olmadığımız için, bilgileri paylaşılabilir bir formda tutmayı tercih edin. Bu pratik, edindiğiniz bilgiyi (zamanla kaybolacağı kesin olan) beyninizde saklamaktan daha iyidir.

Dokümantasyonun tek yolunun Türkçe / İngilizce gibi doğal bir dil kullanmak olmadığını da biliyoruz. **Doğal dil kullanarak yazılmış sayfalarca düz yazı**

yerine bir görsel model insanlar tarafından çok daha hızlı bir şekilde anlaşılır.

51. Bir Yerde Bir Şeyler Yanlış

- Neredeyse ülkedeki bütün İK'cılar "çevik çalışıyor" ama çalışanlar mutsuz, turnover tavan...
- Ülkedeki bütün liderler iki günlük sınıf eğitimi ile "Çevik Lider" oluverdi ama işten ayrılmalarda hala en önemli neden "yönetici"...
- Herkes Scrum uyguluyor ama "scope change" yemeden biten Sprint yok...
- Herkes kaliteden bahsediyor ama gittiğim her kurumda istisnasız her takımın backlog'unun yarısından fazlası "hata" tipinde işler...

3 şirketten 2 tanesinin üst yöneticilerinin dilindeki söylem, sosyal medya postları:

> *"Biz de şahane bir çevik dönüşüm yaptık geçen sene"*

Aynı şirketlerin çalışanlarının söylemleri:

> *"Hocam danışmanlık firması 2 haftalık Sprint'leri zorunlu tutuyor, iş birimlerindeki arkadaşlar da size estimation seanslarına dahil olup bizce bu iş 3 sp diyorlar, kafayı yedik artık"*

Bir yerde bir şeyler yanlış gibi sanki ama hadi neyse...

52. Yardımcı mı, Köstek mi?: Agile Rollerindeki Tehlike

Takımın planlama toplantısına katılan **Scrum Master / Kanban Master / Agile Coach / Agile Danışmanı** takıma yardımcı olduğunu düşünüyor...

Kıssadan hisse:

Videodaki gibi davranan arkadaşlar doğru ürünü üretmenize, ürünü doğru üretmenize veya hızlı üretmenize katkı sağlayamaz...

Bu rollere konumlayacağınız insanları **dikkatli bir şekilde seçiniz...**

53. Tahmin mi, Hedef mi?: Rasyonel Olmayan Beklentiler

Şunu net bir şekilde anlamamız gerekiyor: Yönetimin veya pazarlama departmanının söylediği (ve hatta dayattığı) bir teslim tarihi bir **tahmin değil, bir hedeftir.**

Maalesef çoğunlukla gerçekçi olmayan hedefler belirlemek konusunda çok becerikli insanlar vardır ve bu arkadaşlar takımınızın önüne bu rasyonel olmayan hedefleri şak diye koyarlar.

Bu noktada bazen öyle bir irrasyonellik olur ki takımın önüne hedef koyarken "*27 Nisan benim doğum günüm, bu iş o zamana yetişsin*" diyen insanlar olduğunu görmüşlüğümüz vardır.

> Aynı insanlar evlerine çağırdıkları badana ustasına *"burası benim 120 metrekare evim, bunu 2 güne boyamanı istiyorum"* diyebiliyorlar mı diye hep merak etmişimdir. Usta vallahi fırçayla vurur kafaya!

Belirli bir büyüklükte ve üretkenlik düzeyindeki bir takım, belirli bir zamanda ancak belirli miktarda -kaliteli- işlevsellik üretebilir.

Fiziksel bir ürünün üretildiği üretim bandının bir kapasitesi olduğunu hepimiz kabul ediyorsak, bu durumu da kabul etmemiz gerekir. 4 saatte bir "çıktı" üreten bir üretim bandına "bana 2 saatte bir çıktı ver" diyor musunuz?

Bandı hızlandırmak için yapılabilecek şeyler vardır elbette ama bunlar öyle parmak şıklattığınızda yapılabilecek şeyler değildir, yapısal değişiklerdir ve zaman alır. Belki orta vadede bandınızı 3 saatte bir "çıktı" verebilen hale getirebilirsiniz.

Tahminleme konusundaki öğretiler genellikle;

"Tahminlemeyi" yapılacak işin net olarak tanımladığı bir ortamda, işin yapılabilmesi için gerekli olacak çabanın, maliyeti ve zamanın ne olduğunu ortaya çıkarmak şeklinde anlatır.

Ancak tahminleme bazen farklı bir şekilde de çalışabilir.

Belirli bir tarihe kadar bir şeylerin mutlaka teslim edilmesi gerekiyorsa, o zaman tahmincinin kaliteyi düşürmeden ne kadar işlevselliğin o zaman sınırının içerisinde üretilebileceğini belirlemesi

gerekir.

Çevik yazılım geliştirmede kullanılan kullanıcı hikayelerinin tahminlenmesi ve bunlardan bazılarının sabit uzunluktaki bir sprint'e / iterasyon'a alınmasının arkasındaki mantık budur.

Taahhütler, sadece birileri tarafından önümüze koyulan ve genellikle makul olmayan hedeflere değil, **makul tahminlere dayanmalıdır.**

Belirlenen hedef tarihe kadar işin tamamlanması için gerçekçi bir olasılık yoksa, o iş parçası **gecikmiş olarak kabul edilmemelidir.**

54. Test Yoksa Çeviklik De Yok!

Çevik mühendislik pratikleri, programcıların çoğunun son 70 yılda sergiledikleri davranışları radikal bir şekilde değiştirmesini gerektirir.

Bu pratikler programcıları (çoğunun başlangıçta saçma olduğunu düşündüğü) bir dizi ritüele zorlar.

Eğer bu pratikleri kullanmaya kalkıştıysanız şu cümlelerden en az birini duyduğunuzdan eminim:

- *“Abi şu köşede kulaklığımı takıp kodumu yazıyordum, nereden çıktı şimdi Pair Programming !$!?"*
- *"Patron bu iş Cuma'ya yetişsin istiyor, testleri sonra yazarız."*
- *"Refactoring için zamanımız yok, mevcut kodun üzerine ekleyiverin şu yeni özelliği."*

Birçok programcı Agile dönüşüm yolculukları içerisinde bu pratiklerden uzak duruyor. (Aslında programcılardan öte şirketler demek lazım, çünkü çoğu durumda programcı bu pratikleri kullanmak istese de şirket buna izin vermiyor.)

Bu pratiklerden uzak duranların gerçekten çevik olma olasılığı düşüktür. Çünkü bu uygulamalar çevikliğin özüdür.

TDD, Refactoring, Simple Design ve evet, hatta Pair Programming olmadan, bir takımın gerçekten çevik olması mümkün değildir.

55. Sadece Soru Sormakla Olmaz: Koçluk ve Uzmanlık

Şöyle bir şeye dikkat çekmek istiyorum:

Bakıyorum da çeviklik koçları / danışmanlarının çoğu tarafından çevikliği öğretmek için kullanılan yöntemlerin ve tekniklerin çoğu, **profesyonel koçluk** ile ilgili şeyler.

Bunlar bireylerin ve grupların önlerine çıkan engelleri keşfetmelerine ve nasıl ilerleyeceklerine kendileri karar vermelerine yardımcı olan **“koçluk araçları”**dır. Yeni veya "agile" ile icat edilmiş şeyler de değildir. International Coach Federation (ICF) vb. kurumların programları ile bu alanda kendini yetiştirmiş onbinlerce insan vardır.

Bir koçluk yetkinliği, "keşif, içgörü, bağlılık veya eylem uyandıran sorular sormak" olan **güçlü sorgulama**dır (powerful questioning). Bunlar iyi araçlarıdır, kullanılması faydalıdır.

Ancak;

> Kanban'ı veya XP'yi hiç duymamış ancak bundan faydalanabilecek bir takımla çalışıyorsak, hiçbir güçlü soru veya diğer profesyonel koçluk tekniği, birinin kendiliğinden Kanban'ı veya XP'yi icat etmesine neden olmaz.

Bu noktada, çeviklik koçu /danışmanı, takıma potansiyel olarak faydalı olabilecek uzmanlık sunma moduna geçmelidir. Bunu yapabilmesi için de bu konularda "uzman" olması gerekir.

Birden fazla çevik çerçeve/yöntem, çevik ürün yönetimi, çevik mühendislik pratikleri gibi konularda bilgi ve tecrübe sahibi olmayan bir koçtan / danışmandan alacağınız fayda limitlidir.

"Koç" lafının başına "Agile" koymakla olmuyor maalesef bu işler.

2 günlük Scrum ya da Agile Coach eğitimi sonucunda Linkedin headline'ınını "Agile Coach" olarak güncelleyenler, orada mısınız?

56. Tahmin mi, Atmasyon mu?: Veriye Dayalı İyileştirme

Planlama etkinliklerinizde verdiğiniz efor tahminlerini bir yerlere kaydetmediğiniz ve bu tahminleri gerçekleşen eforlarla karşılaştırmadığınız sürece, **tahminleme konusunda iyileşemezsiniz.**

Türkçe'de ikisi için de aynı sözcüğü kullanıyoruz ama "guess" ile "estimate" arasında fark vardır.

> İş büyüklüklerini "guess" etmek yerine "estimate" etmek istiyorsanız **kayıt tutun.**
>
> Yoksa sonsuza kadar "guess" edersiniz.

57. Agile Sihirli Değnek Değil

Tam, açık, net ve müşterilerin sistem yardımıyla gerçekten ne başarmak istediğini açıklayan iyi gereksinimlere sahip olmak yazılım endüstrisinde hep bir zorluk olmuştur.

Öyle ki, Yazılım Mühendisliği'nin babası Frederick Brooks 1987'de yayınlanan *"No Silver Bullet: Essence and Accidents of Software Engineering"* makalesinde şöyle der:

> *"Bir yazılım sistemi geliştirmenin en zor kısmı tam olarak ne inşa edileceğine karar vermektir. Kavramsal çalışmanın başka hiçbir kısmı, insanlar için, makineler için ve diğer yazılım sistemleri için yapılan tüm arayüzler dahil olmak üzere ayrıntılı teknik gereklilikleri*

> *belirlemek kadar zor değildir.*
>
> *İşin başka hiç bir kısmı, yanlış yapılırsa ortaya çıkan sistemi gereksinimler kadar sakatlayamaz. Başka hiçbir parçanın daha sonra düzeltilmesi daha zor değildir."*

İyi gereksinimlerin eksikliğinden kaynaklanan sorunları hepimiz biliyoruz. İyi gereksinimlere sahip olmadığımızda karşımıza çıkan en temel sorun: **müşterilerin gerçekten ihtiyacı olan şeyleri ona veremeyen projelerdir.**

Tüm yazılım projelerinin iyi gereksinimlere ihtiyacı vardır ve çevik projeler de bu konuda farklı değildir. Agile, yeni gereksinim tekniklerini oyuna dahil eder ve iş süreçlerinin sahiplerinin yazılım geliştirme takımı ile yakın çalışmasına güçlü bir şekilde odaklanır. Müşterinin istedikleri ile gerçek ihtiyaçları arasındaki boşluğu daha iyi görebilmesi için ona belirli aralıklarda çalışır durumda yazılım göstermemizi öğütler.

Ancak;

Çoğu kuruluş, büyük projelerde ve kuruluş

genelinde karşılaştıkları gereksinim zorluklarının yalnızca Agile'a geçişle çözülmediğini görüyor. İyi gereksinimlerin elde edilmesiyle ilgili mevcut sorunlar devam ediyor.

Temel fark, Agile ile kısa döngülerde sorunlardan kaçmak zor olduğu için kurumların gereksinim sorunlarıyla **daha erken yüzleştiklerini** görmeleridir.

Geleneksel gereksinim tekniklerinden kullanıcı hikayelerine geçiş, **sihirli bir şekilde** daha iyi gereksinimlerle sonuçlanmaz. Çalışanlarınızın gereksinimleri ortaya çıkarma ve ifade etme noktasında işe yaradığı bilinen tekniklerde yetkin olması gerekir. Çalışanlarınızın becerilerinde boşluklar varsa, bunların doldurulması gerekir.

Agile'a geçerken ekiplerin karşılaştıkları mevcut gereksinim sorunlarını belirlemeleri, bunları ele almaya yardımcı olacak gereksinim uygulamalarını seçmeleri ve iş birimleri arasında önceliklendirme gibi organizasyon düzeyindeki sorunlarını ele almaları elzemdir.

58. En Anlaşılmayan Pratik: Acceptance Tests ve Gerçekler

Gördüğüm kadarıyla “Acceptance Tests” pratiği çevik pratikler arasında en anlaşılmayan pratiklerden biri.

İçerisinde bulunduğumuz neredeyse her danışmanlık projesinde takımların bu pratiği ya kullanmadığına ya da yanlış kullandığına şahit oluyoruz.

Bu durum biraz da garip, çünkü kabul testleri pratiğinin arkasındaki mantık oldukça basit.

Bu pratiğin öğütlediği şeyler:

- Gereksinimlerin ortaya çıkarılması noktasında business ile teknik ekiplerin yakın çalışması,

- Teknik ekip içerisindeki özellikle QA gibi rollerin gereksinimlerle ilgili çalışmalara dahil edilerek happy path dışındaki akışların ortaya çıkmasına yardımcı olması,

- Üzerinde anlaşılan gereksinimlerin formal bir dil kullanılarak -ve hatta otomatize edilmesi de mümkün olan- testler ile ifade edilmesi.

Tanımlamadığımız bir şeyi üretmemiz mümkün olmadığından elbette kabul testleri hazır olmayan bir story henüz “tanımlı/hazır” hale gelmemiştir. Scrum'da **Refinement aktivitesi** bunun için vardır.

İşi talep edenlerle işi yapacakların "ne" üretileceği konusunda "ortak bir anlayışa" erişmesi elzemdir. User story (kullanıcı hikayesi) formatının olmazsa olmaz parçalarından biri kabul testleridir.

> **Sadece 1 cümle ile ifade edilen bir kullanıcı hikayesini bir takımın sağlıklı bir şekilde implement etmesi mümkün değildir.**

Ve elbette iterasyon/sprint içerisinde kabul testlerini geçemeyen bir story henüz “bitti” olarak değerlendirilmemelidir.

Şimdi lütfen kullandığınız Jira vb. issue tracking uygulamasını açın ve aktif Sprint’inize konu olan maddelerin kaçının içerisinde kabul testlerinin düzgün bir şekilde ifade edilmiş olduğunu kontrol edin.

Eğer elinizdeki kayıtların detayı 1-2 cümleden öteye geçmiyorsa “kabul testleri” pratiğini doğru kullanmıyorsunuz demektir.

Buyrun size bir tane daha **“Nasıl Agile (Çevik) Olunmaz?”** durumu.

Bu durumu keşfettiğinize göre şimdi takım olarak bunu düzeltmek için neler yapabileceğinizi konuşmaya başlayabilirsiniz.

59. Renkli Post-it Sevdası

Üst yönetimler ile çevik dönüşümden beklentilerini ve dönüşümün yol haritasını konuşurken genelde olan şey budur.

Geçmişte dönüşüm danışmanlığı için görüştüğümüz Türkiye'nin çok büyük şirketlerinden birinin CIO'sunun toplantıda şöyle sormuşluğu vardır:

> *"Ya ben görüyorum başka firmaların ofislerinde duvarlarda hep post-it'ler var, rengarenk, çok güzel, bizde de olacak di mi öyle şeyler?"*

"*Olmaz mı, her tarafı sizin için post-it'lerle donatacağız"* desem gönlüm razı değil...

"İşin şekline neden bu kadar takılıyorsunuz? Olay renkli duvarlar mı?" desem CIO'yu mutsuz edeceğiz.

Dönüşümden beklentiniz renkli bir ofis mi gerçekten sevgili üst yönetim?

Maviden yeşile belli belirsiz bir geçiş de olsun mu?

60. Agile'ın Kalbi: XP

Clean Code yaklaşımının mucidi ve Agile Manifesto'nun eş yazarlarından **Robert C. Martin** diyor ki:

> *“Tüm Çevik süreçler arasında Extreme Programming (XP) en iyi tanımlanmış, en eksiksiz ve en az karışık olanıdır. Hemen hemen tüm diğer Çevik süreçler, XP'nin bir alt kümesi veya bir varyasyonudur.*
>
> *Bu, diğer Çevik süreçlerin gözardı edilmesi gerektiği anlamına gelmez. Aslında onları çeşitli projeler için değerli ve kullanışlı bulabilirsiniz.*
>
> *Ancak Agile'ın gerçekte neyle ilgili olduğunu anlamak istiyorsanız, XP’yi anlamaya çalışmaktan daha iyi bir yol yoktur.* ***XP, Çevik'in prototipi ve en iyi temsilcisidir.”***

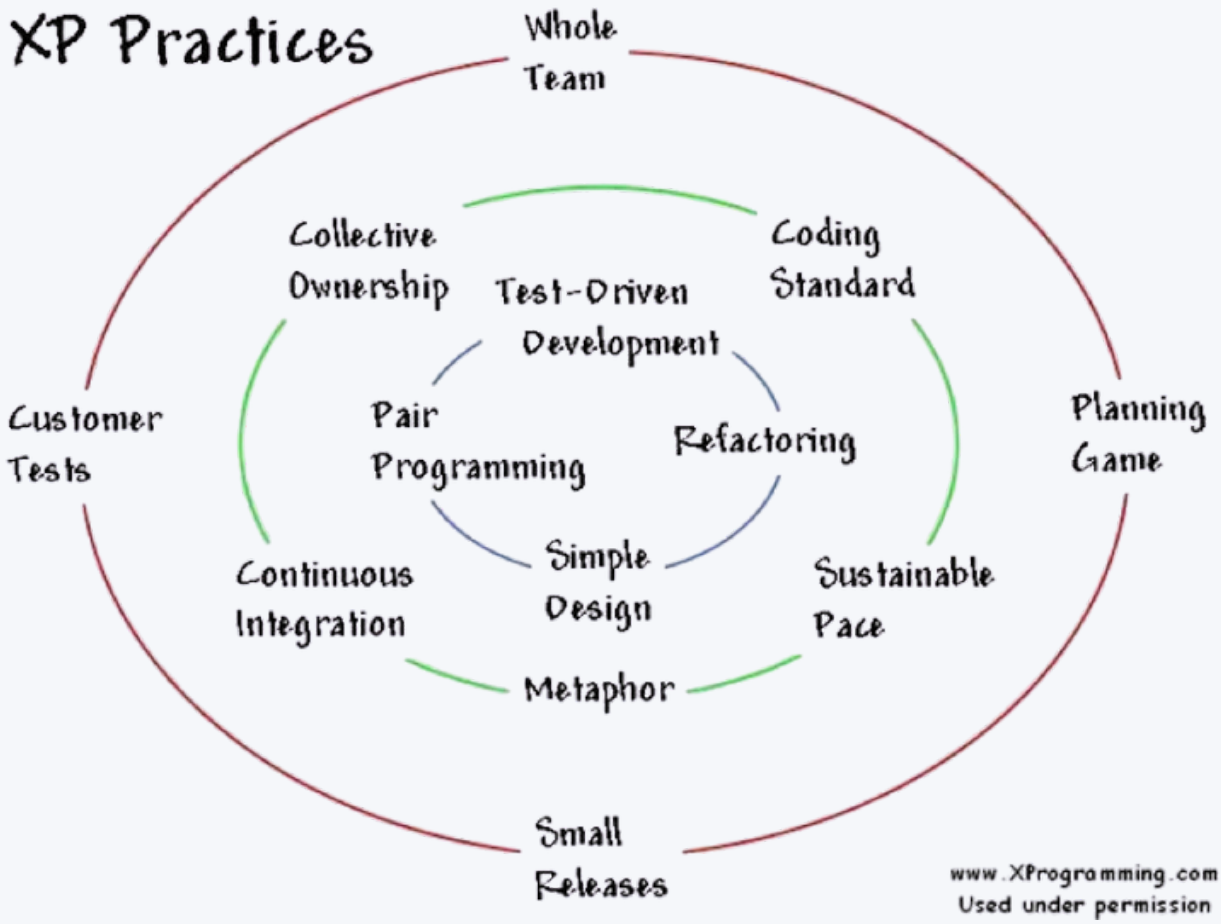
Dolayısıyla, Agile hakkında düşünen, konuşan, yazan birinin **XP’den hiç bahsetmemesi** konu hakkında pek bir şey bilmediğinin açık ispatıdır.

Sektördeki dostlarımızla sohbet ettiğimizde “abi bizim şirketteki agile koç hiçbir işe yaramıyor” diye sitem etmelerinin nedeni de budur.

Business pratikleri, takım pratikleri ve teknik pratikler hakkında yol gösteremeyen bir

koç/danışman size fayda sağlayamaz.

Agile nedir'i anlamak istiyorsanız görseldeki "**Circle of Life**"ı anlamanız gerekir.



61. Ölçemediğini Yönetemezsin

Bir takıma / organizasyona onları iyileştireceğinizi vaat ediyorsanız bunu **sayılarla ortaya koymak zorundasınız.**

Mevcut durumun ne olduğunu sayılarla ortaya koyup, hedef durumun ne olduğunu yine sayılarla ifade etmedikten sonra herkes bir şeyleri iyileştirdiğini iddia edebilir.

Dahası; sizi iyileşmenin ne olduğunu **görmemekle** suçlayabilir.

İşte çevik dönüşümlerin çoğunda olan şey de budur.

"Bak bir agile yapalım gör sen noluyor?"
diye başlarsanız işin sonunda kendinize:

"Eee noldu?" diye sorarsınız.

"Eee iyi de ne ölçelim?"

Buyrun size bazı öneriler:

Kategori	Alt Kategori	Metrik
Değer	Finansal	Gelir
Değer	Finansal	Gider
Değer	Finansal	Kâr
Değer	Finansal	Kullanıcı Kazanım Maliyeti
Değer	Memnuniyet	Kullanıcı Başına Düşen Şikayet
Değer	Memnuniyet	NTS
Değer	Memnuniyet	Müşteri Memnuniyeti Puanı
Değer	Memnuniyet	İç Müşteri NTS
Değer	Memnuniyet	Mağaza Puanı
Değer	Memnuniyet	Referral Oranı
Değer	Tüketim	İndirme Sayısı
Değer	Tüketim	Aktif Kullanıcı Sayısı
Değer	Tüketim	Geri Dönen Kullanıcı Sayısı
Değer	Tüketim	Kayıtlı Kullanıcı Sayısı
Değer	Tüketim	İşlem Sayısı
Değer	Tüketim	Satın Alma Oranı
Değer	Tüketim	Yüklenmiş Versiyon Indeksi
Üretim	Kalite	Test Kapsaması
Üretim	Kalite	Canlıdan Dönen Hata Sayısı
Takım	Mutluluk	Mutluluk Puanı (Niko Niko)
Takım	Uyum	Olgunluk Skoru

Üretim	Kalite	Kullanıcı Kabul Başarı Notu
Üretim	Kalite	Denek Sayısı
Üretim	Kalite	Statik Kod Analizi
Üretim	Kalite	Devreye Alma Başarısı
Üretim	Kalite	Güvenlik Testi Sonucu
Üretim	Üretkenlik	Velocity
Üretim	Üretkenlik	Talep Karşılama Süresi
Üretim	Üretkenlik	Taahhüt Uyum Oranı
Üretim	Üretkenlik	Yayın Sıklığı
Üretim	Üretkenlik	Müşteri Teslim Süresi
Üretim	Üretkenlik	Hata düzeltme süresi
Üretim	Yenilikçilik	Yenilik Oranı

62. Sektörün Sesi Olmak: Tarih Tekerrür Ediyor

"Nasıl Agile (Çevik) Olunmaz?" serisi artık resmen sektör ve şirketler konusunda anlık içgörüler elde etmemizi sağlıyor.

Mesela bugün aynı şirketten 2 çalışan serinin 3 ay önce yayınladığım çok eski bir postu olan 53. post'u aramış bulmuş, yeniden yayınlamış ve beğenmiş.

53. post'ta özetle ne demiştik:

> *"Yönetimin veya pazarlama departmanının söylediği (ve hatta dayattığı) bir teslim tarihi*

> *bir tahmin değil, bir hedeftir. Maalesef çoğunlukla gerçekçi olmayan hedefler belirlemek konusunda çok becerikli insanlar vardır ve bu arkadaşlar takımınızın önüne bu rasyonel olmayan hedefleri şak diye koyarlar."*

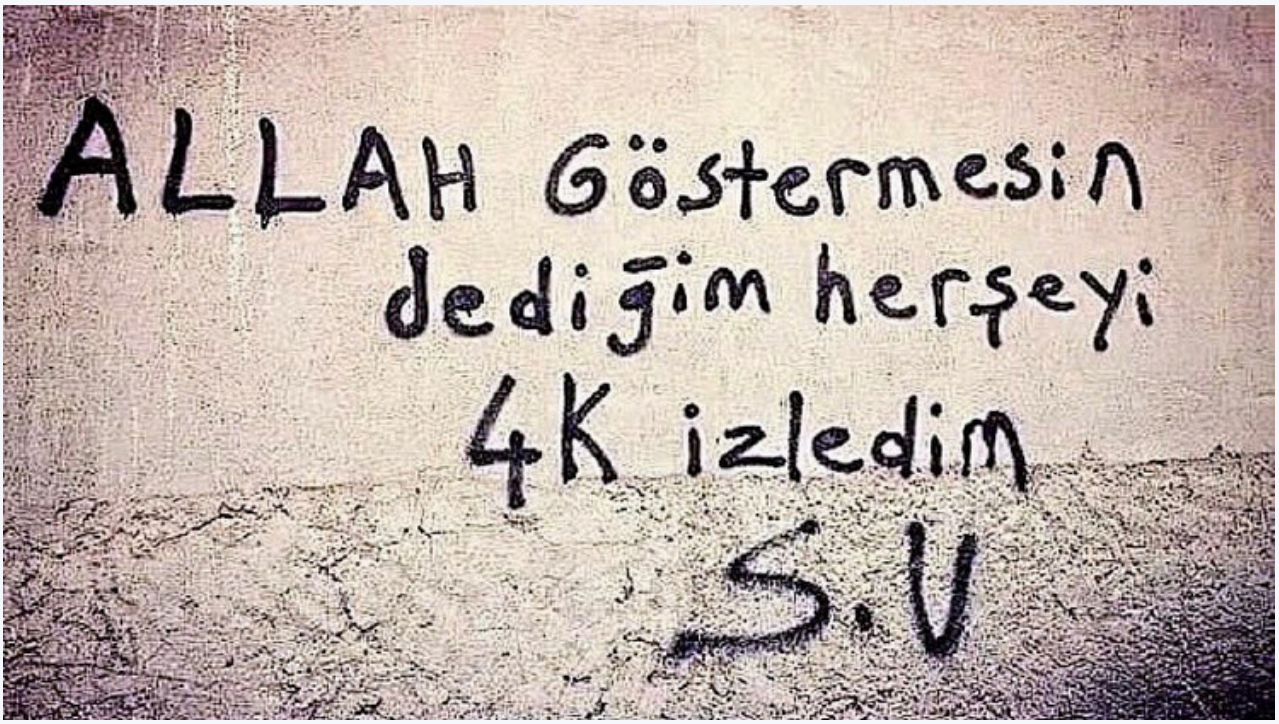
Tahminimce bu şirkette bu hafta yine rasyonel olmayan birileri, bu arkadaşlarımızın önüne şak diye koymuş imkansız hedefleri... Onlar da sitemlerini bu post ile dile getirmek istemişler herhalde. Durum gerçekten böyleyse ve duygularına tercüman olabilmişsek ne mutlu bizlere.

Kendimce inandığım şeyleri içine biraz da mizah katarak yazayım istemiştim seriye başlarken. İlgi gösteren ve destekleyen herkese bu vesile ile sonsuz teşekkürler. Seveni olduğu kadar sevmeyeni de var tabi bu serinin, onlara da geri bildirimleri için teşekkürler.

Ne diyeyim? Şirket isimleri değişse de bizim sektörde yaşananlar pek değişmiyor, tarih hep tekerrür ediyor demek ki.

O zaman 63. gönderide sektörün başka bir yarasında tekrar buluşmak üzere...

63. Göstermelik Dönüşümler ve Gerçekler



Çevik dönüşüm yaptığını söyleyen şirketlerin paylaşımlarını görünce hissettiklerimi anlatan bir görsel tam olarak bu...

Neler görmedik ki bu süreçte:

- "Agile Ormanı" dikenlerden tutun da... (Hatırlayanlar olacaktır o efsane paylaşımı)
- "Hocam biz bu işi çok iyi becerdik, daily'lerimiz 15 dakikayı asla aşmıyor, başka şirketlerde yarım saat süren daily'ler var." diye bununla övünenlere kadar...

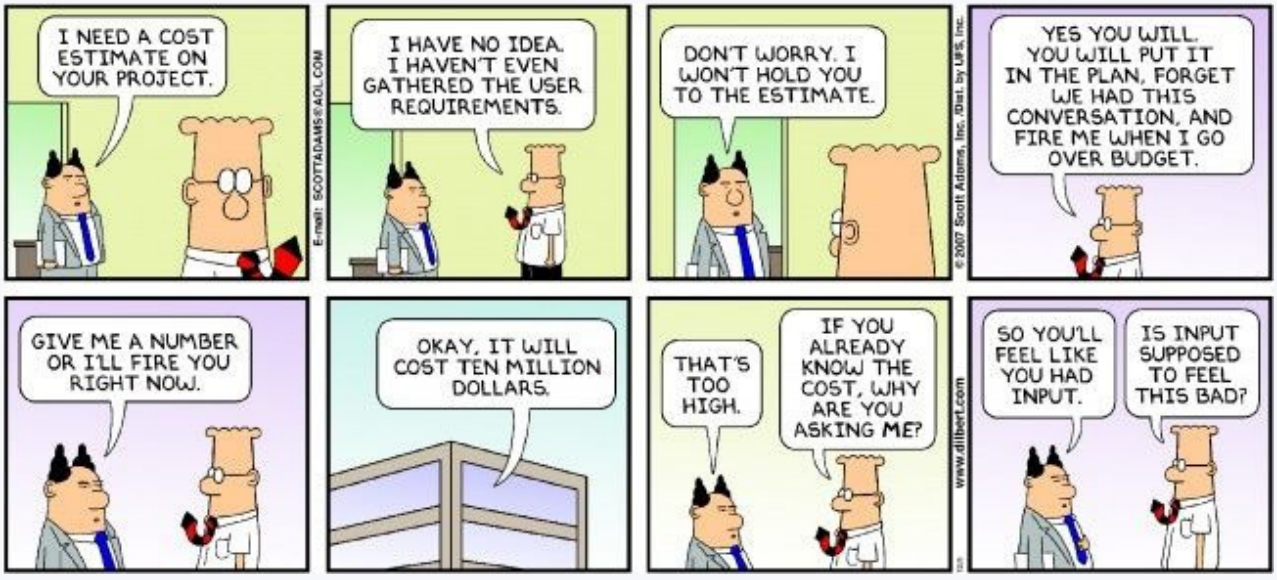
Bir olay bir miktar yanlış anlaşılabilir, hadi buna tamam...

Ama bu kadar da olmaz ki be arkadaş!

Gerçekten sayenizde;

"Allah göstermesin dediğimiz her şeyi 4K izledik."

64. Yazılımda Süre Öngörüsü Paradoksu



Bir yazılım projesinin tamamlanma zamanını nasıl tahmin edersiniz?

Basit cevap, projeyi oluşturan alt parçalara ayırmanız ve ardından bu parçaları tahmin

etmenizdir.Örneğin, çevik yaklaşımlarla çalışan takımlar ellerindeki herhangi bir büyük işi önce epic'lere, ardından story'lere ayırarak bunu yaparlar.

Bu gayet iyi bir yaklaşımdır; ancak ya ayırdığınız bu parçalar doğru tahmin edilemeyecek kadar büyükse ne yapacaksanız?

O zaman bu parçaları daha küçük parçalara ayırırsınız ve tahmin etmeye devam edersiniz. Eminim hepiniz buradaki **özyinelemeli (recursive)** durumu fark ediyorsunuz.

Bu yaklaşımı hangi seviyeye kadar aşağı taşıyabilirsiniz? Bu yaklaşımı kod satırlarına kadar taşıyabilirsiniz. Aslında, programcıların yaptığı şey budur. Programcı, bir görevi kod satırlarına ayırma becerisine sahip kişidir.

Buradan çıkan sonuç nedir?

> Bir projenin doğru ve kesin bir tahminini istiyorsanız, onu yazılacak kod satırlarına kadar ayırın. Bunu yapmak için harcadığınız zaman, projeyi inşa etmenin ne kadar

> süreceğini size çok doğru ve kesin bir şekilde verecektir - çünkü bunu yaptığınızda zaten projenizi yapmış olacaksınız.

Tabii ki, bu yaklaşım bir “tahminin” asıl noktasını kaçırıyor. Tahmin bir tahmindir; projeyi gerçekten inşa etmeden projenin ne kadar süreceğine dair ortaya koyduğumuz bir fikirdir.

Çoğu zaman tahmin maliyetinin düşük olmasını isteriz. Bu nedenle, bir tahmin, tanımı gereği zaten **“kesin”** değildir. Bu kesin olmama durumu, tahmini oluşturmak için gereken süreyi kısaltmamıza olanak tanır. Kesinlik ne kadar az olursa, tahmin o kadar az zaman alacaktır.

Tabi buraya kadar söylediklerimiz, bir tahminin tamamen yanlış olması gerektiği anlamına da gelmiyor. Tahminler mümkün olduğu kadar doğru olmalı, ancak tahminin maliyetini düşük tutmak için gerektiği kadar doğru olmalıdır.

İşte size bir örnek:

> Ölüm anımı önümüzdeki bin yıl içinde bir zaman olarak tahmin ediyorum. Bu tamamen **doğrudur**, ancak çok **kesin değildir**. Bu

> doğru tahmini oluşturmak neredeyse hiç zamanımı almadı çünkü buradaki kesinlik çok düşük.

Yeterince doğru bir tahmin, tahmin edilen olayın neredeyse kesin olarak gerçekleşeceği bir zaman aralığını belirtir. Yazılım takımları için taktik, doğru olacak en küçük aralığı seçmek için mümkün olan en az miktarda zaman harcamaktır.

İşte bu yüzden sevgili üst yönetimler ve iş birimleri;

"Bizim proje ne zaman biter?" diye sorduğunuzda size çok geniş olmayan bir **“aralık”** söyleyen takımlarınıza güvenin. Onlar ne yaptıklarını biliyorlardır.

Hee yok, *"bana günü gününe saati saatine bir tahmin verilsin"* istiyorsanız mevcutta harcadıklarından **12.068 kat eforla** size o tahmini verirler.

Ama sonra bana gelip *"İstediğimiz bir işin tahminini bile vermeleri 3 ay sürüyor."* demeyin.

65. Scrum Master: Gönüllü Değil, ”Usta” Olmalı

Bana en çok sorulan sorulardan biridir:

> *"Hocam Scrum Master nasıl bir profil olmalı?"*

Scrum Guide diyor ki:

> *"The Scrum Master is accountable for the Scrum Team’s effectiveness. They do this by enabling the Scrum Team to improve its practices, within the Scrum framework."*

Yani bu arkadaş ekibi geliştirmek üzere çaba sarf etmeli.

Tek başına çaba yeter mi?

Elbette yetmez, yetkin olmayan bir kişi takımı da geliştiremez. Bu arkadaş **iyi bir mühendis olmalı**, takımı iyi (mühendislik) pratikleri konusunda geliştirmeli.

Bazı danışmanların sizi yönlendirdiği gibi "Aramızdan bir gönüllüyü Scrum Master olarak seçelim" şeklinde ilerlerseniz muhtemelen ekipteki hevesli ama yetkin olmayan bir arkadaşı Scrum Master yapmış olursunuz.

Yapmayın bunu. Bu sorumluluğun adı üzerinde **"Master"**.

Yaparsanız ne olur? Doğru olanı yapmak yerine, kolay olanı yapmış olursunuz. İşte yine Agile'ın şekli var. Özü kayıp.

Yaşar Safkan Hoca'nın dediği gibi:

> *"Hoppala paşam, Türk kaşığıyla Amerikan çikolatası."*

(Bu arada kendisinin "Agile: Türk Kaşığıyla Amerikan Çikolatası" yazısını hala okumadıysanız işi gücü 5 dakika bırakın ve safkan.org'dan okuyun derim.)

66. Tartışmalı Ama Gerekli: Eşli Programlama Gerçeği

Pair programming (eşli programlama) pratiği, yıllardan beri önemli miktarda tartışmayı beraberinde getirdi.

İlk bakışta çoğu kişi, iki (veya daha fazla) kişinin aynı problem üzerinde verimli bir şekilde birlikte çalışabileceği fikrine olumsuz tepki vermeye devam ediyor. Geçen hafta verdiğim bir eğitimde de yine sınıfla uzun uzun bu konuyu tartıştık.

Her şeyden önce; **eşli programlama isteğe bağlı ve aralıklı bir pratiktir.**

Kimse eşli programlama yapmaya zorlanmamalı. Zaman zaman geliştiricilerin tek başına kodlama yapmaları için pek çok iyi neden var. Bir takım yaklaşık olarak vaktinin %50'sini eşli programlama ile geçirebilir. (Buradaki değer kritik değil. %35 veya %75 de olabilir.)

Eşli programlama;

- Takım üyeleri arasında bilgi paylaşmanın ve bilgi

silolarının oluşmasını önlemenin açık ara en iyi yoludur.
- Aynı zamanda takımdaki hiç kimsenin vazgeçilmez olmadığından emin olmanın en iyi yoludur.
- *"Onu yapan arkadaş askere gitti, takımdaki başka kimse de bu konuyu bilmiyor"* söylemleri ile karşılaşmanızı engellemenin de bilinen en iyi yoludur.

Birçok takım, eşli programlamanın **hataları azalttığını ve tasarım kalitesini iyileştirdiğini** belirtiyor. Bu görüş de muhtemelen çoğu durumda doğrudur. Herhangi bir sorun üzerinde birden fazla göze ve beyine sahip olmak genellikle iyidir.

Gerçekten de, birçok takım senkron olmayan kod gözden geçirmeleri (code review) eşli programlama ile değiştirdiğini söylüyor.

"Ne gerek var yahu, 2 kişiye 1 iş yaptıracağımıza, 2 kişiye 2 iş yaptırırız, üretkenliğimiz 2 katına çıkar" diyenlere **Çevik Manifesto'nun 8. İlkesi**ni hatırlatalım:

> *"Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve*

> *kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.”*

Kısa vadede üretkenliği 2 katına çıkartmak mümkün ama üretkenliği sabite yakın tutmak o kadar da kolay değil. Pair programming, **MOB Programming** (çete programlama) gibi pratikler bunun için önemli.

Görseldeki gibi fiziksel bir çalışma koltuğu (işin esprisi tabi) veya uzaktan çalışma araçları (Zoom, Teams vs.) ile bu pratikleri destekleyin.

67. Geliştirme ve Testin ”Blender”ı

Takımlarınız "testi" geliştirmeden sonra başlayacak bir aktivite olarak görüyorsa **çeviklik uğruna yaptığınız her şeyi yeniden gözden geçirin**.

Kalite nasıl elde edilir?

> *"Kalite, geliştirme ve testin bir blendera koyulması ve biri diğerinden ayırt edilemez hale gelinceye kadar karıştırılmasıyla elde edilir."*
>
> *"Quality is achieved by putting development and testing into a blender and mixing them until one is indistinguishable from the other."*

-How Google Tests Software-

68. ”Software” Kelimesinin Hakkını Vermek

"**Software**" birleşik bir kelimedir.

- "**Ware**" kelimesi "ürün" anlamına gelir.
- "**Soft**" kelimesi yumuşak yani değiştirilmesi, yeniden şekillendirilmesi kolay anlamına gelir.

Demek ki yazılım değiştirilmesi kolay bir üründür.

Yazılım, makinelerimizin davranışını hızlı ve kolay bir şekilde değiştirmenin bir yolunu istediğimiz için icat edildi. Makinenin davranışlarını değiştirmenin

zor olduğu bir şeyden bahsettiğimizde buna **"hardware"** adını veriyoruz.

> **"Software must be soft!" (Yazılım yumuşak olmalıdır!)** söylemi bu mesajı verir.

Geliştirme takımları genellikle değişen gereksinimlerden şikayet ederler.

- *"Bu değişiklik mimarimizi tamamen bozuyor."*
- *"Bu değişiklik bize ciddi test maliyeti çıkarır."*

gibi ifadeleri sık sık duyarız.

Bir dakika: Gereksinimlerdeki bir değişiklik mimarinizi tamamen bozuyorsa belki de mimariniz yeterince iyi değildir. Test maliyetlerinizin yüksek olması testleri manuel gerçekleştirmenizden kaynaklanıyor olabilir mi?

Geliştirme takımları olarak değişimi kutlamalıyız çünkü bu yüzden buradayız. **Agile Manifesto'nun 2. ilkesi** bize bunu öğütler:

> *"Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanır."*

Değişen gereksinimler, agile dediğimiz şeyin ortaya çıkış nedenidir. Bütün oyun bunun üzerine kuruludur. **Martin Fowler** bu konuda şöyle diyor:

> *"Gittiğim her sorunlu projede duyduğum bir nakarat var. Geliştiriciler bana gelip 'Bu projenin sorunu gereksinimlerin sürekli değişmesi' diyorlar. Bu durumda şaşırtıcı bulduğum şey, herhangi birinin buna şaşırmasıdır. İş yazılımı geliştirmede gereksinim değişiklikleri normdur (kuraldır), asıl soru bizim bu konuda ne yaptığımızdır."*

Değişen gereksinimler yazılım profesyonelleri olarak kariyerlerimizin ve maaşlarımızın gerekçesidir. İşlerimiz, değişen gereksinimleri kabul etme, tasarlama ve bu değişiklikleri nispeten ucuz hale getirme becerimize bağlıdır.

Agile, değişimi ucuzlatmaktır.

- Yüksek kalitede kod değişimi ucuzlatır. (Agile Manifesto'nun 9. İlkesi)
- Yetkin ve motive takım üyeleri değişimi ucuzlatır. (Agile Manifesto'nun 1. Değeri ve 5. İlkesi)
- Makinenin yapabileceği şeyleri insana yaptırmamak (otomasyon) değişimi ucuzlatır.

(Agile Manifesto'nun 8. İlkesi)
- Müşteri ile etkin iletişim değişimi ucuzlatır. (Agile Manifesto'nun 4. İlkesi)
- Takım içerisindeki etkin iletişim değişimi ucuzlatır. (Agile Manifesto'nun 6. İlkesi)
- Değişimi ucuzlatmak için kafa yormak ve yeni yollar denemek muhtemelen günün birinde değişimi ucuzlatır. (Agile Manifesto'nun 12. İlkesi)
- Müşteriye erken ve sürekli çalışan yazılım sunmak değişen gereksinimleri daha kolay ve erken keşfetmenizi sağlayacağı için değişimi ucuzlatır. (Agile Manifesto'nun 1. ve 3. İlkesi)

Şimdi dönün ve çeviklik namına kullandığınız bütün pratikleri gözden geçirin, siz hangi pratikler ile değişimi ucuzlatıyorsunuz?

69. Agile Sadece Cesurların İşidir

Karşılaştığımız profesyonel yazılım geliştirme ekiplerinin çoğu işlerini nasıl yaptıkları konusunda deney yapmaktan, yeni yollar denemekten pek hoşlanmıyorlar.

Bir şirkete çevik yöntemler konusunda yardımcı olmak için dahil olduğumuzda gördüğümüz şey

genellikle; ekiplerdeki kişilerin ilk şirketlerinden öğrendikleri ve kendilerine uygun buldukları pratikleri benimseyip bunların dışına çıkmama eğilimde olmaları.

Bunun sebebi de oldukça açık:

> *"Farklı pratikleri denemek acı vericidir ve başlangıçta üretkenliği azaltır. Üzerinizde teslimat baskısı varken üretkenliğin azalması isteyeceğiniz bir şey değildir."*

Ancak üst seviye performans gösteren tüm ekiplerin ortak özelliğinin **"yeni pratikleri araştırmaya, öğrenmeye, uygulamaya vakit ayırması"** olduğu da bir gerçek.

Ekiplerinize yeni pratikleri araştırmaları, öğrenmeleri, uygulamaları için alan açmalısınız. Ben söylemiyorum, **Agile Manifesto 12. ilkesi**nde söylüyor

Scrum Master, Kanban Master, XP Coach, Agile Coach, Enterprise Agile Coach, Organizational Agile Coach (ne demekse), DAC gibi rollerden artık sizde hangisi varsa ekiplere bu alanı açmak için savaşmalı.

Evet sevgili üst yönetimler;

Bu rollere sizinle savaşmaları için maaş ödemelisiniz.

Ben de çabuk çorba seven birisi olarak diyorum ki; belki de bu yüzden **yalnızca cesurların işidir agile.**

70. Eczaneden Tavuk İstemek: Çeviklik Ne Değildir?

Türkiye'de yazılım ekiplerine liderlik eden çoğu kişinin çevik yazılım geliştirmeden anladığı ve çeviklik için niyetlendiği şeyler, çevik yazılım geliştirme fikrini ortaya çıkaran **17 kişinin niyetiyle ve anlayışıyla kesinlikle aynı çizgide değil.**

Senelerdir anlatmaya çalışmamıza rağmen halen:

- *"Unit testing ile çevikliğin ne alakası var?"* diye soranlar var.
- *"Biz çeviklik konuşuyorduk, konu nereden pair programming'e geldi?"* diye çıkışanlar var.

Peki *"çeviklik ekseninde sizinle ne konuşalım?"* diye sorunca da gelen cevaplar şunlar oluyor:

- *Sprint süremiz 2 hafta mı olmalı, 3 hafta mı?*
- *Takımların story point tahminlerini doğru verdiğinden nasıl emin olacağız?*
- *Kaynakları(!) doğru planladığımızdan, doğru doldurduğumuzdan nasıl emin olacağız?*
- *Bu takımdan ben yeterince çıktı alıyor muyum? Daha fazla çıktı verebilirler mi? Bizim takımların velocity'si düşük mü?*

Bu sorular bana her zaman şu fıkrayı hatırlatıyor:

> Temel bir gün eczaneye girer ve sorar; - *“Tavuk var mi hemşerim?”*
>
> Eczacı; - *“Yok hemşerim burası eczane ne arasın burada tavuk?”*
>
> Temel çıkar gider, ertesi gün tekrar gelir; - *“Tavuk var mi hemşerim?”* diye tekrar sorar.
>
> Eczane sahibi öfkeli bir sesle; - *“Yok kardeşim burası eczane sana burada tavuk olmaz dedim ya git işine!”* diye çıkışır.
>
> Temel çıkar gider.. Ertesi gün tekrar gelir; - *“Tavuk var mi hemşerim?”* diye yine sorar.

Eczacı çok kızmıştır artık Temele bağırarak; - *“Yok dedik ya kardeşim ne laftan anlamaz bir adamsın burası eczane burda tavuk olmaz git bir daha da rahatsız etme beni!”* der.

Temel; - *“Tamam da kardaşum ne kızayisun madem tavuk yoktur camuna yazsana tavuk yoktur diye!”* der ve eczaneden çıkar gider.

Eczacı bakar ki bundan kurtuluş yok tekrar gelmesin diye koca bir kağıda **"TAVUK YOKTUR"** diye yazar ve eczanenin camına yapıştırır.

Ertesi gün Temel tekrar gelir ve eczacıya sorar; - *“Tavuk ne zaman gelecek hemşerim?”*

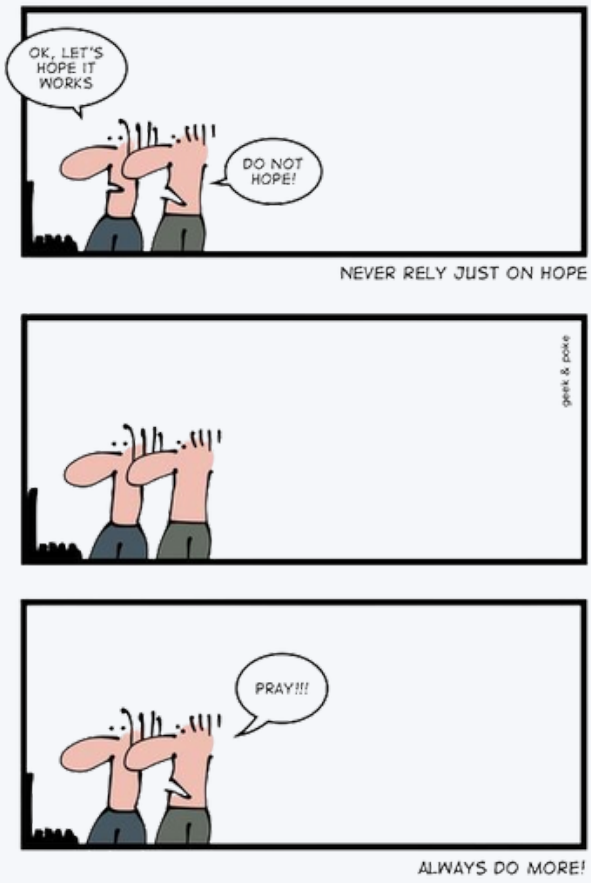
Eczaneden tavuk alamazsınız!

Derdinizi doğru ifade edin ve çözümü doğru yerde arayın.

71. “Umut ve Dua” Yöntemi: Yazılım Projesi Nasıl Yönetilir?

Bir yazılım projesini nasıl yönetirsiniz?

Bu sorunun yanıtı olabilecek pek çok yaklaşım yıllar boyunca denendi.



En sık gördüğümüz yaklaşım: Umut ve Dua'dır. Yazılım projelerinin kaderini yöneten tanrıların olduğuna inanan yöneticiler arasında oldukça popülerdir.

Böyle bir inanca sahip olmayanlar genellikle; kırbaç, zincir, kızgın yağ ve kayalara tırmanan insanların ve okyanus üzerinde uçan martıların resimleriyle takımlarını zorlamak gibi **motive edici(!) yaklaşımlar**a başvururlar.

Bu yaklaşımlar yanlış yönetiminin (neredeyse evrensel diyebileceğimiz) karakteristik semptomlarını ortaya çıkarır:

- Çok fazla mesai yapmalarına rağmen her zaman geç kalan geliştirme ekipleri.
- Müşterinin ihtiyaçlarını karşılamaya yaklaşmayan kalitesiz ürünler üreten ekipler.

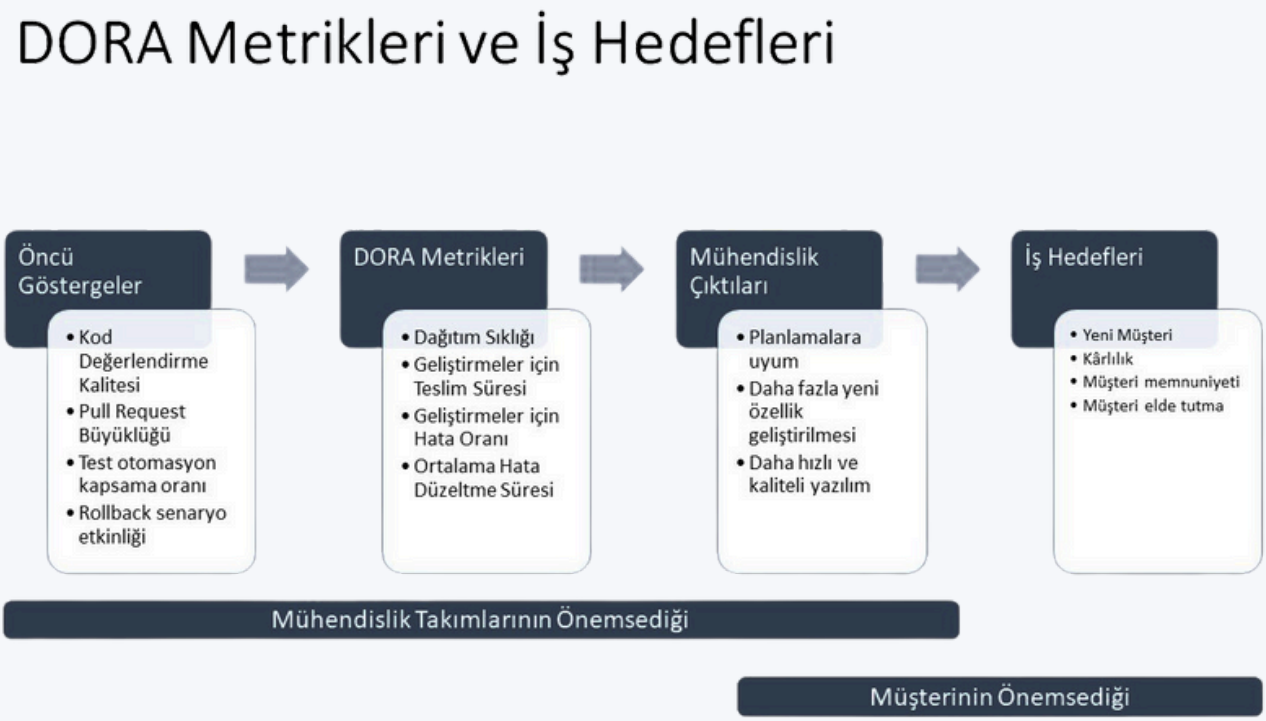
Doğru uygulandığında Agile sizi bunlardan uzak tutar.

Velocity gibi basit metriklerle çoğu zaman hayal ettiğiniz şeyi hayal ettiğiniz sürede yapamayacağınız gerçeğini yüzünüze vurur. Size projeniz ve takımınızla ilgili haber getirir.

Çoğunlukla bu haberler "kötü" haberlerdir. Asıl konu aldığınız bu kötü haberler karşısında ne yaptığınızdır.

Sahi sizin şirketiniz bir yazılım projesini nasıl yönetiyor?

72. Müşteri Sonuca Bakar



Yazılım takımlarının müşterileri, takımın kullandığı yazılım geliştirme yaşam döngüsü modeli veya seçilen teknolojiler ile ilgilenmezler.

Müşterilerin önemsediği şey iş çıktılarıdır:

- Ne kadar yeni müşteri kazanabildik?"
- "Karlılığımız ne durumda?"
- "Müşterilerimiz memnun mu?"
- "Müşterilerimiz rakiplerimizden birine kaçıyor mu?"

Bu iş çıktılarını önemseyen müşteriler karşılarındaki yazılım takımlarından planlara uygun

çalışan yazılım teslimatı beklerler. Onlar için daha kısa sürede daha fazla yeni özellik geliştirmenizi isterler. Geliştirdiğiniz özelliklerin hatasıza yakın çalışmasını önemserler.

Müşterilerinizin bu beklentisini karşılamak için takım olarak **DORA metrikleri**ni ölçmeniz ve bunları sürekli olarak iyileştirmeye çalışmanız gerekir.

Her bir DORA metriğini etkileyen birden fazla **öncü (leading)** metrik vardır. Bir DORA metriğini iyileştirmek için bu öncü metrikleri de ölçmeniz ve iyileştirmeniz gerekir.

Bir yazılım takımı çevikliği; Agile Manifesto'yu duvara asarak, eskiden Genel Müdür Yardımcılığı dediği fonksiyonel yapılara *"Garaj"* diyerek elde etmez.

Müşterisinin önemsediği şeylere etki eden metrikleri ölçerek ve sürekli iyileştirerek elde eder.

Sizin takım hangi metrikleri ölçüyor?

73. 17 Beyaz Adamın İsyanından, Oyun Ablalığına

Organizasyonların Scrum Master, Agile Coach veya Agile danışmanı seçimlerini sürekli neden eleştirdiğimi soranlar oluyor. Sebebini buradan açıklayayım:

Sene 2001, Yer Snowbird-Utah, 17 Beyaz Adamın arasında geçen muhabbet...

"Yazılım geliştirmenin doğası hakkında hiçbir şey anlamayan insanlar bize nasıl çalışmamız gerektiğini anlatıyor ama önerdikleri şeyler hiçbir işe yaramıyor. Yazılım geliştiricilerin işlerinin kontrolünü ele almaları gerekiyor. Bunu nasıl yapabiliriz?"

Sene 2023, Yer Türkiye, organizasyonların çoğunda Scrum Master, Agile Coach veya Agile Danışmanı olarak çalışan insanlar:

"Yazılım Geliştirme konusunda hiçbir şey anlamamıza gerek yok. Geliştiricilerin aslında nasıl düzgün çalışacaklarını anlayabilmeleri için, onlara KOÇLUK yapacak birine ihtiyacı var.

Automated testing veya devops pratiklerinden banane, ben GÜÇLÜ SORULAR sormak için buradayım. Bir iki retrospective toplantısında da eğlenceli oyunlar oynatırım, olur biter."

74. Garanti Yok, Deney Var: İyileşmenin Tek Yolu

Birlikte çalıştığımız çevik takımlara bazen iş yapış şekillerinde "yeni" bir şey denemeleri için önerilerde bulunuruz.

Çoğu zaman takımdaki birkaç üye önerdiğimiz bu "yeni" şeyin çalışıp çalışmayacağından emin olup olmadığımızı sorar.

Peki biz bu "yeni" şeyin işe yarayacağından emin miyiz?

Kesinlikle hayır.

Peki böyle bir durumda ne yapıyoruz?

Takıma dönüp şunu söylüyoruz:

> *"Önerdiğim bu 'yeni' şeyin çalışıp çalışmayacağını bilmiyorum.*

> *Peki siz neyin hiç bir zaman çalışmayacağını biliyor musunuz?*
>
> *'Yeni' bir şeyler denememek."*

Denemekten korkarsanız iyileşemezsiniz.

Çevikliğe giden yol küçük deneylerden oluşuyor.

75. Yazılımın 3 Büyük Düşmanı

Yazılım ürünü geliştirmede karşılaştığımız en büyük sorunlar:

1. **Problemi yanlış anlamak:** Bu durum genellikle problem çözücülerin çok fazla aracı kişiyle çalışması nedeniyle ortaya çıkıyor. Kulaktan kulağa oyununa devam ettiğimiz sürece doğru çözümü ortaya çıkarmak güçleşiyor.
2. **Geribildirim almadan çok uzun süre çalışmak:** Bu aşırı iyimserliğin bir biçimi. "Karanlığa doğru gitmek" olarak da ifade edebiliriz.
3. **Ezici karmaşıklık:** Sistemler basit başlar ve sonunda büyük, anlaşılmaz karmaşalara dönüşür. Sürekli düzeltme olmadan bu durum daha da kötüye gider.

Bu 3 durum göz önüne alındığında Çevikliğin bize öğütlediği şeylere ulaşırız:

1. **Aracıları olabildiğince azaltın:** İşi talep edenin sesini işi yapacak insanların kulağına yaklaştırın. Geliştiriciler mümkün olduğunca sorunu yaşayan kişilerle doğrudan işbirliği yapmalı ve bunu yaparken etkili olacak bağlam ve desteği sağlamalıdır.
2. **Küçük adımlarla çalışın ve geribildirim alın:** Daha küçük adımlar atın. Her adımı test edin. Yazılım ürünleri geliştirirken kendinize yapabileceğiniz en büyük kötülük büyük bir şeyi tek seferde yapmaya çalışmaktır.
3. **Daha basit tasarımları tercih edin ve tasarım geliştikçe sadeliği korumak için temizlik yapın:** Refactoring'in XP'nin en önemli pratiklerinden biri olmasının nedeni tam olarak bu madde.

76. İK'da Çeviklik ve 3 Boyut

Gelin çevik yazılım geliştirmenin odaklandığı **"doğru ürünü yapmak", "ürünü doğru yapmak (kalite)"** ve **"hızlı yapmak"** üçlüsünü herhangi bir iş alanına uyarlayalım.

Çevik düşüncede en fazla önemsenen şey müşteri

memnuniyeti. (Bkz. Agile Manifesto İlke 1)

O zaman başlamamız gereken yer burası. Hangi iş alanında olursanız olun, kendinize önce şu soruları sormanız lazım:

“Müşterim kim?”
“Ona sağlamam gereken değer / fayda ne?”

Örneğin bir İşe Alım Takımıysanız müşteriniz organizasyonunuzda açık pozisyonları bulunan departmanlardır.

1. Doğru Ürünü Yapmak (Değer)

Bu departmanlar için doğru adayları bulur veya uygun olmayan adayları süreçten elerseniz onlar için değer üretmiş olursunuz. Demek ki çevik yazılım geliştirmedeki “doğru ürünü yaptık mı?” meselesi bir işe alım takımı için **“doğru adayı bulduk mu?”** halini alıyor.

Pek tabi bu aynı zamanda ölçmemiz gereken bir şey:

- Yerleştirdiğimiz adayların ne kadarı ile

organizasyon 2 aylık deneme süresi sonunda yolları ayırma kararı alıyor?
- İşe yerleştirdiğimiz adayların hedef gerçekleştirme rasyoları ne durumda?

2. Ürünü Doğru Yapmak (Kalite)

Çevik yazılım geliştirmede 2. boyutumuz "ürünü doğru yapmak" yani kalite. Bu örnekte bu boyutu **"İşe alım sürecimize dahil olan paydaşlara nasıl bir deneyim yaşatıyoruz?"** şeklinde ele alabiliriz.

Bu boyutta nasıl iş çıkarttığımızı şu sorularla ölçebiliriz:

- "Sürece dahil olan adaylar ve yöneticiler sunduğumuz deneyimden ne kadar memnunlar?"
- "Süreci olumsuz sonuçlanan adaylarla nasıl iletişim kurduk?"
- "Süreç içinde tüm paydaşlar için yaptığımız bilgilendirmeler yeterli miydi?"

3. Hızlı Yapmak (Hız)

Ve son olarak "hızlı yapmak". Bu boyut zaten kendini anlatıyor.

- “Bir personel talep formu önümüze geldiği andan pozisyonun kapandığı ana kadar ne kadar zaman geçiyor?”
- “Doğru bir adayı ne kadar sürede alabiliyoruz ya da uygun olmayan bir adayı hızlıca eleyebiliyor muyuz?”

Demek ki bir işe alım takımı için çeviklik bu 3 boyutu sürekli olarak iyileştirmeye çalışmak anlamına geliyor.

Peki nasıl?

Yanıt belli: Otomasyon ile, yapay zeka ile, süreçlerimizde değişiklik yaparak. **Bugün yapmadığımız farklı şeyler deneyerek.**

Çeviklik; sadece Scrum, Kanban gibi metotları uygulamaktan ibaret değil. Hatta bu metotlar yukarıda bahsettiğimiz 3 boyuta hizmet etmiyorsa anlamsız ve gereksiz.

Odaklanmamız gereken şey kendi alanımız için bu 3 boyutu nasıl iyileştirebileceğimiz. Hatta belki de bir süre sonra; *"İşe alım takımı yerine her fonksiyon kendi işe alımını kendi yapsa bu 3*

boyutta çok daha iyi bir hale gelebiliriz" gibi bir sonuca ulaşabilirsiniz.

Yani çeviklik yeri geldiğinde kendinden de vazgeçmeyi gerektirebiliyor.

Farklı iş alanlarında “çevikliğin” peşinde olanlara bir başlangıç noktası sunması temennisiyle.

77. Gerçek Çeviklik ve Mühendislik Pratikleri

Yazılım geliştirmede gerçek çeviklik için sağlam mühendislik pratiklerine ihtiyacınız vardır.

Bizimle benzer işler yapan pek çok eğitim/danışmanlık şirketi maalesef bunları yeterince vurgulamıyor. Neden? Çünkü pek çok danışmanın/eğitmenin yazılım geliştirme konusundaki tecrübesi sıfıra yakın. Hayatında "Hello World" yazmamış insanlar Scrum ile nasıl çevik olunabileceğini, Kanban'ın ne kadar muhteşem bir yöntem olduğunu anlatıyor, satıyor.

Eğitim ya da danışmanlık verdiğimiz pek çok kurumda "Daha önce de çeviklik konusunda eğitim

m ve/veya danışmanlık aldık; ancak hiçbiri sizinki kadar iyi değildi." söylemlerini duymamızın en önemli sebebi bence bu.

Eğer çeviklik mühendislik pratiklerini gözden kaçırırsanız, çevik yazılım geliştirmenin size sağlayabileceği "üretkenlik" ve "yanıt verebilirlik" faydalarını elde edemezsiniz (ya da çok azını elde edebilirsiniz). Extreme Programming'in çevik yöntemler arasında en değerlisi olduğunu düşünmemin nedenlerinden biri de bu.

Çeviklik için size Lego ile Scrum, bozuk para oyunu ile Kanban anlatılan eğitimlerden daha fazlasına ihtiyacınız var. Ancak bu yolda kan var, ter var, gözyaşı var.

Bunlar ilginizi çekiyorsa arayın, görüşelim.

> *"Bunlar bize gelmez hocam, kimi gönderelim 5 gün TDD eğitimine, biz 2 haftalık sprintlerde planladığımızın yarısını deliver etmeye devam edelim."*

diyorsanız, size başarılar dilerim.

78. Agile Nedir? En Net Cevap: 4 Adımda Çeviklik

Agile konusundaki her sohbette bize gelen bir soru: *"Yaw nedir şu Agile, her kafadan bir ses çıkıyor, şunu birkaç cümle ile açıklar mısın?"*

Buradan bir kez daha açıklayayım:

1. Küçük bir şey inşa edin,
2. Bunu kullanıcıya sunun,
3. Geribildirim alın,
4. Aldığınız geribildirimle ürettiğiniz küçük şeyi değiştirin ve/veya 1 nolu adıma geri dönün ve sonsuza kadar (ya da en azından biri size artık inşa ettiğin şeyler değer oluşturmuyor diyene kadar) tekrarlayın.

Bu 4 adımı gerçekleştirebiliyorsanız çeviksiniz, gerçekleştiremiyorsanız üzgünüm ama çevik falan değilsiniz.Hikaye bu kadar basit ancak bu 4 adımı sürekli olarak yapabilmek o kadar basit değil.

Özetle: Yukarıdaki 4 adımı kolaylaştıran her şeye sıkı sıkı sarılın, yukarıdaki 4 adımı zorlaştıran veya bu adımlara etkisi olmayan şeylerden bir an evvel kurtulun.

İşte size "Çevik Dönüşüm Yol Haritası".

Kolaylıklar.

79. Agile'ın Prototipi

Tüm Çevik yöntemler arasında XP en iyi tanımlanmış ve en eksiksiz olanıdır.

Hemen hemen tüm diğer Çevik yöntemler XP'nin bir alt kümesi veya bir varyasyonudur. Agile'ın gerçekte neyle ilgili olduğunu anlamak istiyorsanız XP'yi öğrenmek ve deneyimlemekten daha iyi bir yol yoktur.

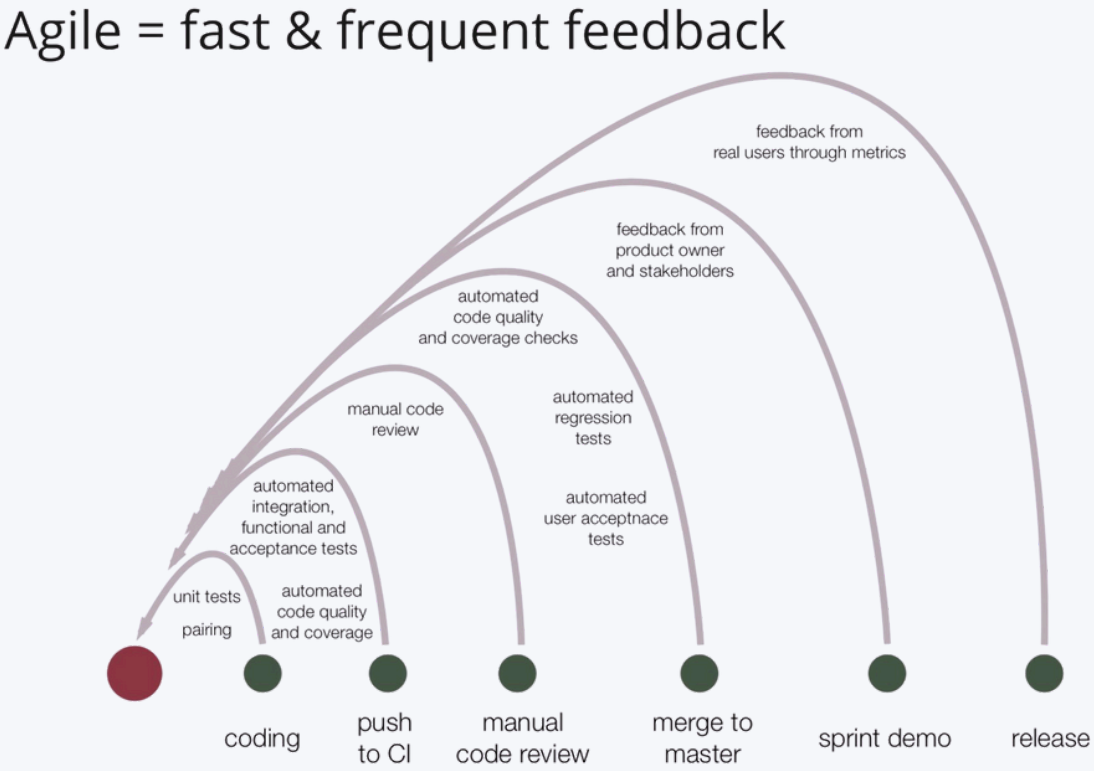
XP, Agile'ın en temel prototipi ve en iyi temsilcisidir.

Agile eğitimlerinde ve/veya danışmanlık projelerinde size XP hakkında hiç bir şey söylemeyen/söyleyemeyenlerden uzak durursanız paranızı ve zamanınızı israf etmemiş olursunuz.

Siz eğitim ve danışmanlık işlerinizle neyi farklı yapıyorsanız diye soranlara verdiğimiz cevap da budur:

“Biz gerçek çevikliğin peşinde olduğumuz için takımlarınızı XP pratikleri konusunda yetkinleştiriyoruz.”

80. Geri Bildirim Yoksa Çeviklik Yoktur



XP gibi gerçek çevik yöntemlerin içerdiği pratiklerin neredeyse tamamı önemli kararlar veren kişilere **hızlı geribildirim** sağlamaya yöneliktir. Kullandığınız pratikler bir geribildirim fırsatı yaratmıyorsa, **o pratiği tekrar gözden geçirin.**

Planlama Oyunu, Refactoring, TDD, CI, Küçük Sürümler, Toplu Sahiplik vb. pratikler geribildirimin

sıklığını ve miktarını **en üst düzeye çıkarır.**

İşlerin ne zaman yanlış gittiğini belirlememize, bunları düzeltmeye yetecek kadar **erken karar vermemize olanak tanırlar.** Daha önceki kararların sonuçları hakkında bize çokça bilgi sağlarlar.

Çevik takımlar geribildirimle gelişir. Geribildirim, çevik takımların etkili çalışmasını sağlayan şeydir.

Çevik olmak istiyorsanız görselde gösterildiği gibi olabilecek en alt seviyeden başlayarak size her seviyede geribildirim sağlayan pratikler kullanın.

Ek not: Scrum görseldeki parçalardan sadece Sprint Review / Demo etkinliğini içeren bir çerçevedir. Tek geribildirim pratiğiniz bu ise size bol şans dilerim.

81. Sendikalaşan Scrum

Türkiye'de Scrum'ın geldiği nokta işçi sendikalarının geldiği nokta ile büyük benzerlikler içeriyor.

- Görünüşte işçileri desteklemek üzere kurulan sendikalar, bir süre sonra yalnızca kendini sürdürmeye odaklanıyor.
- Görünüşte yazılım geliştirme işinde daha başarılı olma ihtimalimizi arttırmak için tasarlanmış minimal bir çerçeve olan Scrum, bir süre sonra yalnızca kendini sürdürmeye odaklanıyor.

Scrum ile çevikleşmeye çalışan her şirkette olay bir noktada şuraya evriliyor:

- Herkese dağıtılan PSM, PSPO sertifikaları,
- Scrum temalı saçma sapan etkinliklere sponsor olmalar,
- Çıktıları kimse tarafından dikkate alınmayan ancak "bol oyunlu" retrospective toplantıları...

> **"Scrum'ı Scrum için yapma" hali organizasyonları ele geçiriyor.**

"*Bireyler ve etkileşimleri süreçler ve araçlardan daha değerli görmek lazım"* demiş olan çeviklik kanaat önderlerinden bazılarının da bu durum karşısında **içi sızlıyor.**

82. Agile Öldü mü? Yoksa Dogma mı Oldu?

Popüler tartışma konusu: ***“Agile öldü mü?”***

Doğal olarak eğitim-danışmanlık işlerimizde denk geldiğimiz sektör profesyonelleri bana da soruyorlar bu tarz soruları.

- *“Agile öldü diyorlar, ne diyorsun?”*
- *“Hocam bunun modası geçmedi mi?”*
- *“Agile’dan sonra sırada ne var?“*

Cevabımı burada da paylaşayım istedim:

> ***“Agile ölmedi ama evriliyor. Daha doğrusu artık evrilmesi gerekiyor.”***

Bugün kullandığımız "çevik" yöntem ve teknikler, projelerin genellikle birkaç yıl(?) sürdüğü 1990'larda ortaya atıldı.

Bu arada (30 küsür yıl geçmiş olmasına rağmen) yaklaşımlarında neredeyse hiçbir şey değişmedi ve **“çevik” yöntem ve teknikler yerini almak üzere tasarlandıkları dogmanın kendisi haline geldiler.**

XP fanatiklerinin yerini **Scrum bağnazları** aldı, Scrum bağnazları kendi içinde savaşan gruplara bölündü ve tüm bunlar olurken insanlar işbirliğinin ve insanları sürecin önüne koymak gibi orijinal çevik değerleri gözden kaçırdılar.

İyi uygulansalar bile bu yaklaşımlar çok aylı sürüm döngüleri için optimize edilmiş şeylerdi ve artık bayatladı.

Teknoloji ilerledi: Eskiden yüzbinlerce dolara mal olan ve tedarik edilmesi aylar süren altyapı artık talep üzerine fındık fıstık parasına temin edilebiliyor. Ama biz yine de 20-30 yıl önce yaptığımız şeyleri hâlâ yapıyoruz. Ve buna hala **“son teknoloji”** diyoruz.

20-30 yıl önce geliştirilmiş bazı konseptler o yıllar için bir devrim niteliğinde olsa da bugün birçok kişi artık Scrum – Agile laflarından bıktı. Köprünün altından çok sular aktı.

Arada geçen zamanda insanlar şunları çokça gördü, deneyimledi:

- Sayısız başarısız dönüşüm.
- Etkisiz Scrum Master'lar.

Peki gerçekten sırada ne var?

Yakında aşağıdaki şeylere olan talebin azalması gerektiğini ve azalacağını düşünüyorum:

- **Organizasyonlara Waterfall'dan Agile'a dönüşüm için rehberlik eden Dönüşüm Uzmanları.** Buna artık gerek yok çünkü en küçüğünden en büyüğüne artık her yer çevik yöntemleri şu veya bu şekilde kullanıyor. Agile mainstream (ana akım) haline geldi.

- **Scrum'ın temellerini öğreten Scrum Master veya Koçlar / Eğitmenler.** 10 küsür sayfa dokümanı olan bir yöntem için içi lego oyunlarıyla doldurulmuş 2-3 günlük eğitimlere gerçekten gerek var mı?

- **Belirli çerçeveleri mekanik şekilde uygulatmaya odaklı Çevik Koçlar.** İçinde bulunduğu bağlamı anlamadan kitaptan okuduğu şeyleri çoğu zaman yanlış şekilde yorumlayarak koca koca şirketlerin yazılım geliştirme süreçlerinin içinden geçen sevgili Çevik Koçlar... Evet, bahsettiğim o koçu siz de tanıyorsunuz. Çünkü muhtemelen son 1 yıldır takımınızı /

şirketinizi çevikleştirme vaadi ile sizi içinde bulunduğunuz ortama/duruma hiç de uygun olmayan şeyler yapmaya zorlayan **“O Koç”.**

> *“Hocam danışman firmanın yolladığı çevik koç takımdan ayrıldığında gerçek çeviklik için bir şeyler yapabileceğiz inşallah”* demenize neden olan “O Koç”.

- Katı, kuralcı çerçeveler.

Peki neyden daha fazla göreceğiz?

O da bir sonraki postun konusu olsun artık.

83. Metot Polisliğinden, Teslimat Koçluğuna

Agile'ın ölmediğini ancak evrildiğini (daha doğrusu evrilmesi gerektiğini) söylemiştim. Ayrıca, yakın gelecekte bu alanda yaşanması muhtemel değişimleri (neleri daha az göreceğiz) kısaca listelemiştim.

Şimdi ise yakın gelecekte nelere olan ilginin artacağına dair bir şeyler karalayayım, yani medyumluğa devam edeyim.

Bizim alanda yakın gelecekte ben şunlara daha fazla ihtiyaç olacağını ve dolayısıyla piyasanın/insanların bu tarafa doğru evrileceğine inanıyorum (ya da inanmak istiyorum):

- Yöntem Polisliği Yerine Gerçek İhtiyaç: Scrum gibi çerçeveleri takımın gerçek ihtiyaçlarını görmezden gelerek dikte eden ve yöntem polisliği yapan koçlar yerine; takımın "gerçek" ihtiyaçlarına odaklanan ve çerçevenin içini başka şeylerle doldurabilen ve/veya gerektiğinde çerçeveyi yeniden tanımlayabilen koçlar.

- Adaptif Metotlar: Kendi bağlamınıza göre daha rahat eğip bükebildiğiniz Kanban gibi adaptif metotlar ve bu tarz metotlar içerisinde geliştirdikleri pratiklerle takımların hayatını iyileştiren insanlar.

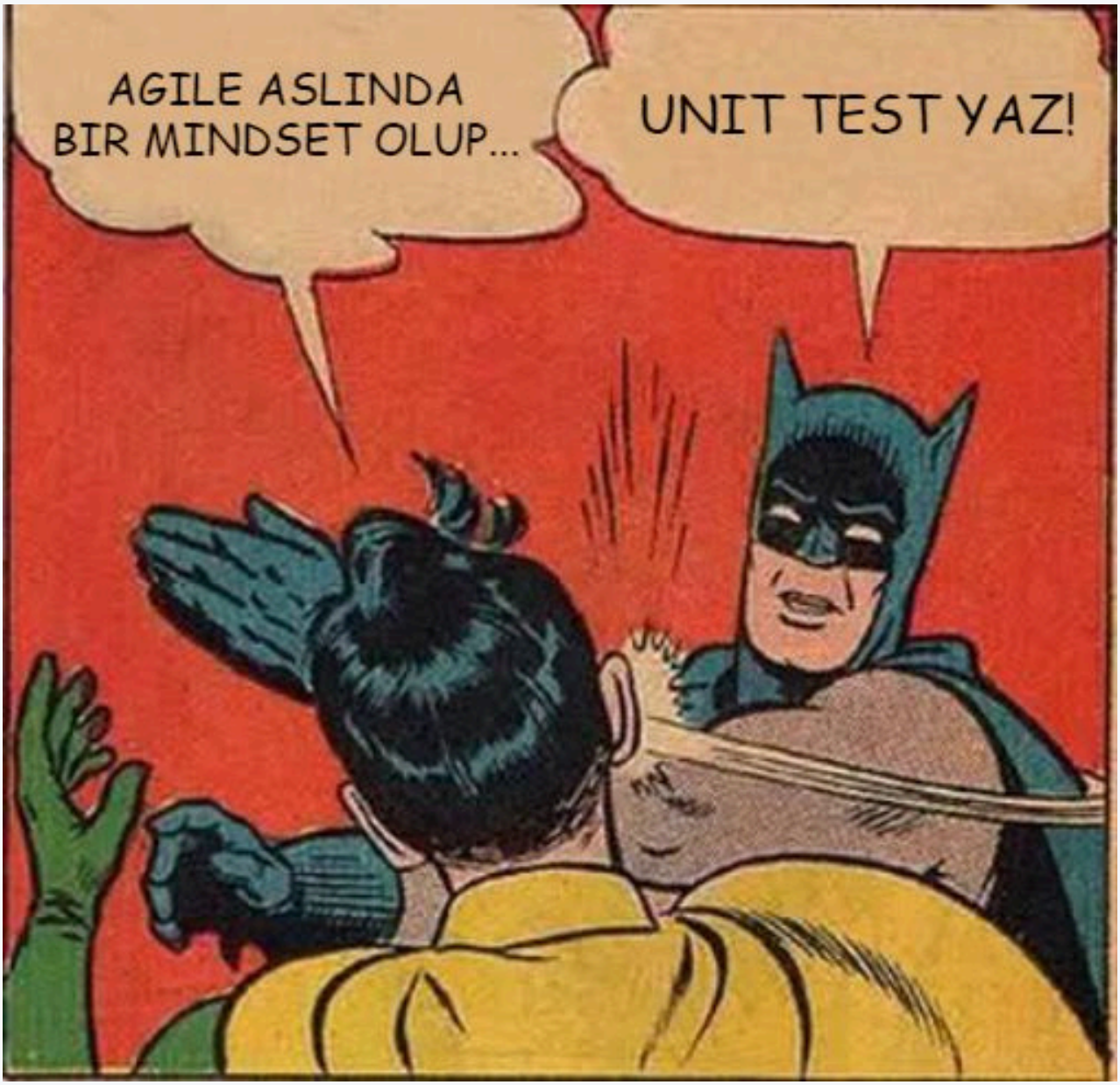
- Yeni Yaklaşımlar:Flight Levels, FaST gibi açık yaklaşımlar.

Özetle, katı çerçevelerden ve bunların polisliğinden uzaklaşarak; kuruma/takıma özel olarak hazırlanmış, bağlam odaklı bir yaklaşıma doğru geçiş olacağını bekliyorum.

Ve daha önceki gönderide bahsettiğim gibi "Agile - Scrum - Scrum Master - Agile Coach" laflarının insanlarda yarattığı bıkkınlığı ortadan kaldırabilmek için (bahsettiğim bıkkınlığın seviyesini görmek için Ekşi Sözlük'te "agile coach" anahtar kelimeleri ile bir arama yapmanız yeterli) belki bu işlere kafa yoran ve gerçekten ellerini kirleten profesyoneller olarak artık **İş Esnekliği Uzmanı ("Business Flexibility Expert")** veya **Teslimat İyileştirme Koçu ("Delivery Improvement Coach")** gibi ünvanlarla anılmamız gerektiğine inanıyorum.

Çünkü çeviklikle asıl amaçladığımız şeyler bunlar; Scrum Guide'dan ezberlediğimiz havalı cümleleri insanların üzerine fırlatmak değil.

84. Agile vs. Gerçek Dünya



85. Gerçek Performans Nerede Gizli?

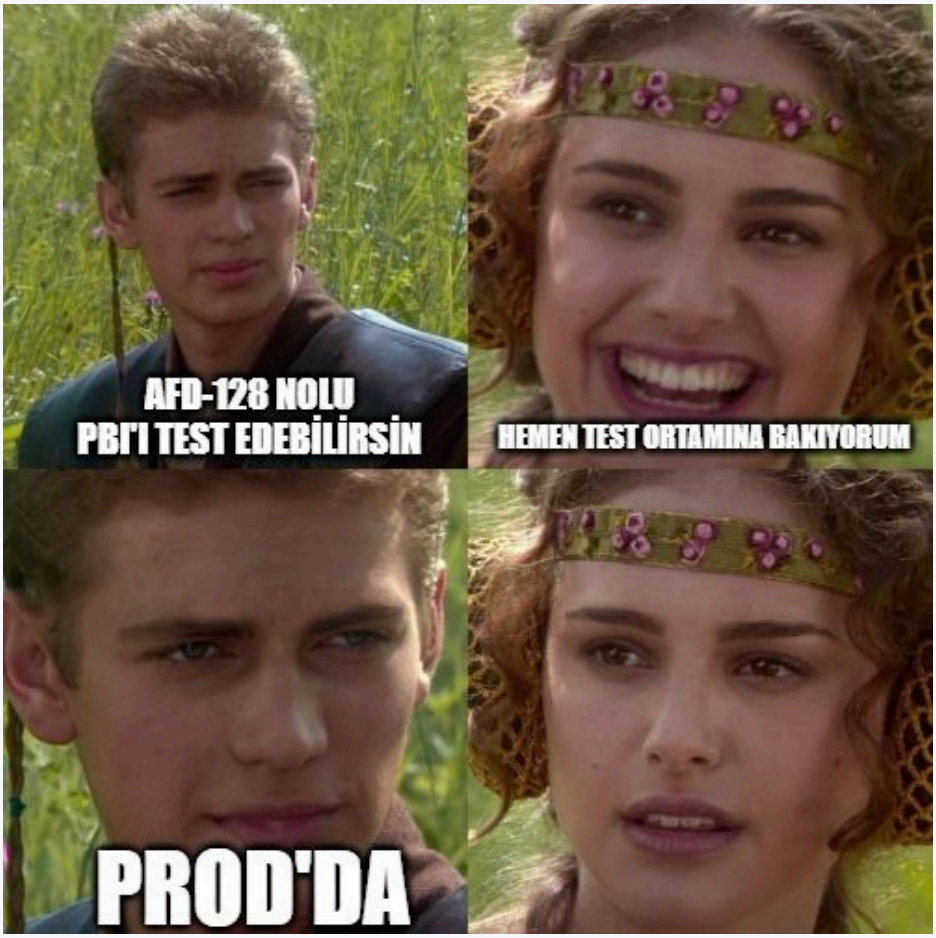
Bir ekipteki bireyleri değerlendirecek seniz gerçek katkıda bulunanların kim olduğunu anlamak için diğer takım üyelerinin (peer) geribildirimini kullanın.

- *“Bu takımda en çok kimin olmasını istiyorsunuz ve neden?”*
- *“Gelişmesine yardımcı olmak için Onur'a ne gibi tavsiyelerde bulunursunuz?”*
- *“Serkan'dan en çok ne öğrenmeyi istersiniz?”*

gibi sorular sorun.

Zeki bir yönetici, bu geribildirimlerdeki olumlu ve olumsuz kalıpları ve eğilimleri hızlı bir şekilde görecektir ve bunlar daha sonra eyleme geçirilebilir.

86. ”Aşırı Çeviklik” Yanılgısı



Yaşar Safkan’ın söylediği gibi:

> *“Olum biz zaten aşırı çeviğiz, production’da kod değiştiriyoruz, test yapıyoruz. Bundan daha çeviği yok ki.* ***Gerçek çeviklik için bizim biraz yavaşlamamız lazım.***”

87. Çevik Dönüşüm ve Kabul Kriterleri Sorumluluğu

> *"Çevik değil misiniz kardeşim, o kadar dönüşüm yaptık diyorsunuz, kabul kriterlerini de bir zahmet siz çıkarın, onu da mı ben söyleyeceğim."*

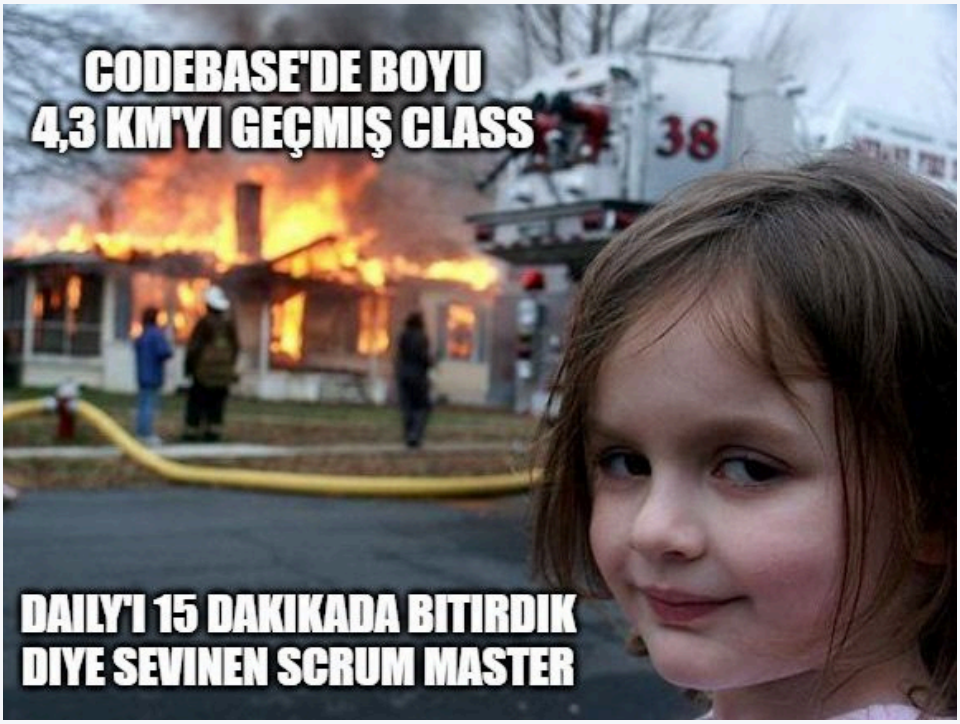
diyen PO'lar, iş birimleri, müşteriler burada mı?

88. Retrospective Toplantıları ve Takım Metrikleri

Retrospective toplantılarında oturup takım metrikleriniz hakkında konuşun.

Eğlenceli zannettiğiniz oyunlar takım üyelerine o kadar da eğlenceli gelmiyor olabilir.

89 - 90. Scrum Master Seçimi ve Tecrübe



2 ay önce işe başlayan takımdaki en junior tester arkadaş çok hevesliydi, gönüllü oldu, biz de kendisini Scrum Master yaptık.

91. Scrum Adı Altında Kanban Uygulaması

Nerede adına "Scrum" diyip aslında "Kanban" yapanlar ve bunun farkında olanlar?

92. 2024: Enkaz Toplama Yılı ve Hatalı Agile Dönüşümleri

2024 bizim için tam bir "enkaz toplama" yılı oldu. Bizden danışmanlık hizmeti talep eden pek çok kurum bizden önce bir danışmanlık şirketi ile agile dönüşüm meselesine bulaşmış ancak geldikleri son noktada dönüşüm öncesi günlerini arar hale gelmişler.

Bu kurumlardaki çalışanlar kendilerine saçma gelse de danışman firmadan gelen (iş bilmez ancak ne hikmetse yönetimin desteğini arkasına almış) insanların hiç bir dayanağı olmayan tamamen keyfi ve basmakalıp önerileri ile yazılım

geliştirme süreçlerinde değişiklikler yapmak zorunda kalmışlar.

***"Neden itiraz etmediniz?"* diye sorduğumda:**

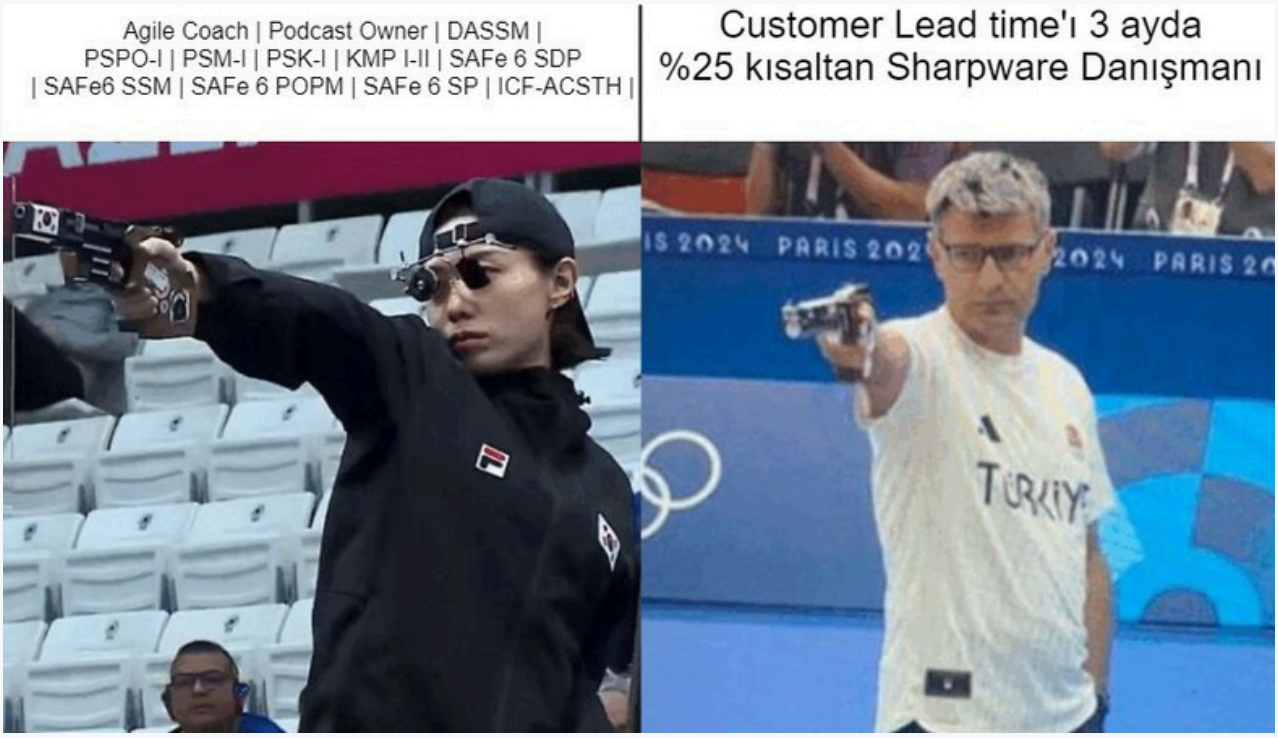
"Hocam yönetim kademesinde ciddi etkileri var, bizim sesimiz oralara ulaşmıyor." dediler.

(*"Kim bilir belki yönetiminiz de iş bilmiyordur"* dedim içimden)

Muhabbettin sonrası malum:

"Hocam eskiden gerçekten daha iyiydi bizim süreçler, şu danışmanlardan bir kurtulalım, tekrar yoluna koyacağız."

93. ’Büyük Çerçeve’ Tuzağı



Bir probleminiz var: Takımlarınız iş problemlerinizi çözecek yazılımları üretmek için çok çaba harcıyor; ancak kullanılmayan özellikler ile dolu ve ciddi kalite problemleri olan şeyler üretiyorlar. Siz bu durumu tersine çevirip doğru çözümleri, yüksek kalitede ve olabilecek en yüksek hızda üretmek istiyorsunuz.

Birileri bu probleminizi 3-4 harften oluşan isimlere sahip sertifikaları, sektörünüzden ve yazılım işinden hiç anlamayan danışmanları ve eğitmenleri, ismi havalı olan ve sevimli görünen rolleri, tuhaf jargonları ve toplantı olarak

adlandırılmayan sonsuz toplantıları olan (adı da güven veren) Büyük Bir Çerçeve ile çözmeyi öneriyor ve siz de verdikleri bol sıfırlı teklifi kabul ediyorsunuz.

Üzgünüm, artık iki probleminiz var.

94. Telepati Beklentisi

Madem ki çevik oldunuz, artık gereksinimleri de telepati yoluyla ortaya çıkarın bir zahmet, ne istediğini de iş birimi mi söylesin?

95. Araçlar Var Ama Sonuç Yok

- Scrum var.
- Jira var.
- Daily var.

Ama hâlâ işler yavaş, müşteri mutsuz ve ekip yorgun, yılgın, bıkmış...

Peki neden? Çünkü:

1. "Ne için" dönüştüğünüzü bilmiyorsunuz.
2. "Neye" dönüşmeniz gerektiği de bilmiyorsunuz.
3. Dönüşenlerin ne yaşadığını, nasıl bir süreçten

4. Bu dönüşümden ne "fayda" beklediğinizi tanımlamadınız. Daha kötüsü, bir fayda sağlanıp sağlanmadığını da ölçmüyorsunuz.
5. Danışmanlar bir şeyler söylüyor, siz de 'bize her şeyi onlar öğretir' diye düşünüp sorgulamıyorsunuz. Danışmanların sizden daha çok bildiğini varsayıyorsunuz, itiraz etmiyorsunuz. Zaten yönetiminiz de danışmanlarınızı sorgulamıyor. "Onlar ne derse öyle yapılacak" diyip, işin kolayına kaçıyor.
6. Yönetim "siz dönüşün" diyor ama kendisi eski alışkanlıklarla devam ediyor. Liderlik ortada yok.
7. Deney yapmıyorsunuz, "Bize birisi tam ne yapacağımızı söylesin, biz de ona göre çalışalım" diyorsunuz.

Unutmayın: Agile bir etiket değil, bir zihniyet dönüşümüdür. Görünür bir iyileşme yoksa, dönüşüm sadece PowerPoint'te kalır.

Binlerce dolar harcanan danışmanlık projelerinden sonra, ekiplerin ve müşterinin yüzü hâlâ bu görseldeki gibi donuksa... Orada bir sorun vardır.

Hem de büyük bir sorun.

96. Scrum Expansion Pack ve Yeni Sektör Kakafonisi

Scrum Expansion Pack yayınlandı ve yine bir kakafoni başladı.

Danışmanlar, koçlarınız aşağıdaki cümleleri üzerinize doğru fırlatıyorlarsa oradan koşarak uzaklaşın:

- "Scrum is evolving"
- "Scrum must adapt fast"
- "AI is reshaping our workflows"
- "Not Agility, AIgility"
- "Scrum Expansion Pack published"

Scrum tek başına bugüne kadar hiç bir yere gerçek çeviklik getirmemiştir ve ne kadar güncellenirse güncellensin tek başına gerçek çeviklik için yeterli olmayacaktır. AI da benzer şekilde gerçek çeviklik için sınırlı bir etki sağlayabilir.

Asıl olan müşteriyi merkeze koyarak onun için "doğru ürünü" üretmektir. Yazılımda en zor şey tam olarak ne üretileceğine karar vermektir. Geri kalan hiç bir aktivite bundan daha zor değildir.

Ürünü doğru üretmek ve hızlı üretmek bundan sonra gelir.

Doğru ürünü üretmek müşteri etkileşimi gerektirir, geribildirim gerektirir. Dolayısıyla halen en önemli şey bireyler ve onlar arasındaki etkileşimlerdir.

Sektör 20 yıl sonra Agile'dan sonra tüketebileceği yeni bir kavram buldu. Yürüyün bakalım buradan, bunun da içini boşaltırsınız 1-2 seneye.

97. ”Scrum Expansion Pack” diye başlayan cümleler vs. Batman

98. Yazılım Takımlarının Sağlığını Ölçmek: DORA Metrikleri

"Yazılım takımlarının sağlığını nasıl ölçeriz?" diye sık sorulur. Cevaplar genellikle şöyle başlar:

- *“Takım kaç iş bitirdi?”*
- *“Sprintten kaç story çıktı?”*
- *“Üretim ne kadar sürüyor, kısa mı?”*
- *“Velocity’miz arttı mı?”*

Spoiler: Bunlar fazla lokal ve yanıltıcı olabilir.

Gerçek resme bakmak istersen, DORA metrikleri

daha işe yarar:

1.Lead Time: Bir fikrin ya da sorunun fark edilmesinden, çözümün kullanıcının eline ulaşmasına kadar geçen toplam süre.

- Sadece üretim zamanınız (cycle time) değil.
- Takımın iş bitirme hızı da değil.
- Ürünü piyasaya çıkarma yolculuğunuzun ne kadar hızlı olduğu.

2. Deployment Frequency: Ne sıklıkla değer sunuyorsunuz? Küçük parçalar = Daha sık gönderim. Haftada birkaç gün idealdir. Her gün? Harika!

3.Başarısız Dağıtım Oranı: Kaç dağıtım geri alınmak zorunda kaldı? Kaç özellik "müşteri problemini" çözmedi? (Sadece teknik değil, iş değeri açısından da düşünün.)

4.Recovery Süresi (MTTR): Bir hata olduğunda ne kadar sürede toparlıyoruz? Sadece sistem hatası değil... Kullanıcıyı hayal kırıklığına uğratmak da bir hatadır.

Eğer bu metriklere ilk kez denk geliyorsanız:

- Nichole Forsgren’in “Accelerate” kitabını mutlaka okuyun.
- https://dora.dev/ sitesine göz atın.

Unutmayın: Yüksek velocity, tek başına iyi süreç demek değildir.

Gerçek başarı = “Ne kadar değerli, ne kadar kaliteli, ne kadar hızlı” ölçebilmektir.

99. İşte Bunlar Hep Mindset!

100. Agile Dönüşümünün Bir Bedeli Vardır

Geldik serinin 100. gönderisine. Ne yazsam, kime seslensem diye düşündüm ve herkese özellikle liderlere çok temel bir şeyi hatırlatayım istedim.

“Agile dönüşümünün bir bedeli vardır.”

Agile, hız kazandırır, müşteriyle bağı güçlendirir, ekipleri motive eder. Ama çoğu liderin gözden kaçırdığı bir gerçek var: Bu dönüşümün de bir bedeli vardır. Hem de sağlam bir bedeli.

İşte dönüşümün gerektirdiği temel yatırımlar:

- **Altyapı yatırımları:** Test otomasyonu, sürekli entegrasyon, release otomasyonu olmadan çeviklik sadece bir slogan olur.

- **Organizasyonu yeniden yapılandırma:** Yeni roller, çapraz fonksiyonel ekipler, farklı iş yapış biçimleri... “Eski yapıyla yeni kültür” yola devam edemez.

- **Yeni beceriler:** Dikey hikâye dilimleme, retrospektif yürütme, çevik mimari, TDD,

refactoring gibi yetkinlikler zamanla gelişir. Bunlar için ekiplere eğitim ve koçluk desteği gerekir.

- **Yeni alışkanlıklar:** Sık müşteri teması, kısa teslimatlar, ekip üyelerin kendi uzmanlıkları dışında da yetkinlikler geliştirmesi — bunlar da disiplin ister.

- **Şeffaflık acısı:** Problemler görünür hale gelir. Hem de çok hızlı bir şekilde. Neden teslim edemediğinizi hemen görürsünüz. Bu kötü bir şey değildir, ama çoğunlukla insanların konfor alanını sarsar.

Son söz:

Bu dönüşüm “Big Bang bir dönüşüm” değildir. Bir anda değil, yukarıdaki alanlarda atılacak bilinçli adımlarla olgunlaşır. Çoğu zaman bu konuda tecrübeli bir iş ortağı ile yürütülmesi gerekir. (Bu satırda ürün yerleştirme yapılmıştır: **Sharpware** 𝌆)

Melike Şahin güzel söylüyor, *"bedelini ödeyin"*. Yoksa dönüşümünüz sadece havalı laflardan ibaret olur.

101. Teknik Borç ve ”Sonra Düzeltiriz” Tuzağı

Teknik Borç (Technical Debt), kredi kartı borcuna benzer; borcunuzu ödemezseniz bir gün o kart patlar!

> "Teknik borç, çoğu zaman iş kararı olarak ertelenen teknik iştir ama çok hızlı bir şekilde ciddi bir iş problemine dönüşür."

Yöneticilerin ya da müşterilerin "Hadi bu sprint de böyle geçsin, sonra düzeltiriz" laflarına kanıp, faizini bile ödeyemeyeceğiniz bir borç batağına sürüklenmeyin.

Eğer her yeni özelliği eklemek bir öncekinden iki kat daha uzun sürüyorsa, o "sonra düzeltiriz" dediğiniz kodlar artık çürümüş demektir. Yazılım geliştirme işi ile ilgili çok sevdiğim sözlerden biridir "Later means never!"

Çevik olacağım derken iflas bayrağını çekmeyin. "Sonra" diye bir zaman dilimi yazılım dünyasında yoktur, o borcu şimdi ödeyin!

Peşin ödeyemiyorsanız en azından her iterasyon taksit taksit ödeyin.

102. Bilgi Panoları ve ”Karpuz Proje”

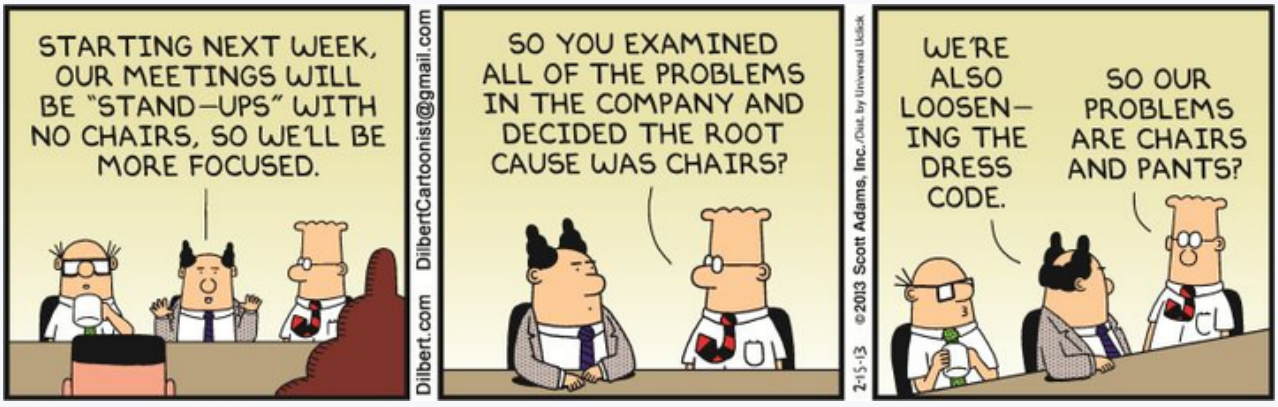
Ofis duvarlarına astığınız veya havalı ekranlarda görselleştirdiğiniz o rengarenk grafikler (Information Radiators) sadece üst yönetime "her şey yolunda" demek için oradaysa, size geçmiş olsun.

Bilgi panoları "Stark" (Çıplak) olmalıdır. Yani problemleri halının altına süpürmek için değil, kabak gibi ortaya çıkarmak için kullanılmalı.

Eğer panolarınızda sürekli "Her şey Yeşil", "Proje Harika Gidiyor" yazıyor ama takımınızdan biri çay-sigara muhabbetinde bana "abi çok kötü patlayacağız, kimse de ses çıkarmıyor" diyorsa, buna şeffaflık denmez, **"Karpuz Proje" (Dışı yeşil, içi kıpkırmızı)** denir.

Gerçek çevik takımlar, problemleri gizlemek için değil, canlarını sıksa bile herkes görsün ve çözülsün diye asarlar o grafikleri.

103. Daily Scrum ve Tekmil Verme Yanılgısı



Stand-up (Daily Scrum) toplantılarını yöneticinize "Dün şunu yaptım efendim, bugün de bunu arz edeceğim" şeklinde bir tekmil verme seansına çevirmeyin.

Bu toplantıların amacı takımın iş birliğini başlatmasıdır, yöneticinin egosunu tatmin etmesi değil.

Eğer Daily bittiğinde takım arkadaşınızın hangi problemle boğuştuğunu bilmiyorsanız ama yöneticiniz "tamam herkes çalışıyor" diye mutluysa, o toplantıyı yapmasanız da olur.

Statü raporu verecekseniz mail atın, milleti 15 dakika (ki sizde o kesin 45 dakikadır) ayakta dikmeyin.

104. User Story Yanlışları ve INVEST Prensibi

User Story (Kullanıcı Hikayesi) yazarken INVEST prensibini hiçe sayıp, Jira'ya "Veritabanında XYZ tablosunu oluştur" diye story girenler... Size de merhaba!

"Kullanıcı bunun neresinde?" diye sorunca "E developer da bir kullanıcıdır" diye felsefe yapmayın.

Hikayeleriniz **Test Edilebilir (Testable)** ve **Değerli (Valuable)** değilse, onlar hikaye değil, yapılacaklar listesidir (To-do list). Müşteriye "Login ekranının arkasındaki stored procedure'ü yazdık" derseniz size boş gözlerle bakar.

Müşterinin anlayacağı dilden konuşmuyorsanız, o kartları yırtıp atın daha iyi.

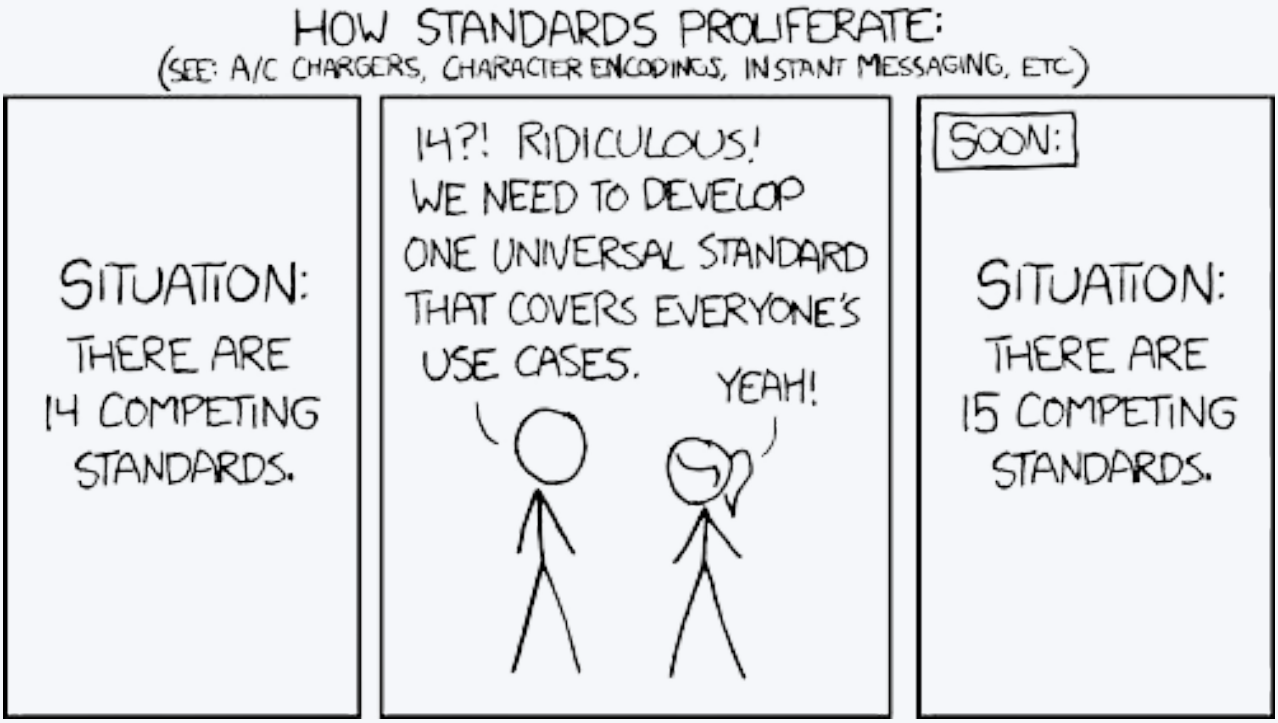
Hadi beni dinlemiyorsunuz Aşık Dertli'yi dinleyin bari:

User Story'dir bunun adı
Kullanıcıdır bunun tadı

Bizde yazan ancak anlar kendi
User bunun neresinde?

Backend'den gelir verisi
API'dandır bunun gerisi
Hey Allah'ın şaşkın analisti
User bunun neresinde?

105. Kodlama Standartları ve Kolektif Kod Sahipliği



"Bizde herkesin yoğurt yiyişi farklıdır" diyip, her geliştiricinin kendi kafasına göre kod yazdığı bir ortamda çeviklikten bahsedemezsiniz.

Kolektif kod sahipliği için olmazsa olmaz şey Kodlama Standartları'dır.

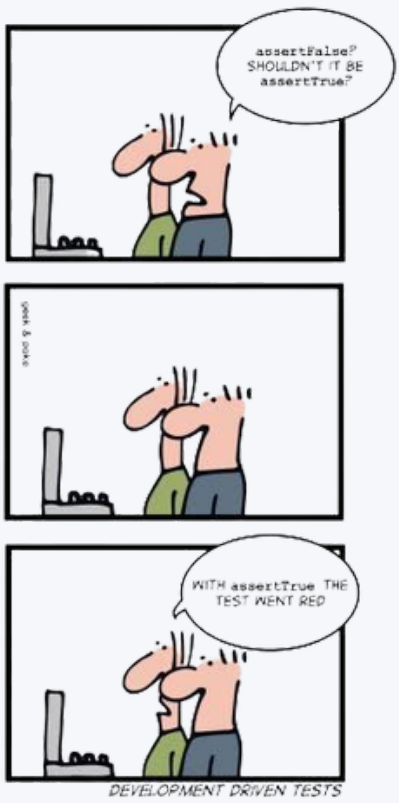
Bir dosyayı açtığınızda kodu kimin yazdığını şak diye anlıyorsanız, orada bir problem vardır. Kod, "Ahmet'in kodu" veya "Ayşe'nin kodu" gibi görünmemeli; **"Bizim takımın kodu" gibi görünmeli.**

Yoksa Ahmet askere gittiğinde o kodu düzeltecek adam bulamaz, "Ahmet gelince bakar" diye beklersiniz.

Beklemeyin. Standartlaşın.

(Sizin Ahmet inşallah askerliği kısa dönem veya bedelli yapar.)

106. Test Yazmayı Ertelemek



"Zamanımız yok, testleri sonra yazarız" cümlesi, bir yazılım projesinde duyabileceğiniz en tehlikeli yalandır.

TDD (Test Driven Development) yapmıyorsanız da en azından yazdığınız koda bir miktar unit test yazın be kardeşim!

Testleri sonraya bırakmak, paraşütü uçaktan atladıktan sonra kontrol etmeye benzer.

"Hızlı gidiyoruz" diye seviniyorsunuz ama production'da patlayan hataların üzerine story point koyup bunları velocity'e ekliyorsanız, kendinizi kandırıyorsunuz.

Kalite bir lüks değildir, hızın ta kendisidir.

107. Refactoring ve 'Bulaşık Yıkama' Analojisi

Refactoring (Yeniden Düzenleme) yapmak için yöneticisinden "Refactoring Sprint'i" dilenen takımlar...

Refactoring, kod geliştirme sürecinin doğal bir parçasıdır.

> Bulaşık yıkamak için "Bulaşık Yıkama Haftası" ilan eden bir aşçı gördünüz mü?
> Göremezsiniz, çünkü o işin bir parçasıdır.

Kodu temizlemek için özel izin istiyorsanız, o kodun sahibi siz değilsiniz demektir.

Her "Green" (başarılı test) sonrası kodu temizlemeden yeni feature eklemeye kalkmayın. Sonra o kodlar "Legacy" oldu diye ağlıyorsunuz.

108. Agile Takımları ve Yalnız Kovboylar

"*Ben sadece kendi işimi yaparım, gerisine karışmam"* diyen "Yalnız Kovboylar"... Sizin yeriniz Agile takımlar değil, Vahşi Batı.

Agile takımında *"Benim işim bitti"* diye bir şey

yoktur, ***"Bizim işimiz bitti"*** vardır.

Sprint'in son günü gelmiş, arkadaşın yetişemiyor, sen orada ayaklarını uzatıp "Benim tasklar bitti valla" diyorsan, takım oyuncusu değil, takımın önündeki engelsin demektir.

Silo mantığını kırmadan, yan yana oturan (veya Zoom'da buluşan), aynı Jira projesinde aynı Board'u kullanan ama birbirinden kopuk insanlar topluluğuna "Takım" denmez, **yan yana oturan silolar denir.**

109. Agile Dünyasında ”Bitti” Kavramı

"Bu iş bitti mi?" sorusuna; "Bitti sayılır", "Localde çalışıyor", "%99 bitti", "Sadece testi kaldı" gibi cevaplar veriyorsanız, "bitti"nin ne olduğunu anlamamışsınız demektir.

Agile dünyasında bir işin sadece iki durumu vardır: Bitti (Done) veya Bitmedi (Not Done).

Yarım yamalak yapılmış, testi geçmemiş, deploy edilmemiş işten "kredi" almaya çalışmayın.

Öğlen yemek yediğiniz yer yarı pişmiş tavuğu önünüze koyuyor mu? Koymuyor. O zaman test edilmemiş kodu da "Bitti" diye production'a itelemeyin.

Şeffaf olun. Yetişmediyse "Yetişmedi" deyin. **Dürüstlük, "mış gibi" yapmaktan iyidir.**

110. Jira, 3C Kuralı ve İletişim Sorunu

Jira issue’larının altına destanlar yazıp, yan masadaki (veya Zoom'daki) adamla konuşmaya üşenenler... Size de selam olsun!

Agile dünyasında meşhur 3C kuralı vardır: Card, Conversation, Confirmation.

Ron Jeffries der ki;

> *"Kart, gerçek şey değildir; sadece bir sonraki iletişim için verilen bir sözdür".*

Siz o kartı (Jira issue) sözleşme metni gibi her detayla doldurup, sonra da "Ama Jira’da yazmıyordu, o yüzden yapmadım" diyorsanız, siz çevik yazılım geliştirmeyi değil, **"Sorumluluktan Kaçma Sanatı"** icra ediyorsunuz.

User story’ler yazmakla ilgili değil; yüzyüze iletişim kurmakla ilgilidir.

“Writing Better User Stories” temalı eğitimler, kitaplar bu meselenin doğru anlaşılmadığına işaret eden, deli saçması şeylerdir.

Şiştim artık bunlardan.
(Bakınız görsel)

Devam edecek...